Vorkurs

Formale Methoden der Informatik

Christoph Lüders

Institut für Informatik

Wintersemester 2016/2017

26.9.2016 bis 7.10.2016

Version 3.2

Inhaltsverzeichnis

1. Intro

> Mathematicians claim that math is not a spectator sport:
> you cannot understand math, or enjoy it, without doing it.
>
> — Barbara Burke Hubbard, The World According to Wavelets

1.1. Danksagung

Für die Möglichkeit, diesen Vorkurs zu halten und seine Hilfe bei der Vorbereitung danke ich besonders herzlich Andreas Weber.

Auch möchte ich mich herzlich bei allen Tutoren bedanken: Michael Albert, Florian Nelles, Thomas Scheurich, Christian Windeck und Ivo Winkelholz.

Weiterhin danke ich Leif Thiemann und Christopher Voss für ihre Arbeit am Skript und den Übungszetteln in den vergangenen Jahren.

Teile dieses Vorkurses orientieren sich an dem Skript zur Vorlesung "Logik und diskrete Strukturen" von Heiko Röglin [Rö13] aus dem Wintersemester 2012/13. Ebenso folgen einige Abschnitte Teilen aus "Einführung in die Informatik" von Wolfgang Küchlin und Andreas Weber [KW05]. Herzlichen Dank für die Inspiration und Vorlage.

Ebenso herzlichen Dank an alle Fehlersucher und -finder! [1]

Bonn, September 2016
C. L.

1.2. Organisatorisches

Der Vorkurs findet statt von Montag, dem 26.9. bis Freitag, dem 7.10. jeweils von 10h–12h, außer am Montag, dem 3.10., da fällt er ersatzlos aus. Ort des Vorkurses ist der Hörsaal 2 in der Römerstraße 164, 53117 Bonn.

Die Übungen sind unterteilt sieben Gruppen, ebenfalls in der Römerstraße. Fünf Gruppen finden statt von 12h–14h, zwei weitere Gruppen von 14h–16h.

Die angegeben Startzeiten sind wie in der Universität üblich "c.t.", *cum tempore*, d.h. eine Viertelstunde nach der vollen Stunde. Das Gegenteil ist "s.t.", *sine tempore*, also pünktlich zur vollen Stunde.

Während des Vorkurses wird es wahrscheinlich neue Versionen dieses Skripts geben, die Fehler korrigieren oder etwas erweitert sind. Die neueste Version ist immer auf meinem Blog erhältlich oder im `precampus`-System der Uni Bonn.

[1] Sie haben trotzdem noch einen Fehler gefunden? Am Besten sagen Sie mir direkt nach der Vorlesung Bescheid. Vielen Dank!

1.3. Raison d'être

Der Vorkurs Formale Methoden der Informatik wendet sich an (kommende) Erstsemester des Bachelorstudiengangs Informatik. Der Vorkurs dient mehreren Zwecken:

- Schaffung eines einheitlichen Niveaus & Wiederholung von "Vokabeln"
- Übung des mathematischen Formalismus
- Stimulation zu Gruppenarbeit, Übung von "social skills"
- Ausblick auf einige interessante Themen der Informatik

Wir versuchen den Spagat zwischen dem Auffrischen von bereits aus der Schule bekanntem Stoff und der Präsentation von neuem, der Informatik eigenem Stoff. Wir trainieren formale Genauigkeit einerseits und geben den großen Überblick über die Breite des Anfängerstudiums andererseits.

Aufgrund dieser widerstrebenden Interessen und der Kürze der Zeit werden wir das nur zu einem gewissen Grade schaffen. Bitte bleiben Sie trotzdem dabei! Der Sinn des Vorkurses ist, Sie mit den Themen der Informatik zum ersten Mal in Berührung zu bringen. Alles, was wir hier besprechen, kommt im Laufe Ihres Studiums erneut dran und wird genauer eingeführt und ausgiebiger bearbeitet. Wenn Sie dann beim zweiten Durchgang des Themas denken, "wo war denn da das Problem?", hat der Vorkurs seinen Sinn erfüllt.

Lassen Sie sich aber bitte auch nicht abschrecken, falls gewisse Themen des Vorkurses Ihnen zu einfach erscheinen. Nicht alle Erstsemester haben den gleichen Hintergrund und damit das gleiche Wissen. Die Informatik vereint viele verschiedene Aspekte auch anderer Wissenschaften und wir hoffen, dass für Jede und Jeden in diesem Vorkurs genug Neues und Interessantes zu finden ist.

Selbst, wenn Ihnen dieser Vorkurs leicht fällt, lassen Sie sich nicht täuschen: das Niveau und die Intensität des Lernens an der Universität sind nicht mit denen der Schule zu vergleichen. Daher hören Sie lieber den gleichen Stoff doppelt, als ihn zu verpassen und möglicherweise ein Modul wiederholen zu müssen.

1.4. Selbsthilfe

Wie Jürgen Fohrmann, Rektor unserer Universität von 2009-2015, bei der Absolventenfeier 2014 sagte, ist das Ziel jedes Studiums *Bildung in einem bestimmten Fachbereich.* Dazu ist meist das Erlernen von Wissen erforderlich, welches später in Prüfungen abgefragt wird. *Wie* Sie dieses Wissen erwerben, ist dabei eher unwesentlich und zudem von Person zu Person sehr unterschiedlich. Nutzen Sie alle Möglichkeiten, die sich Ihnen bieten, nicht nur die Vorlesungen, Übungen und Literatur und finden Sie heraus, wie Sie am besten lernen können.

Einige sinnvolle Hilfsmittel könnten für Sie sein:

- Dieses Skript: schauen Sie zumindest mal drüber, bevor Sie zur dieser Vorlesung gehen. Und wenn die Vorlesung dann läuft, können Sie auch auf einen der vielen Links klicken (alle blauen Texte sind externe Links), wenn Sie mehr zu einem Thema wissen wollen. Oft verlinkt es auf Wikipedia, siehe den nächsten Punkt.

- Wikipedia: Muss man dazu noch mehr sagen? Lesen Sie aber auch mal die englische Wikipedia. Die Inhalte und Qualität sind nicht immer wie in der deutschen, oft kann man einiges mehr oder anders lernen.
- Es gibt sehr gute Foren im Netz. Zum Beispiel hat Stack Overflow für Fragen rund ums Programmieren oder Mathematics Stack Exchange für Fragen zur Mathematik eine hohe Qualität. Ansonsten ist natürlich Google immer wieder die erste Anlaufstelle.
- Nutzen Sie natürlich auch die Bibliotheken der Universität. Bücher zum Thema Informatik stehen in der "Abteilungsbibliothek für Medizin, Naturwissenschaften und Landbau", Nußallee 15a, 53115 Bonn. Die Öffnungszeiten sind sehr leger: Montag–Sonntag, 8:00–24:00 Uhr. Es gibt in den Bibliotheken große Lesesäle, in denen man in Ruhe lesen und arbeiten kann. Die Lehrbuchsammlung hält von den Standardwerken viele Exemplare zum Ausleihen bereit. Die ULB hat auch eine Facebook-Seite und einen Twitter-Account!

  In der Römerstraße und im LBH, Raum E.15 stehen den Studierenden in den Fachschaftsräumen Handapparate mit wichtiger Grundlagenliteratur zur Verfügung, siehe auch hier. Die Bücher der Handapparate können nur vor Ort eingesehen werden.

  Sie wollen vorher wissen, ob und wo ein Buch verfügbar ist (es gibt ja noch andere Bibliotheken der Uni)? Nutzen Sie bonnus, das Suchportal der Uni online.
- Vielleicht wollen Sie Ihre Aufzeichnungen direkt schön im Computer setzen? Dann nutzen Sie das TEX/LATEX System. Es erzeugt ausgesprochen schöne Dokumente, ist kostenlos und früher oder später müssen Sie es sowieso lernen. Dieses Skript ist mit MiKTeX für Windows erstellt worden. Andere Betriebssysteme werden auch unterstützt, suchen Sie einfach im Netz nach "`latex mybrandofoperatingsystem`".[2]
- Wolfram Alpha: kann gut rechnen, auch symbolisch.
- Als kostenlose Alternative zu teuren Computer Algebra Systemen wie Maple oder Mathematica bietet sich Sage an, auch online als *Sage cell server*. Sage programmiert sich in Python, das könnte sich als hilfreich erweisen.
- Kennen Sie den Google Graph Plotter? Geben Sie mal bei Google "`sin(e^x)`" ein!
- Sie wollen das Programmieren in C/C++ oder Python beginnen? MinGW für Windows ist ein GNU C/C++, ADA und FORTRAN (!) Compiler, der ebenso in Cygwin verfügbar ist. Wer Linux hat, hat `gcc` wahrscheinlich schon auf dem Rechner. Unter Windows ist Microsoft Visual Studio 2015 für C/C++, C#, Visual Basic und F# kostenlos und sehr leistungsfähig. Nicht zuletzt kann man Python völlig frei laden und benutzen. Prima Sprache!

  In Anhang E finden Sie eine Liste von Problemen und weitere Links, falls Sie sich üben wollen.
- Sie programmieren gerne oder arbeiten lange Zeit an den gleichen Dateien, die Sie immer weiter verändern (wie z.B. ein LATEX-Dokument)? Verwalten Sie Ihre Dateien mit einem Sourcecode Management System wie Subversion, Mercurial oder Git. Sie können damit jederzeit sehen, wann Sie welche Änderung gemacht haben, können gleichzeitig mit vielen Anderen an Ihren Dateien arbeiten und haben obendrein ein Backup mit unendlich vielen Generationen.

[2] LATEX kann einen in den Wahnsinn treiben. Aber das kann Word auch, habe ich mir sagen lassen. Sollten Sie LATEX benutzen, werden Sie tex.sx lieben lernen. Eine gute Einführung findet sich in The Not So Short Introduction to LATEX 2_ε und Wikibooks LaTeX hat viele einfache Beispiele.

- Und wenn Sie jetzt schon so eifrig programmieren, dann vergessen Sie nicht, Test Code zu schreiben. Am besten schon von Anfang an.
- Fragen zum Uni-Betrieb, Ärger mit dem Dozenten, Probleme mit dem Stoff? Die Fachschaft Informatik weiß Rat.
- Ein letzter Tipp: Gehen Sie zum Uni-Sport! Es gibt dort fast alles (von Aikido bis Zumba), es kostet nichts oder fast nichts, es macht Spaß und Sie sitzen sowieso genug am Schreibtisch.

1.5. Literatur

Teile dieses Vorkurses orientieren sich an der Vorlesung "Logik und diskrete Strukturen" von Heiko Röglin aus dem WS 2012/13 [Rö13].

Aussagenlogik, die Definition von Termen und O-Notation finden sich auch in "Einführung in die Informatik" von Küchlin und Weber, [KW05].

Eine schöne Übersicht über mathematische Sprache und Symbolik findet sich in dem PDF "Einführung in Sprache und Grundbegriffe der Mathematik" von Markus Junker von der Universität Freiburg [Jun10].

Eine etwas tiefere Einführung in die Mathematik mit vielen Aufgaben und Lösungen hält der "Vorkurs Mathematik" von Georg Hoever bereit [Hoe14].

Immer wieder gute Dienste leistet die "kleine Enzyklopädie Mathematik" [KEM80], wird aber leider nicht mehr aufgelegt. Sie lässt sich jedoch noch gebraucht kaufen.

Schön zu lesen und mit vielen interessanten Beispielen ist auch "Mathematics for Computer Science" von Eric Lehman und Tom Leighton [LL04], per Download im Internet zu finden.

2. Mathematische Sprache

> Stimmen die Namen und Begriffe nicht, so ist die Sprache konfus.
> Ist die Sprache konfus, so entstehen Unordnung und Mißerfolg.
> [...] Darum muß der Edle die Begriffe und Namen korrekt benutzen
> und auch richtig danach handeln können.
>
> — Konfuzius, Gespräche, Buch XIII, 3.

Der Sinn mathematischer Symbolik ist, einen Sachverhalt *exakt* auszudrücken. Wir bedienen uns dazu spezieller mathematischer Symbole und einer speziellen mathematischen Sprache.

Die Aussage "x ist kleiner zehn" mag auf den ersten Blick klar erscheinen, es stellen sich aber bei genauerer Betrachtung mehrere Fragen:

- Meinen wir nur ganze Zahlen oder Brüche oder noch was anderes?
- Sind negative Zahlen auch gemeint?
- Genau 10 oder nur so ungefähr?

Um solche Unklarheiten zu vermeiden, verwenden wir eine genaue Schreibweise von klar definierten Symbolen. Leider ist selbst in der Mathematik "klar definiert" nicht immer ganz klar. So gibt es zum Beispiel verschiedene Auslegungen zu dem Begriff der "natürlichen Zahl". Solche Unklarheiten werden dann z.B. durch ein Symbolverzeichnis (siehe Anhang A) eines Buches geklärt.

Trotzdem ist mathematische Sprache wesentlich genauer als natürliche Sprache. Wichtig für Sie zu lernen ist zweierlei:

1. Wie drücke ich mich klar in dieser Sprache aus? Unser Beispiel schreiben wir klarer so: "Sei $x \in \mathbb{R}$ mit $x < 10$".
2. Es bleibt trotz alledem Sprache, also ein Mittel der Kommunikation. Es sollte kein blinder Formalismus werden. Wenden Sie sich an den Leser, um Ihre Gedanken möglichst einfach und klar darzustellen.

Zu üben, sich zwischen diesen beiden Punkten zu bewegen, ist unter anderem auch Ziel dieses Vorkurses.

Mathematische Sprache ist typischerweise nicht sehr schön, im Sinne von eloquent. Schlimmer noch, sie ist oft sehr repetitiv, langweilig und variantenarm. Das ist leider der Sinn der Sache, da es für uns sehr sinnvoll ist, immer die gleichen Wörter zu nutzen, die wir vorher hoffentlich einmal definiert haben. Nur so können wir uns exakt ausdrücken.

In diesem Skript stehen die englischen Fachbegriffe immer in Klammern hinter den deutschen, da Sie häufig auch englische Texte lesen werden und wer kommt schon auf die Idee, dass ein *Körper* im Englischen *field* heißt? Überhaupt sollten Sie sich um ein gutes Englisch bemühen in der Reihenfolge: Lesen, Hören, Schreiben, Sprechen, da wenige Aufgaben für Informatiker vorstellbar sind, in denen das nicht wichtig sein wird. Zu diesem Thema gibt es auch Kurse der Uni.

Anhang B enthält eine Tafel der griechischen Buchstaben, die oft in mathematischen Texten vorkommen (man hat sonst einfach zu wenige Buchstaben). Sie erleichtern sich das Lesen,

Sprechen und sogar das Verständnis der Texte, wenn Sie die Buchstaben benennen und aussprechen können.

Es gibt viele normale Worte, die in der Mathematik eine genau definierte Bedeutung haben. Im Laufe des Vorkurses werden wir davon einige kennen lernen, wie z.B. "geordnetes Paar", "genau dann, wenn", "beliebig, aber fest", "fast alle", "trivial" oder "ohne Beschränkung der Allgemeinheit".

Im Folgenden führen wir einige Vokabeln und Schreibweisen ein, die dann später mit weiterem Inhalt gefüllt werden. Im Moment geht es uns nur um die mathematische Sprache.

2.1. Term, Gleichung, Ungleichung

Als *Term* bezeichnen wir wohlgeformte mathematische Ausdrücke, die aus Zahlen, Unbestimmten, Klammern und Operatoren ($+$, $-$, $\cdot$, $:$) bestehen. Sie bilden die gültigen Worte der mathematischen Sprache. D.h., ein Term enthält kein Gleichheits- oder Ungleichheitszeichen.

Zwei Terme, welche durch die Vergleichsoperation "$=$" verbunden sind, nennen wir *Gleichung* (engl. *equation*).

Beispiel 2.1 (Pythagoras)**:** Seien $a, b, c \in \mathbb{R}$ die Seitenlängen einen rechtwinkligen Dreiecks, wobei die Seite der Länge c gegenüber des rechten Winkels liegt. Dann gilt die Gleichung

$$a^2 + b^2 = c^2.$$

Sowohl "$a^2 + b^2$" als auch "c^2" sind Terme, aber auch nur "a^2" ist ein Term. Kein Term dagegen ist "$a^2 +$" (nicht wohlgeformt, da das rechte Argument für "$+$" fehlt). $\lhd$ [3]

Terme, welche einen Vergleichsoperator wie "$<$", "$\leq$", "$>$", "$\geq$" oder "$\neq$" beinhalten, nennen wir *Ungleichungen* (engl. *inequation*).

Beachten Sie, dass sich beim Umformen einer Ungleichung durch Multiplikation oder Division mit einer negativen Zahl das Ungleichheitszeichen umdreht!

2.2. Rechengesetze

Vorab seien einige Rechengesetze wiederholt, die Sie aus $\mathbb{R}$ kennen. In Kapitel 8 werden wir algebraische Strukturen kennenlernen, für die manche dieser Gesetze nicht gelten.

Seien $a, b, c \in \mathbb{R}$. Dann gelten folgende Rechenregeln:

$$\begin{aligned}
(a+b)+c &= a+(b+c), && \text{(Assoziativität der Addition)}\\
(a\cdot b)\cdot c &= a\cdot(b\cdot c), && \text{(Assoziativität der Multiplikation)}\\
a+b &= b+a, && \text{(Kommutativität der Addition)}\\
a\cdot b &= b\cdot a, && \text{(Kommutativität der Multiplikation)}\\
(a+b)\cdot c &= a\cdot c + b\cdot c. && \text{(Distributivität)}
\end{aligned}$$

Diese Regeln folgen (wie wir in Abschnitt 9.5 sehen werden) aus der Tatsache, dass $\mathbb{R}$ ein Körper ist.

Achten Sie auf diese "Vokabeln". Diese Worte kommen immer wieder vor.

[3] Dieses Zeichen benutzen wir, um das Ende eines Beispiels zu markieren.

2.3. Konventionen

Es gilt als Konvention, dass das Rechenzeichen "·", welches oft für die Multiplikation steht, nicht geschrieben werden muss.

D.h., folgende Terme sind gleich:

$$2 \cdot a = 2a,$$
$$a \cdot b \cdot c = abc.$$

Im Zusammenspiel mit dem Zeichen "+", welches oft für die Addition steht, gilt Punkt-vor-Strichrechnung. D.h., ohne Angabe von Klammern bindet das Zeichen "·" stärker als das Zeichen "+":

$$2 \cdot a + b = (2 \cdot a) + b,$$
$$a + b \cdot c = a + (b \cdot c).$$

Das kennen Sie alle aus der Schule. Wichtig zu wissen ist hier, dass das Ganze eine *syntaktische Konvention* ist, d.h., es gilt auch, falls die beiden Zeichen für etwas Anderes stehen als Multiplikation und Addition. Das wird uns in Kapitel 8 beschäftigen.

2.4. Summen- und Produktschreibweise

Zur Addition von mehreren Summanden, die man abhängig von einer *Index-* oder *Laufvariable* beschreiben kann, benutzt man gerne das *Summenzeichen* $\sum$. Die Laufvariable nimmt alle ganzzahligen Werte von ihrem Startwert bis zum Endwert an, inklusive dieser beiden.

Beispiel 2.2:

$$1+2+3+4+5+6+7+8+9+10 = \sum_{i=1}^{10} i,$$
$$1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} = \sum_{i=0}^{4} \frac{1}{2^i}.$$

◁

Beachten Sie, dass der Name der Indexvariable (hier i) keinen Einfluss auf das Summe hat. Es handelt sich um eine *gebundene Variable*. [4]

Wenn der Anfangswert der Indexvariablen größer ist als sein Endwert, dann ist die Summe leer und ihr Wert ist 0:

$$\sum_{i=n+1}^{n} i = 0.$$

[4] Für die Programmierer unter Ihnen: in C/C++ könnte man eine Summe so schreiben:

```
sum = 0;
for (int i = start; i <= end; ++i)
   sum += term(i);
```

Beachten Sie, dass `i` ein Integer ist und jeweils um 1 erhöht wird und dass die obere Grenze in der Schleife auch durchlaufen wird (`i <= end`).

Manchmal ist hier der Gebrauch von Klammern ratsam, da sonst nicht klar ist, was alles summiert wird. Was meint wohl $\sum_{i=0}^{n} i - 1$? Wollte der Autor $\sum_{i=0}^{n}(i-1)$ oder $(\sum_{i=0}^{n} i) - 1$ sagen?

Ebenso gibt es das *Produktzeichen* $\prod$ zur Darstellung von Produkten aus mehreren Faktoren.

Beispiel 2.3:

$$1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 = \prod_{i=1}^{6} i$$

◁

Beachten Sie die Punkt-vor-Strichrechnung! Da das $\prod$-Zeichen eine Abfolge von Faktoren darstellt, müssen Sie klammern, falls die Faktoren Additionen oder Subtraktionen enthalten.

Beispiel 2.4:

$$(a_1 - 1)(a_2 - 2)(a_3 - 3) = \prod_{i=1}^{3}(a_i - i) \neq \prod_{i=1}^{3} a_i - i$$

◁

Auch hier gibt es ein leeres Produkt, welches den Wert 1 hat:

$$\prod_{i=n+1}^{n} i = 1.$$

Summen- und Produktzeichen werden uns wieder begegnen in Abschnitt 9.3. Ähnliche Schreibweisen für andere Operationen lernen wir schon in Kapitel 3 kennen.

3. Logik

Young man, in mathematics you don't understand things.
You just get used to them.

— John von Neumann [5]

3.1. Aussagenlogik

Mit Hilfe der Aussagenlogik (engl. *propositional calculus*) können wir Elementaraussagen verknüpfen und auf ihren Wahrheitswert untersuchen. Elementaraussagen sind wahr oder falsch und nicht weiter zerlegbar. [6] [7]

Definition 3.1 (Aussage)**:** *Nach Aristoteles [8] ist eine Aussage (engl.* proposition*) in unseren Sinne ein sprachliches Gebilde, von dem es sinnvoll ist, zu fragen, ob es* wahr *(engl.* true*) oder* falsch *(engl.* false*) ist.*

Man nennt dies *zweiwertige Logik*: jede Aussage ist entweder wahr oder falsch, dies ist ihr *Wahrheitswert* (engl. *truth value*).

Beispiel 3.1: Einige Aussagen:

1. Alle Studierenden sind Menschen.
2. Alle Menschen sind Studierende.
3. Es gibt Außerirdische.
4. Es gibt unendlich viele Primzahlen.
5. Es gibt unendlich viele Primzahlzwillinge (zwei Primzahlen, deren Differenz 2 ist).

Man kann zeigen, dass die Aussagen 1 und 4 wahr sind. Aussage 2 ist jedoch falsch, solange auch nur ein Mensch existiert, der kein Student und keine Studentin ist. Über den Wahrheitswert der 3. und der 5. Aussage können wir zum heutigen Zeitpunkt kein Urteil abgeben, wir wissen es nicht.

Keine Aussagen im mathematischen Sinne sind:

1. Bitte komm nach Hause. (Was könnte hier wahr oder falsch sein?)
2. Wie geht's?
3. Du bist böse. (Dies ist eine moralische Äußerung)
4. Groß. (Das ist nur ein Wort, es ist nicht wahr oder falsch)
5. Colorless green ideas sleep furiously. (Noam Chomsky, 1957: Ein grammatikalisch korrekter, aber unsinniger Satz. Er ist weder wahr noch falsch)

◁

[5] Amerikanischer Mathematiker, Physiker und Informatiker ungarischer Abstammung, 1903–1957
[6] Weiterführend und vertiefend siehe [KW05, Kap. 16].
[7] Wenn Sie mehr möchten: Vorlesung über Aussagenlogik als Video von Christian Spannagel von der PH Heidelberg.
[8] Griechischer Philosoph und Schüler des Platon, 384–322 v. Chr.

3.2. Operationen auf Aussagen

Im Folgenden seien A und B Aussagen. Wir sagen "es gilt A" oder "A gilt nicht". Eine Aussage hat immer einen der Werte wahr oder falsch. Wir stellen die Werte wahr und falsch häufig auch als 1 und 0 (oder w/f oder T/F) dar.

Wahrheitswerte können wir durch verschiedene Operationen (deren Operatoren heißen Junktoren) miteinander verknüpfen (die dadurch eine boolesche Algebra[9] bilden können). Da es nur endlich viele mögliche Werte gibt, können wir diese einfach alle auflisten. Dadurch entsteht eine Wahrheitstabelle (engl. *truth table*). Ein Operator ist durch seine Wahrheitstabelle eindeutig bestimmt.

Die einfachste Operation ist die *Negation*, oft auch als `NOT` bezeichnet: das Gegenteil einer falschen Aussage ist eine wahre Aussage und ebenso ist das Gegenteil einer wahren Aussage eine falsche Aussage (tertiam non datur). In der Schreibweise der Logik wird für die Negation das Zeichen "¬" verwendet, gesprochen "nicht". Häufig schreibt man auch $\overline{A}$ statt $\neg A$. Die Wahrheitstabelle für die Negation sieht folgendermaßen aus:

A	$\neg A$
f	w
w	f

Die *Konjunktion* (auch `AND` oder "Und") ist wahr, falls *beide* Teilaussagen wahr sind; ansonsten ist sie falsch. Zum Beispiel bedeutet "die Tür kann geöffnet werden, wenn der Schlüssel gedreht wurde *und* die Klinke gedrückt wurde", dass eine der beiden Aktionen alleine nicht ausreichend ist. Das mathematische Symbol für `AND` ist "∧". Die Wahrheitstabelle sieht so aus:

A	B	$A \wedge B$
f	f	f
f	w	f
w	f	f
w	w	w

Die *Disjunktion* (auch `OR` oder "(inklusives) Oder") ist wahr, falls *mindestens eine* der beiden Teilaussagen wahr ist. Beispielsweise sagt der Satz "Ich komme nach Hause, wenn es regnet *oder* dunkel wird" aus, dass einer der beiden Gründe ausreichend ist. **Vorsicht!** Unser normalsprachliches "Oder" ist meist ein *exklusives* Oder (siehe unten). Der Satz "Trinkst du Bier *oder* Wein?" bedeutet eben meist, dass man nicht beides möchte.

Das mathematische Symbol für `OR` ist "∨" und dies ist seine Wahrheitstabelle:

A	B	$A \vee B$
f	f	f
f	w	w
w	f	w
w	w	w

[9] nach George Boole, englischer Mathematiker, 1815–1864

Die *exklusive Oder* (`XOR`, auch: die Kontravelenz) ist wahr, falls *genau eine* der beiden Teilaussagen wahr ist. Das mathematische Symbol dafür ist nicht eindeutig, wir verwenden hier "$\oplus$". $A \oplus B$ wird ausgesprochen "entweder A oder B" oder einfach "`XOR`" oder "`EXOR`".

A	B	$A \oplus B$
f	f	f
f	w	w
w	f	w
w	w	f

Wie die `XOR`-Wahrheitstabelle zeigt, ist `XOR` selbstinvers, das heißt "$A \oplus A = \mathsf{f}$". Man kann `XOR` auch aus `AND`, `OR` und `NOT` zusammensetzen:

$$A \oplus B = (A \wedge \neg B) \vee (\neg A \wedge B), \qquad \text{alternativ:}$$
$$A \oplus B = (A \vee B) \wedge \neg(A \wedge B).$$

Die *Implikation* oder *Folgerung* ist dann wahr, wenn aus der ersten Aussage die zweite folgt. Aus einer falschen Aussage darf sowohl etwas Falsches oder Wahres folgen (*ex falso quodlibet*: jeder Schluss aus Falschem ist zulässig); aus einer wahren Aussage darf aber nur etwas Wahres folgen. Mit anderen Worten: aus einer wahren Aussage darf nie etwas Falsches folgen, alles andere ist erlaubt. Man sagt: "aus A folgt B" oder "wenn A, dann B". Das Symbol ist "$\Rightarrow$" und dies ist die Wahrheitstabelle:

A	B	$A \Rightarrow B$
f	f	w
f	w	w
w	f	f
w	w	w

Auch die Implikation lässt sich mit einfacheren Operationen ausdrücken:

$$(A \Rightarrow B) = \neg A \vee B.$$

Man sagt auch "A ist eine *hinreichende* Bedingung für B". Das bedeutet, wenn A vorliegt, dann folgt daraus auch B.

Davon ist zu unterscheiden, dass A eine *notwendige* Bedingung für B ist. Das bedeutet, dass es kein B gibt ohne A. A ist also eine *conditio sine qua non*, eine Bedingung, ohne die es nicht geht. Formal schreiben wir $B \Rightarrow A$ oder $A \Leftarrow B$.

Vorsicht! Die *logische* Implikation, wie hier geschildert, kann unserer natürlichen Sprache widersprechen und Zusammenhänge nahelegen, die keine sind. Man nennt das auch die *Paradoxien der materialen Implikation*. Die logische Aussage "Wenn London in England liegt, dann ist ein Fuchs ein Säugetier" ist logisch wahr, aber es existiert kein kausaler Zusammenhang, obwohl es so klingt. Schlimmer noch: "Wenn London in Frankreich liegt, dann ist ein Fuchs ein Säugetier" ist formal ebenfalls wahr!

Als letztes bleibt noch die *Äquivalenz*. Das mathematische Symbol ist "$\Leftrightarrow$" und man sagt: "A genau dann, wenn B", "A dann und nur dann, wenn B" oder "A ist äquivalent zu B" (gelegentlich auch abgekürzt als "gdw." und in englischen Texten manchmal geschrieben als "iff", mit zwei "f"). Es bedeutet, dass beide Teilaussagen immer zur gleichen Zeit wahr oder falsch sind. Zum Beispiel: "eine ganze Zahl heißt *gerade* genau dann, wenn sie ohne Rest durch 2 teilbar ist". Die Wahrheitstabelle dazu ist:

A	B	$A \Leftrightarrow B$
f	f	w
f	w	f
w	f	f
w	w	w

Man sagt auch, dass A notwendig und hinreichend für B ist, daher auch die Schreibweise. In Formeln: $(A \Leftrightarrow B) = (A \Rightarrow B) \wedge (A \Leftarrow B)$.

Zum Beschreiben von aller zweiwertigen Operationen reichen die Operationen `AND`, `OR` und `NOT`. [10] Daher konnten wir aus ihnen die anderen Operationen aufbauen. Allgemein kann man jede n-wertige boolesche Operation aus diesen drei basishaften Operationen aufbauen, z.B. mit der disjunktiven Normalform (DNF). Die erste Formel für `XOR` ist in DNF.

Es gibt eine Rangfolge der Operatoren (engl. *operator precedence*), die angibt, welcher Operator stärker bindet. Ohne diese wäre die Aussage $A \vee B \wedge C$ mehrdeutig, könnte sie doch $(A \vee B) \wedge C$ oder $A \vee (B \wedge C)$ bedeuten. Die Rangfolge der Operatoren von stark nach schwach bindend ist:

- Klammern (sind kein Operator),
- Negation, $\neg$,
- Konjunktion, $\wedge$,
- Disjunktion, $\vee$,
- Implikation, $\Rightarrow$,
- Äquivalenz, $\Leftrightarrow$.

Damit ist $A \Rightarrow B \Leftrightarrow \neg A \vee B$ immer wahr und sieht geklammert so aus: $(A \Rightarrow B) \Leftrightarrow ((\neg A) \vee B)$. Es ist aber besser, zu viele Klammern zu setzen als zu wenige, wenn dadurch das Verständnis erleichtert wird.

3.3. Gesetze für Aussagen

Wir nennen zwei Aussagen A und B *äquivalent*, wenn sie unter allen Belegungen denselben Wahrheitswert annehmen, das heißt, wenn ihre Wahrheitstabellen identisch sind. Wir schreiben dann $A \Leftrightarrow B$.

Nach der Definition der Äquivalenz ist dann die Aussage $A \Leftrightarrow B$ immer wahr und wir nennen sie eine *Tautologie* oder *allgemeingültig*. Eine einfache Aussage, die immer wahr ist, ist z.B. $A \vee \neg A$.

Das Gegenteil wäre eine *Kontradiktion* oder ein *Widerspruch*. Das ist eine Aussage, die immer falsch ist. Ein Beispiel dafür ist $A \wedge \neg A$.

In der folgenden Tabelle sind einige Äquivalenzen von Aussagen aufgeführt.

[10] Sie sind hinreichend, aber auch notwendig! Mit weniger Operationen geht es nicht.

Konstanz:	$(A \wedge \neg A) \Leftrightarrow \mathsf{f}$ $(A \vee \neg A) \Leftrightarrow \mathsf{w}$ $(A \oplus A) \Leftrightarrow \mathsf{f}$
Doppelte Negation:	$(\neg\neg A) \Leftrightarrow A$
Assoziativität:	$((A \vee B) \vee C) \Leftrightarrow (A \vee (B \vee C))$ $((A \wedge B) \wedge C) \Leftrightarrow (A \wedge (B \wedge C))$ $((A \oplus B) \oplus C) \Leftrightarrow (A \oplus (B \oplus C))$
Kommutativität:	$(A \vee B) \Leftrightarrow (B \vee A)$ $(A \wedge B) \Leftrightarrow (B \wedge A)$ $(A \oplus B) \Leftrightarrow (B \oplus A)$ $(A \Leftrightarrow B) \Leftrightarrow (B \Leftrightarrow A)$
Idempotenz:	$(A \vee A) \Leftrightarrow A$ $(A \wedge A) \Leftrightarrow A$
Absorption:	$(A \vee (A \wedge B)) \Leftrightarrow A$ $(A \wedge (A \vee B)) \Leftrightarrow A$
Neutralität:	$(A \vee \mathsf{f}) \Leftrightarrow A$ $(A \wedge \mathsf{w}) \Leftrightarrow A$
Distributivität:	$(A \vee (B \wedge C)) \Leftrightarrow ((A \vee B) \wedge (A \vee C))$ $(A \wedge (B \vee C)) \Leftrightarrow ((A \wedge B) \vee (A \wedge C))$
De Morgansche Gesetze: [11]	$\neg(A \vee B) \Leftrightarrow (\neg A \wedge \neg B)$ $\neg(A \wedge B) \Leftrightarrow (\neg A \vee \neg B)$
Umkehr der Implikation:	$(A \Rightarrow B) \Leftrightarrow (\neg B \Rightarrow \neg A)$

3.4. Prädikatenlogik

Mittels Prädikatenlogik (engl. *predicate logic*) können wir Aussagen formulieren, ohne dazu ein bestimmtes Element betrachten zu müssen. Wir können also Eigenschaften formulieren.

Mittels Quantoren und Prädikaten können wir Aussagen über mehrere Elemente machen und Eigenschaften verallgemeinern.

Definition 3.2 (Prädikat)**:** *Ein Prädikat erlaubt das Einsetzen einer festen Anzahl von Variablen und liefert darauf einen Wahrheitswert zurück. Ein Prädikat, welches n Variablen annimmt, nennen wir* n-stellig.

Beispiel 3.2: Das Prädikat "... ist fiktional" liefert auf das Einsetzen von "Moria", "Donald Duck" oder "Elysium" den Wahrheitswert wahr, auf das Einsetzen von "Jackie Kennedy" oder "Ian McKellen" den Wahrheitswert falsch.

Eine Eigenschaft wie "$x < 5$" ist ebenso ein Prädikat, welches z.B. für $x = 3$ den Wahrheitswert wahr und für "$x = 10$" den Wahrheitswert falsch zurückgibt. ◁

Im Folgenden bezeichnen wir Prädikate mit Großbuchstaben und Variablen, die das Prädikat annimmt, mit Kleinbuchstaben. Also ist $P(x)$ ein einstelliges Prädikat.

[11] Augustus De Morgan, englischer Mathematiker, 1806–1871

3.5. Quantoren

Als Erstes definieren wir uns vielfache Junktoren. Ähnlich dem Summenzeichen drücken diese Zeichen ein vielfaches `AND` (“$\wedge$”) bzw. `OR` (“$\vee$”) aus.

Definition 3.3 (Vielfache Junktoren)**:** *Sei M eine Menge und alle $x_i \in M$. Weiterhin sei $P(x)$ ein Prädikat. Wir definieren:*

$$\bigwedge_i P(x_i) \coloneqq \underbrace{P(x_1) \wedge P(x_2) \wedge P(x_3) \wedge \dots}_{alle\ x_i}$$

$$\bigvee_i P(x_i) \coloneqq \underbrace{P(x_1) \vee P(x_2) \vee P(x_3) \vee \dots}_{alle\ x_i}$$

Damit definieren wir jetzt Quantoren.

Definition 3.4 (Allquantor)**:** *Sei $P(x)$ ein einstelliges Prädikat. Um auszusagen, dass das Prädikat $P(x)$ für alle x gilt, schreiben wir $\forall x : P(x)$, gelesen: “für alle x gilt $P(x)$”. “$\forall$” heißt* Allquantor*.*

Formal: Enthalte die Folge der x_i alle x, dann definieren wir:

$$\forall x : P(x) \coloneqq \bigwedge_i P(x_i)$$

Definition 3.5 (Existenzquantor)**:** *Sei $P(x)$ ein einstelliges Prädikat. Um auszusagen, dass $P(x)$ für* mindestens *ein x gilt, schreiben wir $\exists x : P(x)$ und lesen “es existiert ein x für das $P(x)$ gilt”. “$\exists$” nennt sich* Existenzquantor*.*

Formal: Enthalte die Folge der x_i alle x, dann definieren wir:

$$\exists x : P(x) \coloneqq \bigvee_i P(x_i)$$

Bemerken Sie, dass beide Quantoren eine Aussage *über alle x* machen, also alle Elemente aller Mengen [12] (inklusive Zahlen, Studierender und Fahrräder). Meist möchte man spezifischere Aussagen machen und gibt die Grundmenge direkt mit an. Sei M eine Menge, dann ist

$$\forall x \in M : P(x) \coloneqq \forall x : (x \in M \Rightarrow P(x)).$$

Ebenso kann man direkt Bedingungen angeben: $\forall \varepsilon > 0 : \frac{1}{n} < \varepsilon$. Auch das schriebe sich formal exakt: $\forall \varepsilon : (\varepsilon > 0 \Rightarrow \frac{1}{n} < \varepsilon)$. Das Analoge gilt für $\exists$.

Der Allquantor verallgemeinert ein Prädikat auf eine Menge von Elementen. Da diese Aussage für alle Elemente der Menge wahr sein muss, genügt ein einziges Gegenbeispiel, um die Aussage zu widerlegen. Z.B. ist die Aussage

$$\forall x \in \mathbb{N} : (x \text{ ist Primzahl } \Rightarrow x \text{ ist ungerade})$$

falsch, da es ein (einziges) Element gibt, für das das nicht stimmt.

[12] Siehe nächstes Kapitel.

Wenn "$\emptyset$" die leere Menge bezeichnet, dann ist die Aussage $\forall x \in \emptyset : A(x)$ wahr für ein beliebiges Prädikat $A(x)$. Es gibt kein x, für das die Aussage falsch wäre.

Das lässt sich auch einfach beweisen:

$$\begin{aligned}\forall x \in \emptyset : A(x) &\Leftrightarrow \forall x : (x \in \emptyset \Rightarrow A(x)) \\ &\Leftrightarrow \forall x : (\mathsf{f} \Rightarrow A(x))\end{aligned}$$

Aus der Definition der Implikation wissen wir aber, dass eine Folgerung aus etwas Falschem immer wahr ist:

$$\begin{aligned}\forall x \in \emptyset : A(x) &\Leftrightarrow \forall x : \mathsf{w} \\ &\Leftrightarrow \mathsf{w}.\end{aligned}$$

□

Umgekehrt ist $\exists x \in \emptyset : A(x)$ falsch, da kein x existiert, für das die Aussage wahr wäre. Der formale Beweis funktioniert analog.

Die Schreibweise für Quantorenaussagen ist nicht einheitlich. Man liest $\forall x : P(x)$, $\forall x P(x)$ oder $\forall x.P(x)$.

In der Aussage $\forall x : P(x)$ bezeichnen wir x als *gebundene Variable*, da sie an den Quantor gebunden ist. Im Gegensatz dazu ist in der Aussage $\forall x : P(x, y)$ die Variable y eine *freie Variable*.

Quantoren stehen in der Rangfolge der Operatoren (siehe Seite 12) auf gleicher Höhe wie die Negation und über der Konjunktion. D.h., $\forall x : P(x) \wedge Q(x)$ ist das gleiche wie $(\forall x : P(x)) \wedge Q(x)$. Sonst müssen Sie klammern: $\forall x : (P(x) \wedge Q(x))$.

Es gibt auch einen Quantor, der aussagt, dass ein Prädikat *genau einmal* gilt (Einzigkeitsquantor). Er ist so definiert:

$$\exists! x : B(x) \;:=\; \exists x : (B(x) \wedge \forall y : (B(y) \Rightarrow y = x)).$$

Beispiel 3.3: Benutzung von Quantoren:

- $\forall x \in \{2, 4, 6\} : x$ ist gerade
- $\exists x \in \{2, 4, 6\} : x \leq 4$
- $\forall \varepsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : \frac{1}{n} < \varepsilon$

  Man sagt: "Für alle ϵ größer 0 existiert ein n_0 aus $\mathbb{N}$, sodass für alle n größer oder gleich n_0 gilt: $\frac{1}{n}$ ist kleiner als ε."
- $\forall x \in \mathbb{N} : (x > 4 \Rightarrow x^2 < 2^x)$

◁

3.6. Quantorenregeln

Sei A eine Aussage. Dann kann man die Negation einer Quantorenaussage direkt vor die Aussage A ziehen, wenn man den Quantor "umdreht":

$$\begin{aligned}\neg(\forall x : A) &\Leftrightarrow \exists x : \neg A \\ \neg(\exists x : A) &\Leftrightarrow \forall x : \neg A\end{aligned}$$

Wenn man annimmt, dass es nur zwei verschiedene Werte für x gibt, dann ergeben sich aus den Quantorenregeln die de Morganschen Gesetze. [13]

[13] Übersicht über Quantorenregeln im Netz: http://www.reisz.de/qa.htm und http://www.reisz.de/q

`a2.htm` (secco).

4. Mengen

Die ganzen Zahlen hat der liebe Gott gemacht, alles andere ist Menschenwerk.

— Leopold Kronecker [14] (1893)

4.1. Beschreibung

Nach Georg Cantor [15] ist eine *Menge* (engl. *set*) eine Ansammlung von wohlunterscheidbaren Objekten der Anschauung oder des Denkens. Das können Zahlen sein, aber auch jede andere Form von (evtl. abstrakten) Objekten. Auch andere Mengen können in einer Menge enthalten sein. Wir werden uns mit diesem naiven Mengenbegriff begnügen, eine tiefere Betrachtung liefert die axiomatische Mengenlehre in der Mathematik.

Sei M eine Menge und x ein Objekt dieser Menge, so sagen wir, dass x ein *Element* der Menge M ist. Wir schreiben dafür $x \in M$. Es muss entscheidbar sein, ob ein Element x in der Menge M enthalten ist oder nicht, das nennt sich die *Wohldefiniertheit* der Menge. Ist x kein Element der Menge M, so schreiben wir $x \notin M$. Wollen wir eine Aussage über mehrere Elemente machen, so schreiben wir auch $x, y \in M$.

Wir können *endliche Mengen* beschreiben durch Aufzählung ihrer Elemente. Dabei werden die Elemente durch geschweifte Klammern ("{" und "}") eingefasst:

$$\begin{aligned} B &= \{0, 1\}, \\ F &= \{\text{rot}, \text{grün}, \text{blau}\}, \\ A &= \{\alpha, \omega\}, \\ P &= \{1, x, x^2, x^3\}.^{16} \end{aligned}$$

Die leere Menge (engl. *empty set*) (die Menge ohne Elemente) wird mit $\emptyset$ (oder auch mit $\{\}$) bezeichnet.

$$\emptyset := \{\}.$$

Mengen sind *ungeordnet* (engl. *unordered*):

$$\{2, 3, 5\} = \{5, 2, 3\}.$$

Jedes Element kommt nur einmal in der Menge vor, selbst, wenn es mehrfach angeben wird. Daher müssen die Elemente wohlunterscheidbar sein.

$$\{6, 4, 6\} = \{4, 6\}.$$

Wenn die Abfolge klar ist, können wir uns mit "…" Schreibarbeit sparen:

$$D = \{0, 1, 2, \ldots, 9\}.$$

Auf diese Weise kann man auch *unendliche Mengen* beschreiben:

$$G = \{0, 2, 4, \ldots\}.$$

[14] Deutscher Mathematiker, 1823–1891
[15] Deutscher Mathematiker und Begründer der Mengenlehre, 1845–1918

Wir können auch eine Menge definieren, indem wir eine Eigenschaft ihrer Elemente beschreiben. Die folgende Zeile liest sich "die Menge aller x aus $\mathbb{N}$ mit der Eigenschaft: x ist eine Primzahl":

$$P = \{x \in \mathbb{N} \mid x \text{ ist eine Primzahl}\}.$$

Damit können wir die Prädikatenlogik aus Kapitel 3 benutzen, um Mengen zu definieren:

$$G' = \{x \mid \exists y \in \mathbb{Z} : x = 2y\}.$$

Es gibt einige häufig benutzte grundlegende Mengen, die zur besseren Kennzeichnung mit einem doppelten senkrechten Strich geschrieben werden (in der englischsprachigen Literatur schreibt man diese Mengen auch gerne fett, also **N**, **Z** oder **R**). Das sind unter anderem diese:

- Die Menge der *natürlichen Zahlen* (engl. *natural numbers*): $\mathbb{N} \coloneqq \{1, 2, 3, \ldots\}$.
- Die Menge der natürlichen Zahlen mit Null: $\mathbb{N}_0 \coloneqq \{0, 1, 2, 3, \ldots\}$.
- Die Menge der *ganzen Zahlen* (engl. *integers*): $\mathbb{Z} \coloneqq \{0, \pm 1, \pm 2, \pm 3, \ldots\}$.
- Die Menge der *rationalen Zahlen* (engl. *rational numbers*): $\mathbb{Q} \coloneqq \{\frac{a}{b} \mid a \in \mathbb{Z}, b \in \mathbb{N}\}$.
- Die Menge der *reellen Zahlen* (engl. *real numbers*): $\mathbb{R}$.
- Die Menge der *komplexen Zahlen* (engl. *complex numbers*): $\mathbb{C} \coloneqq \{a + ib \mid a, b \in \mathbb{R}\}$, wobei i die *imaginäre Einheit* ist und definiert ist als $i^2 = -1$.

Sei x ein Element einer dieser Mengen, dann heißt x

- *positiv*, falls $x > 0$,
- *negativ*, falls $x < 0$,
- *nicht-negativ* (engl. *non-negative*), falls $x \geq 0$.

Intervalle einer Menge werden durch ihre untere und obere Grenze angegeben. Dabei unterscheidet man *offene* (engl. *open*) und *abgeschlossene* Intervalle (engl. *closed intervals*).

- Das abgeschlossene Intervall $[a, b]$ einer Menge M ist definiert als $[a, b] \coloneqq \{x \in M \mid a \leq x \leq b\}$, das heißt, die Grenzen liegen im Intervall.
- Das offene Intervall (a, b) einer Menge M ist definiert als $(a, b) \coloneqq \{x \in M \mid a < x < b\}$, das heißt, die Grenzen sind nicht im Intervall enthalten.
- Es gibt auch *halboffene* Intervalle, beispielsweise ist das Intervall $[a, b)$ einer Menge M definiert als $[a, b) \coloneqq \{x \in M \mid a \leq x < b\}$.

Die obere Intervallgrenze kann ∞ sein, resp. die untere Grenze $-\infty$. Das ist eine Art zu schreiben, dass auf dieser Seite keine Grenze existiert. [17] Beachten Sie, dass die Seite mit dem "∞"-Zeichen eine offene Grenze beschreibt, also runde Klammern zu benutzen sind.

[16] Kapitel 11 beschäftigt sich mit Mengen von Funktionen.

[17] Das ist ein Beispiel für *abuse of notation*: eine mathematische Schreibweise, die formal inkorrekt, aber intuitiv (hoffentlich) richtig verstanden wird.

Beispiel 4.1:

$$\mathbb{R}_{\geq 0} \coloneqq [0, \infty)$$
$$\mathbb{R}_{-} \coloneqq (-\infty, 0)$$

◁

4.2. Vereinigung und Schnitt

Wir können auf Mengen diverse Operationen anwenden. [18] Die Operationen können wir sehr schön mit den Methoden der Logik aus dem letzten Kapitel beschreiben und beweisen.

Oft benutzt man auch *Venn-Diagramme*, [19] um Beziehungen von Mengen darzustellen. Sie sind sehr intuitiv, ersetzen aber keinen formalen Beweis.

Definition 4.1 (Mengenoperationen)**:** *Seien Mengen A und B gegeben, dann definieren wir*

- *die* Vereinigung *(engl.* union*)* $C = A \cup B$*: C enthält alle Elemente aus A und alle Elemente aus B. In der Sprache der Logik heißt das:*

$$(A \cup B) \coloneqq \{x \mid x \in A \vee x \in B\}.$$

- *Der* Schnitt *(engl.* intersection*)* $C = A \cap B$*: C enthält alle Elemente, die sowohl in A als auch in B sind. Mittels der Logik definieren wir:*

$$(A \cap B) \coloneqq \{x \mid x \in A \wedge x \in B\}.$$

- *Die* Differenz *(engl.* set difference*)* $C = A \setminus B$*: C enthält alle Elemente aus A, die nicht in B sind. Man sagt auch "das* Komplement *von B in Bezug auf A" oder "A ohne B". Die Definition lautet:*

$$(A \setminus B) \coloneqq \{x \mid x \in A \wedge x \notin B\}.$$

  Eine andere Schreibweise statt $A \setminus B$ ist $\overline{B}$, wobei hier zuerst unklar bleibt, welches die Obermenge ist.

4.3. Teilmengenbeziehungen

Definition 4.2 (Teilmengenbeziehungen)**:** *Weiterhin können wir Aussagen über das Verhältnis zweier Mengen zueinander machen:*

- *Wir nennen A und B* gleich*, geschrieben $A = B$, falls gilt:*

$$(A = B) \coloneqq \forall x : (x \in A \Leftrightarrow x \in B).$$

- *Wir nennen A eine* Teilmenge *(engl.* subset*) von B, geschrieben $A \subseteq B$, falls alle Elemente aus A auch in B liegen. Die Definition lautet:*

$$(A \subseteq B) \coloneqq \forall x : (x \in A \Rightarrow x \in B).$$

[18] Video von Christian Spannagel über Mengenlehre.
[19] John Venn, englischer Logiker und Philosoph, 1834–1923

- *Umgekehrt heißt B* Obermenge *(engl.* superset*) von A: $B \supseteq A$.*
- *Eine* echte Teilmenge *(engl.* proper subset*) $A \subset B$ (oder, noch expliziter: $A \subsetneq B$) ist eine Teilmenge A von B mit $A \neq B$. Also:*

  $$(A \subset B) \coloneqq A \subseteq B \wedge A \neq B.$$

  Analog $B \supset A$.
- *Wir nennen A und B* disjunkt *(engl.* disjoint*), falls es kein Element gibt, welches in beiden Mengen enthalten ist, also falls gilt: $A \cap B = \emptyset$.*

Beispiel 4.2: Seien die Mengen $A = \{2, 3, 5, 7\}$, $B = \{1, 2, 4, 8\}$, $C = \{5, 7\}$ gegeben. Dann gilt:

$$\begin{aligned} C &\subseteq A, \\ A &\supset C, \\ A \cup B &= \{1, 2, 3, 4, 5, 7, 8\}, \\ A \cap B &= \{2\}, \\ A \setminus B &= \{3, 5, 7\}, \\ B \cap C &= \emptyset, \\ A \setminus \emptyset &= A. \end{aligned}$$

$\triangleleft$

Ebenso gilt für die uns wohlbekannten Mengen:

$$\emptyset \subset \mathbb{N} \subset \mathbb{N}_0 \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R} \subset \mathbb{C}.$$

Einige Eigenschaften, die aus den Definitionen folgen, sind:

- Die Vereinigung der leeren Menge mit einer beliebigen Menge A ergibt A: $A \cup \emptyset = A$.
- Der Schnitt aus der leeren Menge mit jeder Menge A ergibt die leere Menge: $A \cap \emptyset = \emptyset$.
- Die leere Menge $\emptyset$ ist Teilmenge jeder Menge A: $\emptyset \subseteq A$.
- Jede Menge A ist Teilmenge ihrer selbst: $A \subseteq A$. Diese beiden Teilmengen nennt man auch die *trivialen* Teilmengen.
- Die Gleichheit zweier Mengen A und B gilt genau dann, wenn:

  $$(A = B) \Leftrightarrow (A \subseteq B \wedge B \subseteq A).$$

  Das ist teilweise einfacher zu beweisen als die Gleichheit nach der obigen Definition.
- Der Schnitt einer Menge A mit sich selbst und die Vereinigung mit sich selbst ergeben wieder A:

  $$A \cup A = A \quad \text{und} \quad A \cap A = A.$$

- Vereinigung und Schnitt sind assoziativ. Seien A, B und C Mengen, dann gilt:

  $$\begin{aligned} (A \cup B) \cup C &= A \cup (B \cup C), \\ (A \cap B) \cap C &= A \cap (B \cap C). \end{aligned}$$

- Die de Morganschen Gesetze gelten auch auf Mengen. Seien A, B und C Mengen mit $A \subseteq C$ und $B \subseteq C$. Die De Morganschen Gesetze besagen dann:

$$\begin{aligned} \overline{A \cap B} &= \overline{A} \cup \overline{B} \quad \text{oder anders geschrieben} \\ C \setminus (A \cap B) &= (C \setminus A) \cup (C \setminus B). \end{aligned}$$

  Ebenso

$$\begin{aligned} \overline{A \cup B} &= \overline{A} \cap \overline{B} \quad \text{oder anders geschrieben} \\ C \setminus (A \cup B) &= (C \setminus A) \cap (C \setminus B). \end{aligned}$$

Jetzt können wir die Gesetze der Logik benutzen, um aus den Definitionen der Mengenoperationen einige Eigenschaften zu beweisen:

Seien A, B Mengen. Zu zeigen:

$$A \cap \emptyset = \emptyset.$$

Beweis: Nach Definition von Schnitt ist das:

$$\begin{aligned} \{x \mid x \in A \wedge x \in \emptyset\} &= \emptyset \\ \{x \mid x \in A \wedge \mathsf{falsch}\} &= \\ \{x \mid \mathsf{falsch}\} &= \\ \{\} &= \emptyset. \end{aligned}$$

□

Nun beweisen wir die Assoziativität des Schnitts:

Seien A, B Mengen. Zu zeigen:

$$(A \cap B) \cap C = A \cap (B \cap C)$$

Beweis: Nach Definition von Schnitt ist das:

$$\forall x : \big(x \in (A \cap B) \cap C \Leftrightarrow x \in A \cap (B \cap C)\big)$$

Wähle ein x aus: sei x fest, aber beliebig.

$$\begin{aligned} x \in (A \cap B) \cap C &\Leftrightarrow x \in (A \cap B) \wedge x \in C \\ &\Leftrightarrow (x \in A \wedge x \in B) \wedge x \in C \\ &\Leftrightarrow x \in A \wedge (x \in B \wedge x \in C) \\ &\Leftrightarrow x \in A \wedge x \in (B \cap C) \\ &\Leftrightarrow x \in A \cap (B \cap C) \end{aligned}$$

□

Als Letztes beweisen wir eines der De Morganschen Gesetze:

Seien A, B, C Mengen mit $(A \cup B) \subseteq C$. Zu zeigen:

$$\overline{A \cup B} = \overline{A} \cap \overline{B}$$

Oder, mit C als Obermenge von A und B:

$$C \setminus (A \cup B) = (C \setminus A) \cap (C \setminus B)$$

Beweis: Nach Definition von Komplement:

$$\{x \mid x \in C \wedge \neg(x \in (A \cup B))\} =$$

Nach Definition von Vereinigung:

$$\{x \mid x \in C \wedge \neg(x \in A \vee x \in B)\} =$$

Nach Anwendung von De Morgan:

$$\{x \mid x \in C \wedge (x \notin A \wedge x \notin B)\} =$$
$$\{x \mid x \in C \wedge x \notin A \wedge x \notin B\} =$$

Wir erweitern (nach Idempotenz):

$$\{x \mid x \in C \wedge x \notin A \wedge x \in C \wedge x \notin B\} =$$

Nach Definition von Komplement:

$$\{x \mid x \in (C \setminus A) \wedge x \in (C \setminus B)\} =$$

Nach Definition von Schnitt:

$$(C \setminus A) \cap (C \setminus B) = (C \setminus A) \cap (C \setminus B) \qquad \square$$

Eine interessante Beobachtung: bei Mengen ist (im Gegensatz zu reellen Zahlen) der Fall möglich, dass weder $A \subseteq B$ noch $A \supseteq B$ gilt! Das heißt, die Negation von $A \subseteq B$ ist nicht $A \supset B$, sondern $\neg(A \subseteq B)$ oder $A \not\subseteq B$. [20]

4.4. Weitere Operationen auf Mengen

Definition 4.3 (Kardinalität)**:** *Die Anzahl der Elemente einer endlichen Menge M heißt* Kardinalität *(engl.* cardinality*) oder* Mächtigkeit*. Wir schreiben dafür $|M|$, manchmal findet sich auch $\#M$.*[21]

Hat eine unendliche Menge M die gleiche Mächtigkeit wie $\mathbb{N}$, d.h., existiert eine Bijektion [22] zwischen M und $\mathbb{N}$, so sagt man M sei *abzählbar unendlich* und habe die Mächtigkeit $\aleph_0 := |\mathbb{N}|$, gesprochen "Aleph Null" (der erste Buchstabe des hebräischen Alphabets).

Existiert eine solche Abbildung nicht, so nennt sich M *überabzählbar*. Z.B. ist $\mathbb{R}$ überabzählbar.

Beispiel 4.3: Sei $A = \{1, 2, 3\}$, dann ist $|A| = 3$.

Die leere Menge hat Mächtigkeit 0: $|\emptyset| = 0$. $\triangleleft$

[20] Der Grund dafür ist, dass die Teilmengenbeziehung zwar eine partielle, aber keine totale Ordnung ist, siehe Abschnitt 7.4.

[21] Die Kardinalität einer unendlichen Menge ist nicht so einfach anzugeben, erstaunlicherweise gibt es hier verschiedene Mächtigkeiten.

[22] Zum Begriff der Bijektivität siehe Kapitel 7.2.

Definition 4.4 (Potenzmenge)**:** *Zu einer gegebenen Menge A ist die* Potenzmenge *(engl.* power set*)* $\mathcal{P}(A)$ *die Menge aller Teilmengen von A.*

$$\mathcal{P}(A) = \{B \mid B \subseteq A\}.$$

Die Potenzmenge $\mathcal{P}(M)$ zu einer Menge M hat die Kardinalität $2^{|M|}$. Die trivialen Teilmengen von A ($\emptyset$ und A selbst) sind auch in der Potenzmenge enthalten.

Beispiel 4.4: Sei $A = \{1, 2, 3\}$, dann ist $\mathcal{P}(A) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$. Nachzählen zeigt, dass $|\mathcal{P}(A)| = 2^{|A|} = 8$. ◁

Definition 4.5 (Kartesisches Produkt)**:** *Das* kartesische Produkt *$A \times B$ zweier Mengen A und B ist die Menge aller* geordneten Paare *(a, b) mit $a \in A$ und $b \in B$. "Geordnetes Paar" bedeutet, dass die Reihenfolge von a, b wichtig ist, im allgemeinen also $(a, b) \neq (b, a)$ gilt.* [23] *Formal gilt:*

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

Beispiel 4.5: Sei $R = \{1, 2, 3, \ldots, 8\}$ und $L = \{a, b, c, \ldots, h\}$. Dann ist

$$\begin{aligned} R \times L = \{&(1, a), (1, b), (1, c), \ldots, (1, h), \\ &(2, a), (2, b), (2, c), \ldots, (2, h), \\ &\vdots \\ &(8, a), (8, b), (8, c), \ldots, (8, h)\}. \end{aligned}$$

◁

Sei B das n-fache kartesische Produkt $A_1 \times A_2 \times \ldots \times A_n$ von Mengen A_i, dann nennt man ein Element von B ein *n-Tupel*. Das heißt: $B = \{(a_1, a_2, \ldots, a_n) \mid a_i \in A_i\}$. Ein Paar ist also ein 2-Tupel.

[23] Wir schreiben "*im allgemeinen* $(a, b) \neq (b, a)$", da es spezielle Belegungen von a, b gibt, für die es eben doch gilt, z.B. hier für $a = b$.

5. Datentypen

To program is to understand.

— Kristen Nygaard [24]

In der Informatik beschäftigen wir uns unter anderem damit, mathematische Sachverhalte in Programmen abzubilden und mit deren Hilfe Ergebnisse zu berechnen. Dazu müssen wir Elemente von Mengen in *Variablen* speichern, um mit ihnen rechnen zu können. Diese Variablen haben einen *Datentyp* (engl. *data type*), der bestimmt, wie sie gespeichert werden und welchen Wertebereich sie haben.

Dieses Kapitel schlägt eine Brücke zwischen Theorie und Praxis. Wir werden sehen, wie die theoretischen Konzepte der Mathematik sich in der Praxis des Programmierens wiederfinden lassen.

5.1. Zahlensysteme

Mittels der Summennotation aus Abschnitt 2.4 können wir einen neuen Blick auf Zahlensysteme werfen. Welche Zahl wird beschrieben, wenn wir "138" schreiben? Normale Konvention ist, dass wir in der Mathematik Zahlen im Zehnersystem notieren. Damit hat die letzte Stellen den Wert 1, die zweitletzte den Wert 10, etc. Wir können es allgemeiner schreiben:

Definition 5.1 (Dezimalnotation)**:** *Sei eine Zahl a im Dezimalsystem als eine Abfolge von Ziffern gegeben, also als $a_n a_{n-1} \dots a_0$, wobei $0 \leq a_i \leq 9$. Dann können wir a auch schreiben als* [25]

$$a = \sum_{i=0}^{n} a_i \cdot 10^i.$$

Im obigen Beispiel ist dann $a_0 = 8$, $a_1 = 3$ und $a_2 = 1$.

Die Dezimalschreibweise hat ihren Ursprung natürlich in der Anzahl unserer Finger. Die Simpsons sollten eigentlich im *Oktalsystem* (engl. *octal numeral system*) rechnen, da sie nur acht Finger haben (lediglich Gott hat zehn).

In der Informatik haben wir nur zwei "natürliche" Ziffern: 0 und 1, die ganz simpel die Zustände "an" und "aus" darstellen. Damit bietet es sich für uns an, Zahlen im Dualsystem oder *Binärsystem* (engl. *binary numeral system*) darzustellen. Wir verwenden hier einen Index am Ende der Zahl, um das Zahlensystem kenntlich zu machen. Der Index gibt die *Basis* des Zahlensystems in Dezimalschreibweise an. Die Dezimalzahl "138_{10}" schreibt sich in oktal also "212_8" und in binär "10001010_2".

Damit erweitern wir jetzt unsere Definition von Zahlensystemen:

[24] Norwegischer Informatiker und Pionier der Programmiersprachen, 1926–2002
[25] Sie erinnern sich an das Summenzeichen aus Abschnitt 2.4?

Definition 5.2 (Notation in beliebigen Zahlensystem)**:** *Sei im Zahlensystem mit Basis $b \in \mathbb{N}$, $b > 1$ eine Zahl a als eine Abfolge von Ziffern gegeben, also als $a_n a_{n-1} \dots a_0$, wobei $0 \leq a_i < b$. Dann können wir a schreiben als*

$$a = \sum_{i=0}^{n} a_i \cdot b^i.$$

Häufig werden Zahlen in der Programmierung auch in *hexadezimal* (kurz: "in hex") angegeben, das heißt in Basis 16. Die fehlenden Ziffern über der 9 werden durch die Buchstaben a, b, c, d, e, f dargestellt. 138 wird in hex also als "$8a_{16}$" geschrieben.

Viele Programmiersprachen bieten die Möglichkeit, Zahlen direkt in verschiedenen Zahlensystemen anzugeben. Dafür wird oft ein Präfix verwendet:

	oktal	binär	hexadezimal
C/C++ [26]	`0`	—	`0x`
Python	`0o`	`0b`	`0x`

Unsere Beispielzahl 138 schreibt sich also in C/C++ auch als `0212` oder `0x8a` und in Python auch als `0o212`, `0b10001010` oder `0x8a`.

5.2. Skalare Datentypen

Jede Programmiersprache hat ihre eigenen Datentypen. Wir betrachten hier exemplarisch die Datentypen aus C/C++ und Python. Die hier angegebenen Wertebereiche W gelten für viele, aber nicht notwendigerweise alle Implementationen dieser Sprachen.

- C/C++ `unsigned` speichert nicht-negative ganze Zahlen kleiner als 2^{32}, also: $W = \{x \in \mathbb{N}_0 \mid x < 2^{32}\}$.
- C/C++ `int` speichert ganze Zahlen zwischen $-(2^{31})$ und $2^{31} - 1$ inklusive, also: $W = \{x \in \mathbb{Z} \mid -(2^{31}) \leq x < 2^{31}\}$.
- Python `int` speichert ganze Zahlen so lange der Speicher reicht. Für die meisten praktischen Belange heißt das also: $W = \mathbb{Z}$.
- C/C++ `double` und Python `float` speichern Gleitkommazahlen mit doppelter Genauigkeit (engl. *floating point numbers with double precision*) mit Betrag kleiner als $\approx 1.798 \cdot 10^{308}$ und einer Genauigkeit von ca. 15 Dezimalstellen. Also: $W = \{x \in \mathbb{R} \mid x$ darstellbar als IEEE 754 Gleitkommazahl mit doppelter Genauigkeit $\}$.
  Die Details der Gleitkommadarstellung sind knifflig und geben immer wieder Anlass zum Staunen, siehe unten.
- C/C++ `bool` speichert die Wahrheitswerte `true` und `false`, Python `bool` speichert die Wahrheitswerte `True` und `False`.
  Eine Darstellung ist: $W = \{\mathsf{wahr}, \mathsf{falsch}\}$.

[26] Ja, C/C++ hat keinen Präfix für Binärzahlen und ja, eine 0 leitet eine Oktalzahl ein! Das gibt wunderschöne Bugs, wenn ein Vergleich mit 42 einfach nicht klappen will: `if (a == 042) { ... }`. ☺

Es folgt ein kleiner Exkurs zu den Freuden der floating point Arithmetik. Siehe Floating Point Arithmetic: Issues and Limitations für eine kurze Einführung zu Problemen von `floats` in Python. Mehr unter The Perils of Floating Point für einige erstaunliche Effekte (in FORTRAN, yikes!).

```
Python 3.3.0 (v3.3.0:bd8afb90ebf2, Sep 29 2012, 10:57:17) ...
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 0.1
>>> sum = 0.0
>>> for i in range(10):
...     sum += a
...
>>> sum == a*10
False
>>> a*10
1.0
>>> sum
0.9999999999999999
>>> sum = 0.0
>>> for i in range(10):
>>>     sum += a
>>>     print("{:.30}".format(sum))
>>> 0.100000000000000005551115123126
>>> 0.200000000000000011102230246252
>>> 0.300000000000000044408920985006
>>> 0.400000000000000022204460492503
>>> 0.5
>>> 0.599999999999999977795539507497
>>> 0.699999999999999955591079014994
>>> 0.799999999999999933386618522491
>>> 0.899999999999999911182158029987
>>> 0.999999999999999888977697537484
```

Exkurs Ende. ☺

Man darf also nicht glauben, dass alles, was man sich ausdenkt und als Programm formuliert, auch genau so hinkommt, wie es der Programmcode suggeriert. Das Problem in obigem Beispiel ist die mangelnde Genauigkeit. An anderer Stelle ist es oft der zu kleine Wertebereich einer Variablen (der dann einen Überlauf (engl. *overflow*) erzeugt). Behalten Sie das im Kopf, wenn Sie Probleme aus der Mathematik als Programm formulieren.

Jetzt haben wir Datentypen zur Darstellung von Werten aus $\mathbb{N}$, $\mathbb{N}_0$, $\mathbb{Z}$ und $\mathbb{R}$ kennengelernt. Wie aber stellt man Werte aus $\mathbb{Q}$ dar? Die Antwort: es gibt in C/C++ keinen eingebauten Datentyp, um Werte aus $\mathbb{Q}$ exakt zu speichern, man kann ihn aber nachrüsten, z.B. durch die Boost Rational Number Library. In Python wird eine Klasse mitgeliefert, die durch den Befehl `import fractions` geladen werden kann.

Wir könnten uns den Datentyp auch selber schreiben. Skizzieren wir, was dazu nötig wäre:

- Zwei Zahlen $a, b \in \mathbb{Z}$ mit $b \neq 0$,
- eine Funktion, die den größten gemeinsamen Teiler (ggT) errechnet, damit man kürzen kann. [27] Das wird nötig z.B. für Vergleiche: $\frac{1}{2} = \frac{2}{4}$.

[27] Siehe Abschnitt 9.6

- eine Funktion, die auf das kleinste gemeinsame Vielfache erweitert, damit man zwei Zahlen addieren kann: $\frac{1}{2} + \frac{1}{3} = \frac{5}{6}$.

5.3. Mengen in C/C++ und Python

In C++ kann man Mengen durch das `set` Template deklarieren. Für die entsprechenden Operationen auf Mengen werden Funktionen verwendet, die allerdings aufwändig zu benutzen sind.

Python hat schönen Support für Mengen durch den Datentyp `set`, der einfach zu bedienen ist. Die Operatoren sind stark der mathematischen Notation nachempfunden.

Wir listen hier einige der C++ Funktionen und Python Operatoren auf und ihre vergleichbare Bedeutung in mathematischer Notation. Dabei ist x ein Element und A und B Mengen.

	Math. Notation	C++ Code	Python Code
Ist Element von	$x \in A$	`A.find(x) != A.end()`	`x in A`
Vereinigung	$A \cup B$	`set_union()`	`A | B`
Schnitt	$A \cap B$	`set_intersection()`	`A & B`
Differenz	$A \setminus B$	`set_difference()`	`A - B`
Gleichheit	$A = B$	`A == B`	`A == B`
Teilmenge	$A \subseteq B$	`includes()`	`A <= B`
Echte Teilmenge	$A \subset B$	`includes() && A != B`	`A < B`
Obermenge	$A \supseteq B$	`includes()`	`A >= B`
Echte Obermenge	$A \supset B$	`includes() && A != B`	`A > B`

Es sollte klar sein, dass man aus praktischen Gründen mit den Mengenoperationen einer Programmiersprache nur endliche Mengen darstellen und behandeln kann. Die Menge mit allen Elementen wird dazu im Speicher des Rechners hinterlegt und der ist nun mal (engen) Grenzen unterworfen. Prädikate wie $x \in \mathbb{N}$ kann man nicht über Mengenoperationen in Programmiersprachen lösen.

So sehen die Python Operatoren "in action" aus:

```
Python 3.3.0 (v3.3.0:bd8afb90ebf2, Sep 29 2012, 10:57:17) ...
Type "help", "copyright", "credits" or "license" for more information.
>>> A = {0,2,1}
>>> A
{0, 1, 2}
>>> B = {2,3}
>>> B
{2, 3}
>>> A & B
{2}
>>> A | B
{0, 1, 2, 3}
>>> A - B
{0, 1}
>>> A <= B
False
>>> B <= A
False
```

```
>>> 2 in A
True
```

Und jetzt kommt der entsprechende C++ Code. Man sieht, dass es hier einiges mehr an syntaktischem Overhead gibt. Dafür ist der Code einiges schneller in der Ausführung.

```
#include <iostream>
#include <set>
#include <algorithm>
#include <iterator>
using namespace std;

void set_out(const set<int>& S) {
   for (auto x : S) cout << x << " ";
   cout << endl;
}

void main() {
   set<int> A, B, C, D, E;
   A.insert(0);
   A.insert(2);
   A.insert(1);
   set_out(A);          // output: 0 1 2
   B.insert(2);
   B.insert(3);
   set_out(B);          // output: 2 3
   set_union(A.begin(), A.end(), B.begin(), B.end(), inserter(C, C.begin()));
   set_out(C);          // output: 0 1 2 3
   set_intersection(A.begin(), A.end(), B.begin(), B.end(), inserter(D, D.begin()));
   set_out(D);          // output: 2
   set_difference(A.begin(), A.end(), B.begin(), B.end(), inserter(E, E.begin()));
   set_out(E);          // output: 0 1
   cout << includes(A.begin(), A.end(), B.begin(), B.end()) << endl;   // output: 0
   cout << includes(B.begin(), B.end(), A.begin(), A.end()) << endl;   // output: 0
   cout << (B.find(2) != B.end()) << endl;                             // output: 1
}
```

Was lehrt uns das? Antwort: nicht jede Sprache ist gleichermaßen für jedes Problem geeignet. Daher sollte man eine "passende" Sprache für das jeweilige Problem wählen, so weit das möglich ist.

Ebenso lehrt es uns, dass Diskussionen á la "Sprache X ist besser als Sprache Y" zu wenig führen. [28]

5.4. Boolesche Operationen in Programmiersprachen

Natürlich können die Programmiersprachen C/C++ und Python die Operationen `AND`, `OR`, `XOR` und `NOT` berechnen. Allerdings muss man unterscheiden zwischen *bitweisen* und *logischen* Operationen.

Was wir in Kapitel 3 besprochen haben, nennt sich in den Programmiersprachen *logische Operationen*. Diese schreiben sich folgendermaßen:

[28] OK, alles ist besser als Intercal oder Whitespace! ☺

	math. Notation	C/C++	Python
Konjunktion	$A \wedge B$	`A && B`	`A and B`
Disjunktion	$A \vee B$	`A || B`	`A or B`
Negation	$\neg A$	`!A`	`not A`

Sie liefern die Ergebnisse, die man von ihnen (mathematisch gesehen) erwartet und der Datentyp ist `bool` in C/C++ oder Python.

Weiterhin gibt es aber auch *bitweise Operationen*. Das Vorhandensein dieser Operationen ist eine besondere Eigenschaft von Programmiersprachen auf Binärrechnern, in der Mathematik sind sie wenig gebräuchlich.

Bitweise Operationen arbeiten auf jedem Bit einer Ganzzahl-Variablen.[29] Z.B. wird die Zahl 42 binär dargestellt als $42 = 2 + 8 + 32 = 101010_2$, ebenso $15 = 1 + 2 + 4 + 8 = 1111_2$. Das *binäre* `AND` ist die Anwendung der Konjunktion auf jeder Binärstelle beider Werte: also 101010_2 `AND` $1111_2 = 1010_2$. Der Datentyp des Ergebnisses ergibt sich aus den Datentypen der Operanden.

In unseren Programmiersprachen schreiben sich die bitweisen Operationen wie folgt:

	math. Notation	C/C++	Python
`AND`	$A \wedge B$	`A & B`	`A & B`
`OR`	$A \vee B$	`A | B`	`A | B`
`XOR`	$A \oplus B$	`A ^ B`	`A ^ B`
`NOT`	$\neg A$	`~A`	`~A`

Warum gibt es zwei Ausführungen dieser Operationen? Der hauptsächliche Unterschied ist, dass die logischen Operationen *short-circuit evaluation* unterstützen, d.h., die Auswertung eines logischen Ausdrucks wird so früh wie möglich beendet.

Beispielsweise wird in der Zeile "`if (1 || b) ...`" der Wert von `b` nicht ausgewertet, weil klar ist, dass der ganze Ausdruck `true` ist, da das erste Argument schon `true` ist. Analoges gilt für "`if (0 && b) ...`": dort wird `b` nicht ausgewertet, da klar ist, dass der ganze Ausdruck `false` sein wird. In den meisten Programmiersprachen ist klar geregelt, dass in solchen Fällen der überflüssige Teil der Aussage *garantiert* nicht ausgewertet wird.

Daher kann man in C/C++ folgende Zeile ohne Gefahr einer Division durch Null schreiben: "`if (a != 0 && 1/a > b) ...`". Der zweite Ausdruck ("`1/a > b`") wird nur dann ausgeführt, wenn der erste wahr war, das ist im Standard der Programmiersprache so festgelegt. Sollte der erste Ausdruck falsch sein, wird der zweite nicht mehr ausgewertet, da die Konjunktion nicht mehr wahr werden kann: wir wissen, dass das `if()` nicht ausgeführt werden kann. Damit spart man nicht nur Rechenzeit, sondern kann Programmcode auch kompakter schreiben.

Auch hier wird wieder augenfällig, dass teilweise eine direkte Übertragung mathematischer Sachverhalte in Programmiersprachen zu erstaunlichen Ergebnissen resp. Problemen führen kann.

[29] Siehe Zahlensysteme, Abschnitt 5.1.

6. Beweistechniken

Math answers aren't determined by votes.

— Marilyn vos Savant[30]

Im folgenden Abschnitt wollen wir uns damit befassen, was es heißt, eine Aussage zu beweisen und wie wir dabei vorgehen. Dazu gibt es verschiedene *Beweistechniken*.

Ein mathematischer Satz besteht immer aus zwei Teilen: Einer Behauptung (engl. *statement*) und einem Beweis (engl. *proof*), der die Gültigkeit der Behauptung zeigt. Die Behauptung besteht meistens aus einigen Voraussetzungen und der tatsächlichen Aussage.

Beispiel: Seien $\underbrace{a, b \in \mathbb{R}}_{\text{Voraussetzungen}}$, dann gilt: $\underbrace{(a+b)^2 = a^2 + 2ab + b^2}_{\text{Aussage}}$.

Der Beweis dazu folgt im nächsten Abschnitt.

Es gibt verschiedene Arten von Beweisen. [31] Die wichtigsten gehen wir hier einmal durch.

6.1. Umformen von Gleichungen

Um Gleichungen der Form $T_1 = T_2$ zu lösen, wobei T_1 und T_2 gültige Terme sind, benutzen wir sogenannte *Äquivalenzumformungen*. Dabei handelt es sich um Umformungen, die den Wahrheitsgehalt der gesamten Gleichung erhalten. In $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ oder $\mathbb{C}$ sind dies beispielsweise Addition, Subtraktion, sowie Multiplikation und Division mit beliebigen Konstanten $\neq 0$. Beachten Sie, dass Quadrieren in $\mathbb{Z}$, $\mathbb{Q}$ oder $\mathbb{R}$ keine Äquivalenzumformung ist! Eine hinreichendes Kriterium für Äquivalenzumformungen ist die *Injektivität* der Operation, siehe Abschnitt 7.2.

Beispiel 6.1:

$$7x + 12 = 5x + 16.$$

Diese Gleichung lässt sich durch die folgenden Umformungen sehr leicht lösen.

$$\begin{aligned}
& 7x + 12 = 5x + 16 && \mid -5x \\
\Leftrightarrow\ & 7x + 12 - 5x = 5x + 16 - 5x \\
\Leftrightarrow\ & 2x + 12 = 16 && \mid -12 \\
\Leftrightarrow\ & 2x + 12 - 12 = 16 - 12 \\
\Leftrightarrow\ & 2x = 4 && \mid \div 2 \\
\Leftrightarrow\ & 2x/2 = 4/2 \\
\Leftrightarrow\ & x = 2.
\end{aligned}$$

◁

In der Linearen Algebra lernen Sie das *Gaußsche Eliminationsverfahren* (engl. *Gaussian elimination*) kennen. Damit können lineare Gleichungssysteme in beliebig vielen Variablen gelöst werden.

[30] Amerikanische Kolumnistin und Schriftstellerin, geb. 1946. Bekannt durch ihre Kolumne, speziell durch ihren Artikel zum *Ziegenproblem*.

[31] Es gibt noch viele andere Arten von Beweisen, die wir hier aber nicht behandeln. ☺

6.2. Direkte Beweise

Bei einem direktem Beweis wird die Aussage aus bereits zuvor bewiesenen Aussagen oder aus (per Definition) als wahr vorausgesetzten Aussagen gefolgert. Nicht weiter beweisbare Aussagen nennen wir *Axiome*. [32]

Ein direkter (engl. *direct proof*) Beweis beginnt häufig mit dem zu Zeigenden und endet mit einer wahren Aussage. Wie wir in Abschnitt 3.2 gesehen haben, kann man aber aus jeder Aussage eine wahre Aussage machen. Im Falle einer Gleichung kann man z.B. einfach beide Seiten mit 0 multiplizieren.

Um einen korrekten Beweis zu führen, muss man die ganze Zeit Äquivalenzumformungen benutzen, denn dann kann man die Kette von Aussagen von unten lesen und von einer als wahr bekannten Aussage zu der zu beweisenden Aussage kommen. In der Sprache der Aussagenlogik heißt das: wir wollen Aussage A beweisen und zeigen $A \Leftrightarrow B \Leftrightarrow \ldots \Leftrightarrow \mathsf{w}$. Prima, denn das gilt insbesondere auch, $\mathsf{w} \Rightarrow \ldots \Rightarrow B \Rightarrow A$ und das wollten wir zeigen.

In Lehrbüchern findet sich auch oft die schönere Darstellung, in der aus einer bekannten wahren Aussage durch Umformungen das zu Zeigende hergeleitet wird. Dieser Weg ist natürlich auch richtig und gibt zusätzlich Punkte in der B-Note. Das entspricht dann ebenso zu zeigen, dass $\mathsf{w} \Rightarrow \ldots \Rightarrow B \Rightarrow A$.

Als Beispiel nennen wir die wohlbekannten binomischen Formeln:

Seien a, $b \in \mathbb{R}$. Dann gilt:

$$\begin{aligned}(a+b)^2 &= a^2 + 2ab + b^2,\\ (a-b)^2 &= a^2 - 2ab + b^2,\\ (a+b)(a-b) &= a^2 - b^2.\end{aligned}$$

Beispiel 6.2: Wir beweisen jetzt die erste binomische Formel mittels Termumformung. D.h., eine der beiden Seiten der Gleichung verändert sich nicht, während auf der anderen Seite Äquivalenzumformungen gemacht werden.

$$\begin{aligned}(a+b)^2 &= a^2 + 2ab + b^2\\ (a+b)(a+b) &= \\ a(a+b) + b(a+b) &= \\ aa + ab + ba + bb &= \\ a^2 + ab + ab + b^2 &= \\ a^2 + 2ab + b^2 &= a^2 + 2ab + b^2\end{aligned}$$ □

[33]

Alle Beweise, die wir bisher gesehen haben, sind direkte Beweise.

[32] Die Axiome der wohlbekannten reellen Zahlen $\mathbb{R}$ werden wir in Abschnitt 9.5 genauer kennenlernen.

[33] Das Zeichen "□" oder auch "■" findet man oft, um das Ende eines Beweises zu kennzeichnen. "Q.E.D." (*quod erat demonstrandum*) findet sich heute selten.

6.3. Fallunterscheidung

Eine Technik, die oft zur Anwendung kommt, ist die (vollständige) Fallunterscheidung (engl. *proof by exhaustion*):

Beispiel 6.3:

Behauptung: Das Produkt zweier natürlicher Zahlen a, b ist genau dann ungerade, wenn sowohl a als auch b ungerade sind.

Beweis: Was wir zeigen müssen, ist, dass das Produkt zweier ungeraden Zahlen ungerade ist *und* dies in keinem anderen Fall so ist.

Erster Fall: a, $b \in \mathbb{N}$ sind beide gerade. Dann gibt es k, $l \in \mathbb{N}$ mit den Eigenschaften

$$\begin{aligned} a &= 2k, \\ b &= 2l. \end{aligned}$$

Das Produkt $a \cdot b$ ist dann

$$\begin{aligned} a \cdot b &= 2k \cdot 2l \\ &= 2(2kl). \end{aligned}$$

Somit ist das Produkt gerade.

Zweiter Fall: Seien a, $b \in \mathbb{N}$ beide ungerade. Dann existieren k, $l \in \mathbb{N}$ und es gilt

$$\begin{aligned} a &= 2k - 1, \\ b &= 2l - 1. \end{aligned}$$

Das Produkt $a \cdot b$ ist dann

$$\begin{aligned} a \cdot b &= (2k - 1) \cdot (2l - 1) \\ &= 4kl - 2k - 2l + 1 \\ &= 2(2kl - k - l) + 1. \end{aligned}$$

Da $2kl - k - l \in \mathbb{N}$, ist $2(2kl - k - l)$ gerade. Damit ist das Produkt ungerade.

Dritter Fall: Genau eine der beiden Zahlen a, b ist gerade, die andere ist ungerade. Ohne Beschränkung der Allgemeinheit können wir annehmen, dass $a \in \mathbb{N}$ ungerade ist und $b \in \mathbb{N}$ gerade. Sonst könnten wir a und b austauschen, da die Multiplikation in $\mathbb{R}$ kommutativ ist.

Dann gibt es k, $l \in \mathbb{N}$ und wir können a und b schreiben als

$$\begin{aligned} a &= 2k - 1, \\ b &= 2l. \end{aligned}$$

Das Produkt $a \cdot b$ ist dann

$$\begin{aligned} a \cdot b &= (2k - 1) \cdot (2l) \\ &= 4kl - 2l \\ &= 2(2kl - l). \end{aligned}$$

Das Produkt ist also gerade.

Weitere Fälle existieren nicht. Damit ist die Aussage bewiesen. □

Wir haben bei der Behandlung des dritten Falles geschrieben "ohne Beschränkung der Allgemeinheit", kurz "o.B.d.A." (engl. *without loss of generality*). Diese Formulierung benutzt man, wenn es ausreichend ist, nur einen von mehreren Fällen zu betrachten. Allerdings sollte es offensichtlich sein (im Wortsinne: jedermann klar), dass das zu Zeigende o.B.d.A. gilt oder man muss eine kurze Erklärung angeben, warum.

6.4. Indirekte Beweise

Um einen indirekten Beweis (engl. *proof by contradiction or reductio ad absurdum*) zu führen, behaupten wir das Gegenteil der zu beweisende Aussage und leiten daraus einen Widerspruch her. Darum nennt man einen solchen Beweis auch Widerspruchsbeweis.

Beachten Sie, dass es bei einem Widerspruchsbeweis ausreicht, Implikationen zu benutzen. Aussagenlogisch wollen zeigen, dass A gilt. Dafür nehmen wir an, dass $\neg A$ gilt und führen das zu einem Widerspruch: $\neg A \Rightarrow \mathsf{f}$. Nach Abschnitt 3.3 können wir den Implikationspfeil umkehren, wenn wir beide Seiten negieren. Damit erhalten wir $\neg\mathsf{f} \Rightarrow \neg\neg A$, also $\mathsf{w} \Rightarrow A$. Voilà!

Beispiel 6.4: Wir werden die Irrationalität von $\sqrt{2}$ zeigen, d.h., $\sqrt{2} \notin \mathbb{Q}$. Dafür müssen wir zeigen, dass es keine Zahlen a, $b \in \mathbb{Z}$, $b \neq 0$ gibt, für die $a/b = \sqrt{2}$ gilt. Wir verwenden das Prinzip des Widerspruchsbeweises: wir nehmen also an, dass solche Zahlen existieren und zeigen dann, dass dies zu einem Widerspruch führt.

O.B.d.A. können wir annehmen, dass a und b teilerfremd sind, d.h., dass es kein $c \in \mathbb{Z}$ gibt, welches sowohl a als auch b teilt. Falls ein solches c existierte, könnten wir a und b damit kürzen und diesen Prozess fortsetzen, bis sie teilerfremd sind, ohne dass sich der Wert des Bruches ändert. Außerdem können wir aufgrund der Positivität von $\sqrt{2}$ annehmen, dass sowohl a als auch b nicht-negativ sind, sonst könnten wir durch -1 kürzen. Aus dem gleichen Grunde ist auch klar, dass $a \neq 0$.

Damit können wir eine verfeinerte Behauptung treffen: es existieren keine a, $b \in \mathbb{N}$, die teilerfremd sind und für die $a/b = \sqrt{2}$ gilt.

Für den Widerspruchsbeweis nehmen wir zuerst an, dass ein solches a/b existiert und zeigen, dass dies zu einem Widerspruch führt:

$$\frac{a}{b} = \sqrt{2}$$

Hier dürfen wir quadrieren, da a, $b \in \mathbb{N}$ sind und somit Quadrieren eine Äquivalenzumformung ist.

$$\begin{aligned} \frac{a^2}{b^2} &= 2 \\ a^2 &= 2b^2. \end{aligned}$$

Somit ist a^2 eine gerade Zahl.

In Abschnitt 6.3 haben wir gesehen, dass das Produkt zweier natürlicher Zahlen genau dann ungerade ist, wenn beide Zahlen ungerade sind. Die Umkehrung der Aussage gilt ebenso (da äquivalent): das Produkt zweier natürlicher Zahlen ist genau dann gerade, wenn *mindestens eine der beiden* Zahlen gerade ist. Hier heißen beide Zahlen a und da ihr Produkt $a \cdot a = a^2$ gerade ist, muss auch a gerade sein.

D.h., es gibt ein $k \in \mathbb{N}$ mit $a = 2k$ und wir können schreiben

$$\begin{aligned} a^2 &= 2b^2 \\ (2k)^2 &= 2b^2 \\ 4k^2 &= 2b^2 \\ 2k^2 &= b^2. \end{aligned}$$

Also muss auch b eine gerade Zahl sein und damit wäre 2 ein gemeinsamer Teiler von a und b. Dies steht aber im Widerspruch zu unserer Annahme. $\lightning$ [34]

Damit haben wir gezeigt, dass die Annahme $\sqrt{2} \in \mathbb{Q}$ zu einem Widerspruch führt. Somit muss die gegenteilige Aussage wahr sein und es folgt, dass $\sqrt{2} \notin \mathbb{Q}$. □

6.5. Zyklisches Beweisverfahren

Angenommen, man hat mehrere Aussagen A_1, A_2, …, A_n gegeben und will zeigen, dass alle diese Aussagen äquivalent sind, so reicht es zu zeigen, dass:

$$\begin{aligned} A_1 &\Rightarrow A_2 \\ A_2 &\Rightarrow A_3 \\ A_{n-1} &\Rightarrow A_n \\ &\vdots \\ A_n &\Rightarrow A_1. \end{aligned}$$

Dies kann sehr sinnvoll sein, da wir allein für die Äquivalenz von drei Aussagen A, B, C sonst sechs einzelne Beweisrichtungen zeigen müssten:

$$\begin{array}{lll} A \Rightarrow B & A \Rightarrow C & B \Rightarrow C \\ B \Rightarrow A & C \Rightarrow A & C \Rightarrow B. \end{array}$$

Mit Hilfe des zyklischen Beweisverfahrens (oder Ringschlusses) müssen wir nur die minimale Anzahl an Implikationen zeigen, nämlich

$$A \Rightarrow B \qquad B \Rightarrow C \qquad C \Rightarrow A.$$

Jeder andere Ringschluss, der alle Aussagen enthält, wäre natürlich auch legitim. Nach Abschnitt 3.3 können die fehlenden Implikationen daraus konstruiert werden, beispielsweise entsteht $B \Rightarrow A$ aus $B \Rightarrow C \Rightarrow A$.

Das gilt natürlich auch für den Fall von nur zwei Aussagen A_1 und A_2. Möchte man zeigen, dass $A_1 \Leftrightarrow A_2$ gilt, dann ist es oft einfacher, einzeln die Implikation in jede Richtung zu zeigen. Und wie in Abschnitt 3.2 gezeigt, gilt

$$(A_1 \Leftrightarrow A_2) = (A_1 \Rightarrow A_2 \wedge A_2 \Rightarrow A_1).$$

Beispiel 6.5: Seien A, B Mengen. Dann sind folgende Aussagen äquivalent:

- $A \subseteq B$

[34] Einen Widerspruch in der Beweisführung zeigen wir gerne durch ein "$\lightning$" an.

- $A \cap B = A$
- $A \cup B = B$

Beweis: Wir zeigen, dass folgende Implikationen gelten:

- $A \subseteq B \Rightarrow A \cap B = A$:

  Da $A \subseteq B$ gilt, ist jedes Element aus A auch in B enthalten. Der Schnitt von A und B enthält aber alle Elemente, die in beiden Mengen liegen. Das ist offensichtlich die ganze Menge A, also gilt $A \cap B = A$.

- $A \cap B = A \Rightarrow A \cup B = B$:

  Wenn der Schnitt der Mengen A und B gleich der Menge A ist, so heißt das, dass A keine Elemente beinhaltet, die nicht schon in B enthalten sind. Daher fügt eine Vereinigung von A mit B in B keine weiteren Elemente hinzu. Also gilt $A \cup B = B$.

- $A \cup B = B \Rightarrow A \subseteq B$:

  Da die Vereinigung von A und B gleich B ist, heißt das, dass keine Elemente in A existieren, die nicht bereits in B enthalten sind. Da also jedes Element aus A ebenfalls Element aus B ist, gilt $A \subseteq B$.

Damit haben wir den Ringschluss vollendet und die Äquivalenz aller drei Aussagen gezeigt. □

6.6. Vollständige Induktion

Vollständigen Induktion (engl. *mathematical induction*) ist ein sehr mächtiges Beweisverfahren, welches nicht so offensichtlich ist wie die bisherigen. Die Idee dahinter ist die folgende.

Sei eine Aussage A zu zeigen für alle möglichen Werte $n \in \mathbb{Z}$ mit $n \geq n_0$. Wir beweisen A quasi einzeln für jeden Wert von n. Das klingt nach sehr viel — unendlich viel! — Arbeit. Aber wir ordnen unsere Beweise so geschickt an, dass der Beweis für den Wert $n + 1$ ausnutzt, dass die Aussage für den Wert n schon gezeigt worden ist. Dann müssen wir nur noch den Beweis für den Anfangswert n_0 zeigen und erzeugen damit eine Beweiskette für alle $n \geq n_0$.

Sie können sich das vorstellen wie den Dominoeffekt: [35] wenn der erste Dominostein fällt, fallen alle dahinter auch um.

Wir beschreiben das Verfahren jetzt formal. Wir wollen zeigen, dass eine Aussage $A(n)$ gilt für jeden Wert $n \in \mathbb{Z}$ mit $n \geq n_0$.

Es genügt, folgende Eigenschaften zu zeigen:

1. $A(n_0)$ ist wahr.
2. $\forall n \geq n_0 : A(n) \Rightarrow A(n + 1)$.

Daraus folgt, dass $A(n)$ wahr ist für alle $n \geq n_0$.

Entsprechend unterteilt sich der Beweisvorgang bei der vollständigen Induktion in drei Schritte:

[35] Immer wieder schön: eins, zwei. ☺

1. Zuerst zeigen wir die zu beweisende Aussage bezüglich des Startwerts n_0. Dies bezeichnen wir als *Induktionsanfang* (oder kurz IA) (engl. *base case or basis step*).
2. Wir formulieren die *Induktionsvoraussetzung* (kurz IV) (engl. *induction hypothesis*). Das ist die Aussage $A(n)$, deren Richtigkeit wir im Folgenden annehmen und die wir schon für $n = n_0$ gezeigt haben.
3. Dann folgt der *Induktionsschritt* (engl. *inductive step*): aus der Induktionsvoraussetzung beweisen wir die Induktionsbehauptung: da die zu beweisende Aussage für n bereits bewiesen ist, beweisen wir nun die Gültigkeit der Aussage für $n + 1$.

Beachten Sie:

- Es ist entscheidend, dass $n \in \mathbb{Z}$ ist und nicht evtl. $\in \mathbb{R}$. Sonst würden Sie die Aussage $A(n)$ eben nur für einige ganzzahlige Werte von n beweisen, aber nicht für alle $n \geq n_0$.
- Ebenso wichtig ist, dass Sie im Induktionsschritt nur auf Werte von n zurückgreifen, für die die Aussagen bereits bewiesen ist. Es gibt Beweise, die benutzen die *zwei* vorherigen Werte von n. In solch einem Fall müssten Sie auch für zwei Startwerte die Aussage beweisen (d.h., $A(n) \wedge A(n+1) \Rightarrow A(n+2)$).

Beispiel 6.6: Wir beweisen die Gaußsche Summenformel: [36]

Behauptung: Sei $n \in \mathbb{N}$. Dann gilt

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

Beweis: Der Induktionsanfang ist leicht nachzurechnen. Für $n = 1$ gilt

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

In der Induktionsvoraussetzung gehen wir davon aus, dass die Aussage für n bereits bewiesen ist. Wir zeigen im Induktionsschritt, dass die Aussage auch für $n + 1$ wahr ist.

$$\begin{aligned}
\sum_{i=1}^{n+1} i &= \sum_{i=1}^{n} i + n + 1 \\
&= \frac{n(n+1)}{2} + n + 1 && \text{(nach IV)} \\
&= \frac{n^2 + n}{2} + \frac{2n + 2}{2} \\
&= \frac{n^2 + 3n + 2}{2} \\
&= \frac{(n+1)(n+2)}{2}.
\end{aligned}$$

Somit haben wir gezeigt, dass $A(n) \Rightarrow A(n+1)$. Da wir $A(1)$ gezeigt haben, gilt die Aussage somit für alle $n \in \mathbb{N}$. □

[36] Carl Friedrich Gauß, deutscher Mathematiker, Astronom, Geodät und Physiker, 1777–1855 und einer der bedeutendsten Mathematiker aller Zeiten. Er ist auf dem alten 10 Mark-Schein zu sehen, zusammen mit Graph und Formel der Gaußschen Normalverteilung.

Beispiel 6.7: Ein weiteres klassisches Beispiel, welches man mit vollständiger Induktion beweisen kann, ist die Aussage, dass die Summe der ersten n ungeraden natürlichen Zahlen gleich n^2 ist.

Behauptung: Sei $n \in \mathbb{N}$. Dann gilt

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

Beweis: Der Induktionsanfang ist wiederum leicht überprüft: Für $n = 1$ ist

$$\sum_{i=1}^{1}(2i-1) = (2 \cdot 1 - 1) = 1 = 1^2. \qquad \checkmark$$

Wir beweisen im Induktionsschritt die Aussage für $n+1$ unter der Annahme, dass sie für n bereits bewiesen ist:

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

Hier haben wir wieder den letzten Summanden aus der Summe gezogen und einzeln ans Ende geschrieben. Jetzt verwenden wir die Induktionsvoraussetzung:

$$\begin{aligned} &= n^2 + 2(n+1) - 1. \\ &= n^2 + 2n + 2 - 1 \\ &= n^2 + 2n + 1. \end{aligned}$$

Nun wenden wir die erste binomische Formel an und erhalten

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

Damit ist die Aussage bewiesen. □

Es existiert eine gewisse Ähnlichkeit zwischen dem Prinzip der Induktion und dem der Rekursion. Auch bei der Rekursion wird üblicherweise ein Problem auf eine kleinere Version seiner selbst zurückgeführt und im Endeffekt auf einen (einfachen) Basisfall.

7. Relationen und Funktionen

Der Begriff der *Funktion* ist uns geläufiger und erscheint uns einfacher. Jedoch ist die *Relation* eine Verallgemeinerung der Funktion und daher werden wir sie zuerst einführen.

7.1. Relationen

Anschaulich versteht man unter einer (binären) *Relation* (engl. *relation*) R auf gegebenen Mengen A und B eine Beziehung, die zwischen zwei Elementen $a \in A$, $b \in B$ entweder besteht oder nicht. Wir betrachten hier nur binäre Relationen, aber der Begriff lässt sich leicht auf n-stellige Relationen erweitern.

Definition 7.1 (Binäre Relation)**:** *Formal ist eine (binäre) Relation R eine Menge $R \subseteq A \times B$. Ist $a \in A$, $b \in B$ und $(a, b) \in R$, so sagen wir, dass die Relation R zwischen den Elementen a und b besteht. Statt $(a, b) \in R$ schreiben wir auch aRb.*

Beispiel 7.1: Sei $M \coloneqq \{0, 1, 2\}$ und $M \times M \supseteq R \coloneqq \{(0, 1), (0, 2), (1, 2)\}$. Diese Relation kennen wir als "kleiner als". ◁

Definition 7.2 (Definitions- und Wertebereich)**:** *Seien A und B Mengen und sei $R \subseteq A \times B$, so heißt A* Definitions-*, Vor- oder* Quellbereich *(engl.* domain*) und B* Werte-*, Nach- oder* Zielbereich *(engl.* codomain*).*

Der Urbildbereich *(engl.* preimage*) ist definiert als*

$$\mathrm{dom}(R) \coloneqq \{x \mid \exists y : (x, y) \in R\}.$$

Entsprechend ist der Bildbereich *(engl.* range*) definiert als*

$$\mathrm{ran}(R) \coloneqq \{y \mid \exists x : (x, y) \in R\}.$$

Seien zwei Mengen A, B gegeben mit einer Relation $R \subseteq A \times B$. Dann bezeichnet der Definitionsbereich von R die ganze Menge A, während das Urbild nur die Menge der $a \in A$ bezeichnet, zu denen tatsächlich ein $b \in B$ existiert mit $(a, b) \in R$. Analoges gilt für Wertebereich und Bildbereich. [37]

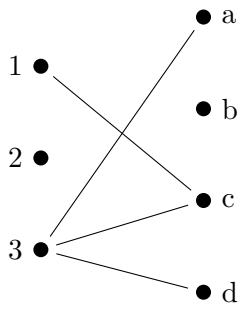


Abbildung 1: Eine binäre Relation

[37] Diese Unterscheidung wird nicht von allen Autoren gemacht.

Beispiel 7.2: Abbildung 1 zeigt eine Relation R zwischen den Mengen $A = \{1, 2, 3\}$ und $B = \{a, b, c, d\}$. Man sieht hier, dass sowohl $\text{dom}(R) = \{1, 3\} \subsetneq A$ als auch $\text{ran}(R) = \{a, c, d\} \subsetneq B$ gilt.

Ebenso kann eine Relation zwischen einem $a \in A$ und mehreren verschiedenen $b_i \in B$ existieren (hier gilt sowohl $4Rc$ wie auch $4Rd$), sowie zwischen mehreren verschiedenen $a_i \in A$ und einem $b \in B$ (wie $2Rc$ und $4Rc$). ◁

Relationen können eine Vielzahl von Eigenschaften besitzen:

Definition 7.3 (Eigenschaften von Relationen)**:** *Seien A und B Mengen und sei $R \subseteq A \times B$ eine Relation, so kann R verschiedene Eigenschaften haben:*

- *R heißt* homogen *genau dann, wenn $A = B$.*
- *R heißt* linkstotal *(engl.* left-total*) genau dann, wenn für jedes $a \in A$ mindestens ein $b \in B$ mit $(a, b) \in R$ existiert. In Prädikatenlogik heißt das:*

$$\forall a \in A : \exists b \in B : (a, b) \in R.$$

- *R heißt* rechtstotal *oder* surjektiv *(engl.* right-total, surjective or onto*) genau dann, wenn für jedes $b \in B$ mindestens ein $a \in A$ mit $(a, b) \in R$ existiert. In Prädikatenlogik heißt das:*

$$\forall b \in B : \exists a \in A : (a, b) \in R.$$

- *R heißt* linkseindeutig *oder* injektiv *(engl.* left-unique or injective*) genau dann, wenn zu jedem $b \in B$ höchstens ein $a \in A$ mit $(a, b) \in R$ existiert. In Prädikatenlogik heißt das:*

$$\forall b \in B : \forall a, c \in A : (a, b) \in R \wedge (c, b) \in R \Rightarrow a = c.$$

- *R heißt* rechtseindeutig *oder* funktional *(engl.* right-unique or functional*) genau dann, wenn zu jedem $a \in A$ höchstens ein $b \in B$ mit $(a, b) \in R$ existiert. In Prädikatenlogik heißt das:*

$$\forall a \in A : \forall b, d \in B : (a, b) \in R \wedge (a, d) \in R \Rightarrow b = d.$$

- *R heißt* bijektiv *(engl.* bijective*) genau dann, wenn zu jedem $b \in B$ genau ein $a \in A$ mit $(a, b) \in R$ existiert. D.h., R ist bijektiv genau dann, wenn R injektiv und surjektiv ist.* [38] *In Prädikatenlogik heißt das:*

$$\forall b \in B : \exists! a \in A : (a, b) \in R.$$

Beispiel 7.3: Hier einige Beispiele:

Die Relationen aus den Abbildung 2–5 haben jeweils nur die genannte Eigenschaft. Abbildung 6 hat dagegen alle vorher genannten Eigenschaften. ◁

[38] Eine Relation ist bijektiv, selbst wenn sie nicht linkstotal ist. Das heißt, dass eine bijektive Relation nicht unbedingt eine (bijektive) Funktion sein muss. Siehe auch Abschnitt 7.2.

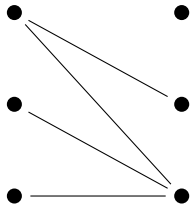


Abbildung 2: linkstotal

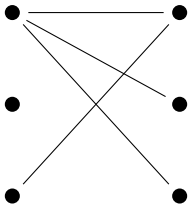


Abbildung 3: surjektiv

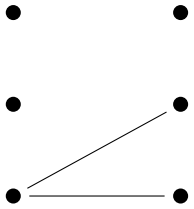


Abbildung 4: injektiv

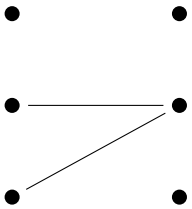


Abbildung 5: rechtseindeutig

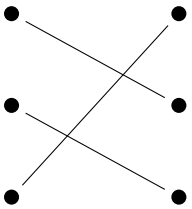


Abbildung 6: bijektiv

Machen Sie sich klar, dass die Relation aus Abbildung 1 keine der in Definition 7.3 genannten Eigenschaften hat.

Die Abbildungen 7–9 im nächsten Abschnitt beschreiben alle linkstotale und funktionale Relationen. Die Relation in Abbildung 7 ist zudem surjektiv, injektiv und damit auch bijektiv. Die Relation in Abbildung 8 ist zudem surjektiv, aber nicht injektiv. Die Relation in Abbildung 9 ist zudem injektiv, aber nicht surjektiv.

Definition 7.4 (Eigenschaften homogener Relationen)**:** *Sei eine Menge A und die Relation $R \subseteq A \times A$ gegeben. Dann heißt R*

- reflexiv *(engl.* reflexive*) genau dann, wenn für alle $a \in A$ gilt: $(a, a) \in R$.*
- total *(engl.* total*) genau dann, wenn für alle $a, b \in A$ gilt: $(a, b) \in R \ \vee \ (b, a) \in R$. Ist eine Relation total, so ist sie auch reflexiv.*
- symmetrisch *(engl.* symmetric*) genau dann, wenn für alle $a, b \in A$ gilt: $(a, b) \in R \Rightarrow (b, a) \in R$.*
- antisymmetrisch *(engl.* antisymmetric*) genau dann, wenn für alle $a, b \in A$ gilt: $(a, b) \in R \ \wedge \ (b, a) \in R \Rightarrow a = b$.*
- asymmetrisch *(engl.* asymmetric*) genau dann, wenn für alle $a, b \in A$ gilt: $(a, b) \in R \Rightarrow (b, a) \neq R$. Jede asymmetrische Relation ist auch antisymmetrisch und nicht reflexiv. Ebenso ist jede nicht leere asymmetrische Relation nicht symmetrisch.*
- transitiv *(engl.* transitive*) genau dann, wenn für alle $a, b, c \in A$ gilt: $(a, b) \in R \ \wedge \ (b, c) \in R \Rightarrow (a, c) \in R$.*
- Äquivalenzrelation *(engl.* equivalence relation*) genau dann, wenn sie reflexiv, symmetrisch und transitiv ist.*
- Partielle Ordnung *oder* Halbordnung *(engl.* partial order*) genau dann, wenn sie reflexiv, antisymmetrisch und transitiv ist.*
- Totale Ordnung *(engl.* total order*) genau dann, wenn sie total, antisymmetrisch und transitiv ist. Jede totale Ordnung ist auch eine partielle Ordnung.*

Symmetrie und Asymmetrie schließen sich gegenseitig aus. Entsprechend schließen sich auch Äquivalenzrelation und partielle Ordnung gegenseitig aus.

Beispiel 7.4: Die uns wohlbekannten Relationen in $\mathbb{R}$ sind

	reflexiv	total	symmetrisch	antisymmetrisch	asymmetrisch	transitiv
$<$				✓	✓	✓
$>$				✓	✓	✓
$\leq$	✓	✓		✓		✓
$\geq$	✓	✓		✓		✓
$=$	✓		✓	✓		✓
$\neq$			✓			

◁

Partielle Ordnungen kann man in einem Hasse-Diagramm veranschaulichen.

Beispiel 7.5: Beispiele für eine partielle Ordnung sind:

- Abstammung in direkter Linie: wenn wir zusätzlich definieren, dass jede Person auch von sich selbst abstammt, ist dies eine partielle Ordnung.
- Teilmengenbeziehung "$\subseteq$".
- Die Eigenschaft von Kisten, ineinander zu passen (wenn man dazu noch annimmt, dass jede Kiste in sich selbst passt).

Für alle partiellen Ordnungen gilt, dass es nicht notwendigerweise *ein* "maximales" Element gibt, welches "größer oder gleich" als alle andere ist, obwohl es Elemente gibt, zu denen man kein "größeres" Element finden kann. ◁

Definition 7.5 (Äquivalenzklasse)**:** *Die Äquivalenzklasse eines Objektes a ist die Klasse der Objekte, die zu a äquivalent sind.*

Formal: Ist R eine Äquivalenzrelation auf einer Menge M, so nennt man für ein $a \in M$ die Teilmenge

$$[a] = \{x \in M \mid xRa\} \subseteq M$$

die Äquivalenzklasse von a.

Zur Beschreibung der Äquivalenzklasse schreibt man einen beliebigen *Repräsentanten* in eckigen Klammern.

Definition 7.6 (Partition)**:** *Sei A eine Menge. Dann heißen die Mengen B_i Partition von A genau dann, wenn gilt:*

$$\bigcup_i B_i = A$$
$$B_i \cap B_j = \emptyset, \text{ für } i \neq j$$
$$B_i \neq \emptyset.$$

Ist eine Äquivalenzrelation R auf der Menge M gegeben, dann bildet die Menge der Äquivalenzklassen eine Partition von M. Ebenso beschreiben die Mengen einer Partition eine Äquivalenzrelation und bilden deren Äquivalenzklassen.

7.2. Funktionen

Definition 7.7: *Eine* Funktion *oder* Abbildung *(engl.* mapping*) f ordnet jedem Element x seines* Definitionsbereichs *(engl.* domain*) D genau ein Element y seines* Wertebereichs *(engl.* codomain*) W zu.*

Damit ist eine Funktion eine linkstotale, rechtseindeutige Relation.

Wir beschreiben eine Funktion durch Angabe ihrer Definitions- und Wertemenge, sowie der Vorschrift der Abbildung. Die Schreibweise ist wie folgt: Sei D der Definitionsbereich, W der Wertebereich und $f(x)$ die Abbildungsvorschrift, so schreiben wir

$$f : D \to W$$
$$x \mapsto f(x).$$

Für $x \in D$ nennen wir $f(x)$ auch das Bild *(engl.* image*) von x. Das Bild der Funktion f ist $\{f(x) \mid x \in D\} \subseteq W$. Das* Urbild *(engl.* preimage*) von $y \in W$ bezeichnet ein $x \in D$, für das gilt: $f(x) = y$.*

Beachten Sie, dass sowohl der Name der Funktion als auch die Bezeichnung der Variable beliebig sind. Das heißt, $f(x) = x^2$ und $g(t) = t^2$ beschreiben ("sind") die gleiche Funktion.

Bei Funktionen sind die Eigenschaften injektiv, surjektiv und bijektiv von großer Bedeutung und kommen immer wieder vor. Wir beschreiben diese Eigenschaften erneut, diesmal in der Sprache der Funktionen.

Sei $f : D \to W$. Dann ist

- f injektiv genau dann, wenn jedes $y \in W$ höchstens ein Urbild hat. D.h., aus $f(x_1) = y = f(x_2)$ folgt $x_1 = x_2$. Injektiv heißt auch linkseindeutig.
- f surjektiv genau dann, wenn jedes $y \in W$ mindestens ein Urbild hat. D.h., zu jedem $y \in W$ gibt es ein $x \in D$ mit $f(x) = y$. Surjektiv heißt auch rechtstotal.
- f bijektiv genau dann, wenn es injektiv und surjektiv ist, also wenn jedes $y \in W$ genau ein Urbild hat.

Zu f existiert eine Umkehrfunktion (oft geschrieben als f^{-1}) genau dann, wenn f bijektiv ist.

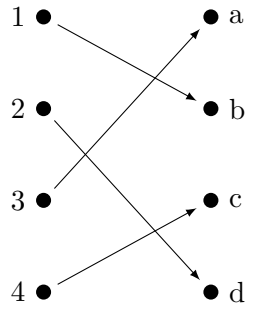


Abbildung 7: Funktion f_1

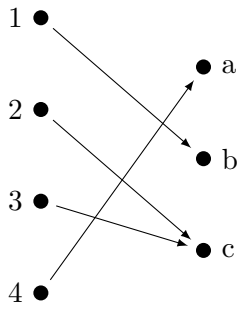


Abbildung 8: Funktion f_2

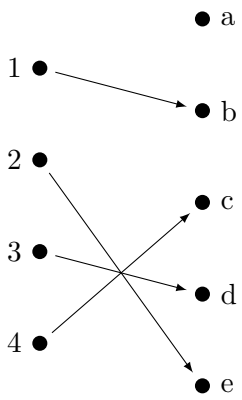


Abbildung 9: Funktion f_3

Beispiel 7.6: Funktion f_1 aus Abbildung 7 ist injektiv: bei jedem $b \in B$ kommt maximal ein Pfeil an. Ebenso ist sie surjektiv: alle $b \in B$ werden erreicht. Damit ist f_1 bijektiv.

Funktion f_2 aus Abbildung 8 ist ebenfalls surjektiv, aber nicht injektiv, da zwei verschiedene $a \in A$ auf das Element "c" abgebildet werden.

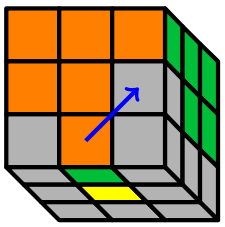
Funktion f_3 aus Abbildung 9 ist zwar injektiv, aber nicht surjektiv: Element "a" wird von keinem $a \in A$ erreicht. $\triangleleft$

Beispiel 7.7: Eigenschaften einiger bekannter Funktionen:

- $g_1 : [0, \pi] \to [-1, 1]$, $x \mapsto \cos x$, ist bijektiv.
- $g_2 : \mathbb{R} \to [-1, 1]$, $x \mapsto \cos x$, ist nur surjektiv, nicht aber injektiv.
- $g_3 : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, $x \mapsto x^2$ ist bijektiv, die Umkehrfunktion ist $g_3^{-1} : x \mapsto \sqrt{x}$.
- $g_4 : \mathbb{R} \to \mathbb{R}$, $x \to x^3$ ist bijektiv.
- $g_5 : \mathbb{R} \to \mathbb{R}$, $x \to x^3 - x^2$ ist nicht injektiv, aber surjektiv.
- $g_6 : \mathbb{R} \to \mathbb{R}$, $x \to e^x$ ist injektiv, aber nicht surjektiv.

$\triangleleft$

8. Gruppen und verwandte Strukturen



Falls Sie weitere Literatur zum Thema Gruppen, Ringe und Körper suchen, dann sehen diese beiden Skripte gut aus: [Gat10] und [Fil07]. Der Klassiker unter den Büchern ist [Fis14]. Wenn Sie viel Liebe für Algebra haben, dann ist [Fis11] auch sehr schön. [39]

8.1. Gruppen

Eine einfache algebraische Struktur, die schon erstaunlich viele interessante Möglichkeiten bietet, ist die *Gruppe* (engl. *group*).

Definition 8.1 (Gruppe)**:** *Eine Gruppe G ist eine nicht-leere Menge zusammen mit einer Verknüpfung $\circ$, die abgeschlossen und assoziativ ist. Es gibt ein neutrales Element $e \in G$ und zu jedem Element $a \in G$ ein Inverses $a^{-1} \in G$.*

Beispiel 8.1: Die Elemente der Drehgruppe $SO(2)$ sind die Drehungen um einen festen Punkt in der Ebene. Sei $G \subset SO(2)$ die Teilmenge der Drehungen um Vielfache von 72°. Klingt kompliziert, ist es aber nicht: ihre Elemente sind die Drehungen um 0°, 72°, 144°, 216° und 288°. Eine Drehung um 360° ist das Gleiche wie um 0° (das Ergebnis ist ja das gleiche), die hatten wir also schon. Wir schreiben:

$$G = \{0°, 72°, 144°, 216°, 288°\}.$$

Wir könnten es auch anders schreiben, um den Aspekt der zweidimensionalen Drehung zu verdeutlichen:

$$G = \{\uparrow, \nearrow, \searrow, \swarrow, \nwarrow\}.$$

Die Elemente sind die *Drehungen* selber, in dem Sinne, dass die Hintereinanderausführung z.B. zweier Drehungen um 72° eine Drehung um 144° ergibt. ◁

Oft benutzt für die Verknüpfung ist das Zeichen "$\circ$", gelesen "verknüpft". Den letzten Absatz können wir schreiben als $72° \circ 72° = 144°$.

Die formale Beschreibung für unser Beispiel ist also:

$$\circ : G \times G \to G$$
$$(a, b) \mapsto (a + b) \bmod 360.$$[40]

[39] Ein tolles Buch, aber viel zu mathematisch für das Bachelor-Studium Informatik. Wenn es Ihnen jedoch Spaß macht ... ☺

[40] Die Division mit Rest, Modulodivision, besprechen wir in Abschnitt 9.2.

Die *Abgeschlossenheit* der Verknüpfung bedeutet, dass durch die Verknüpfung immer ein Element erzeugt wird, welches selbst wieder in der Menge liegt. Das haben wir bereits durch die Beschreibung der Definitionsmenge und Zielmenge der Verknüpfung sichergestellt.

Für endliche Gruppen können wir die Verknüpfungstabelle (engl. *Cayley table*) explizit angeben. In unserem Falle sieht sie aus, wie in Abbildung 10 dargestellt.

	0°	72°	144°	216°	288°
0°	0°	72°	144°	216°	288°
72°	72°	144°	216°	288°	0°
144°	144°	216°	288°	0°	72°
216°	216°	288°	0°	72°	144°
288°	288°	0°	72°	144°	216°

Abbildung 10: Verknüpfungstabelle zu Beispiel 8.1

Weiterhin muss die Verknüpfung einer Gruppe *assoziativ* sein. Das heißt, dass die Reihenfolge der Ausführung der Operationen unerheblich ist [41], also für a, b, $c \in G$ gilt:

$$(a \circ b) \circ c = a \circ (b \circ c).$$

Ebenso muss es ein *neutrales Element* (engl. *identity element*) gegen, d.h. ein Element $e \in G$, für welches gilt:

$$a \circ e = e \circ a = a, \text{ für alle } a \in G.$$

Letztlich muss zu jedem $a \in G$ ein *inverses Element* (engl. *inverse element*) existieren (meist als a^{-1} geschrieben), für welches gilt:

$$a \circ a^{-1} = a^{-1} \circ a = e.$$

Eine Gruppe wird *kommutativ* (oder *abelsch* [42]) (engl. *commutative or abelian*) genannt, falls für alle a, $b \in G$ gilt:

$$a \circ b = b \circ a.$$

Wenn wir einen Blick auf unser Beispiel 8.1 werfen, so können wir an der Gruppentabelle überprüfen, dass die Verknüpfung assoziativ ist (das ist nicht ganz so einfach zu sehen); ebenso ist sie kommutativ (das erkennt man einfach: die Tabelle ist spiegelsymmetrisch zur Hauptdiagonalen). Das neutrale Element ist offensichtlich 0° und zu jedem Element existiert ein Inverses (in jeder Zeile und Spalte kommt das neutrale Element vor).

Wir können die Gruppenaxiome auch noch sparsamer formulieren: Es reicht anzunehmen, dass es ein linksneutrales Element e gibt, sodass für alle $a \in G$ gilt: $e \circ a = a$. Ebenso reicht es, linksinverse Elemente zu jedem $a \in G$ zu fordern, für die gilt: $a^{-1} \circ a = e$.

[41] Sie mögen glauben, dass das bei jeder Verknüpfung der Fall sein muss, aber das täuscht: $\circ : \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}$ mit $(a, b) \mapsto a - b$ ist nicht assoziativ, da im allgemeinen $(a - b) - c \neq a - (b - c)$ ist.

[42] Nach dem norwegischen Mathematiker Niels Henrik Abel, 1802–1829, einem Mitbegründer der Gruppentheorie

Zuerst zeigen wir, dass aus der Linkinversität, also $a^{-1} \circ a$, auch die Rechtsinversität, also $a \circ a^{-1}$, folgt.

$$\begin{aligned}
a^{-1} \circ a &= e \\
a^{-1} \circ a &= (a^{-1})^{-1} \circ a^{-1} \\
a^{-1} \circ a \circ a &= (a^{-1})^{-1} \circ a^{-1} \circ a \\
\underbrace{a^{-1} \circ a}_{=e} \circ a &= (a^{-1})^{-1} \circ \underbrace{a^{-1} \circ a}_{=e} \\
a &= (a^{-1})^{-1} \circ e \\
a \circ a^{-1} &= (a^{-1})^{-1} \circ e \circ a^{-1} \\
a \circ a^{-1} &= (a^{-1})^{-1} \circ a^{-1} \\
a \circ a^{-1} &= e.
\end{aligned}$$

□

Jetzt zeigen wir, dass ein linksneutrales Element auch rechtsneutral ist:

$$\begin{aligned}
e \circ a &= a \\
&= e \circ a \\
&= a \circ \underbrace{a^{-1} \circ a}_{=e} \\
&= a \circ e.
\end{aligned}$$

□

Als nächstes zeigen wir, dass das Inverse zum Inversen wieder das Element selbst ist:

$$\begin{aligned}
a &= a \\
&= e \circ a \\
&= ((a^{-1})^{-1} \circ a^{-1}) \circ a \\
&= (a^{-1})^{-1} \circ (a^{-1} \circ a) \\
&= (a^{-1})^{-1} \circ e \\
&= (a^{-1})^{-1}.
\end{aligned}$$

□

Man kann zeigen, dass es in einer Gruppe nur genau ein neutrales Element geben kann. Angenommen, es gäbe zwei neutrale Elemente e und f, für die beide gilt $e \circ a = a = a \circ e$ und $f \circ a = a = a \circ f$. Dann könnten wir schreiben:

$$\begin{aligned}
a \circ e &= a \circ f \\
\underbrace{a^{-1} \circ a}_{e \text{ oder } f} \circ e &= \underbrace{a^{-1} \circ a}_{e \text{ oder } f} \circ f \\
e \circ e &= e \circ f \\
e &= f.
\end{aligned}$$

□

Jetzt zeigen wir noch eine weitere Eigenschaft: sei G eine (nicht notwendigerweise kommutative) Gruppe und bezeichne e das neutrale Element der Gruppe. Dann gilt:

$$(a \circ b)^{-1} = b^{-1} \circ a^{-1}.$$

Beweis: Als erstes erweitern wir mit e, dem neutralen Element:

$$(a \circ b)^{-1} = e \circ (a \circ b)^{-1}$$

Da für jedes $c \in G$ gilt: $c^{-1} \circ c = e$, können wir schreiben:

$$= b^{-1} \circ b \circ (a \circ b)^{-1}$$

Das wiederholen wir noch einmal:

$$\begin{aligned} &= b^{-1} \circ e \circ b \circ (a \circ b)^{-1} \\ &= b^{-1} \circ a^{-1} \circ a \circ b \circ (a \circ b)^{-1} \\ &= b^{-1} \circ a^{-1} \circ (a \circ b) \circ (a \circ b)^{-1} \end{aligned}$$

Da $(a \circ b) \circ (a \circ b)^{-1} = e$, erhalten wir:

$$= b^{-1} \circ a^{-1}.$$ □

Man kann eine Gruppe beschreiben, indem man alle drei Teile ihrer Definition in einem Tupel nennt. Unser obiges Beispiel schreibt sich dann: $(G, \circ, 0^\circ)$. Das ist also die Menge G mit der Verknüpfung $\circ$ und 0° als neutralem Element. Manchmal lässt man bei der Definition auch das neutrale Element weg und gibt nur das Paar $(G, \circ)$ an. Das neutrale Element sollte dann aber offensichtlich sein.

Beispiel 8.2:

$(\mathbb{Z}, +, 0)$ ist eine Gruppe,
$(\mathbb{N}_0, +, 0)$ ist **keine** Gruppe,
$(\mathbb{Z} \setminus \{0\}, \cdot, 1)$ ist **keine** Gruppe,
$(\mathbb{Q} \setminus \{0\}, \cdot, 1)$ ist eine Gruppe,
$(\mathbb{R}, +, 0)$ ist eine Gruppe,
$(\mathbb{R} \setminus \{0\}, \cdot, 1)$ ist eine Gruppe.

◁

8.2. Verwandte Strukturen

Selbst wenn die Voraussetzungen für eine Gruppe teilweise nicht erfüllt sind, können Mengen noch algebraische Strukturen bilden, denen man einen Namen gegeben hat. Im Folgenden listen wir einige:

Definition 8.2 (Magma)**:** *Ein* Magma *(oder* Gruppoid*) ist eine Menge mit einer Verknüpfung, die abgeschlossen ist, aber nicht notwendigerweise assoziativ ist, ein neutrales Element enthält oder Inverse zu allen Elementen enthält.*

Beispiel 8.3: $\mathbb{R}^3$ mit der Vektormultiplikation $(a, b, c) \times (x, y, z) \coloneqq (bz - cy, cx - az, bx - ay)$ bildet ein Magma. ◁

Beispiel 8.4: Ebenso wird ein Magma gebildet von der Menge $M \coloneqq \{0, 1, 2\}$ und der Verknüpfungstabelle:

$*$	0	1	2
0	0	2	0
1	1	0	0
2	0	0	2

◁

Definition 8.3 (Halbgruppe)**:** *Eine* Halbgruppe *(engl.* semigroup*) ist ein Magma, dessen Verknüpfung assoziativ ist.*

D.h., eine Halbgruppe ist eine Menge mit einer Verknüpfung, die abgeschlossen und assoziativ ist. Ebenso ist eine Halbgruppe eine Verallgemeinerung einer Gruppe, ohne deren Anforderung von einem neutralen Element und Inversen.

Beispiel 8.5: Die Menge der positiven Ganzzahlen $\mathbb{N}$ zusammen mit der Addition bildet eine Halbgruppe. ◁

Beispiel 8.6: Die Menge der ganzen Zahlen $\mathbb{Z}$ zusammen mit der Maximumsfunktion $\max(a, b)$ bildet eine Halbgruppe. ◁

Definition 8.4 (Monoid)**:** *Ein* Monoid *ist eine Halbgruppe, die ein neutrales Element enthält.*

D.h., ein Monoid ist eine Menge zusammen mit einer abgeschlossenen Verknüpfung, die assoziativ ist und einem neutrales Element. Also ist ein Monoid eine Verallgemeinerung der Gruppe, ohne die Forderung von Inversen.

Beispiel 8.7: Die Menge der nicht-negativen Ganzzahlen $\mathbb{N}_0$ zusammen mit der Addition bildet eine Monoid. ◁

Beispiel 8.8: In der boolschen Algebra bildet die Menge $\{\mathsf{wahr}, \mathsf{falsch}\}$ mit der Verknüpfung `AND` ein Monoid. ◁

Übersicht über verschiedene algebraische Strukturen mit *einer* Verknüpfung:

	Verknüpfung	assoziativ	Neutrales	Inverse	kommutativ
Menge					
Magma	✓				
Halbgruppe	✓	✓			
Kommutative Halbgruppe	✓	✓			✓
Monoid	✓	✓	✓		
Kommutatives Monoid	✓	✓	✓		✓
Gruppe	✓	✓	✓	✓	
Abelsche Gruppe	✓	✓	✓	✓	✓

In der Beschreibung von Gruppen verwenden wir oft die Zahlzeichen “0” und “1”, um damit das neutrale Element der Addition bzw. der Multiplikation zu bezeichnen. Das wird häufig auch gemacht, wenn man sich nicht in $\mathbb{R}$ oder einer Untermenge davon befindet oder diese Operationen anders als üblich definiert sind. Wenn Sie so etwas schreiben, stellen Sie sicher, dass der Leser versteht, was Sie meinen. Benutzen Sie im Zweifel ein paar Zeichen mehr, um Missverständnisse zu vermeiden.

Wenn für die Verknüpfung das Zeichen "+" gewählt wurde, verwendet man oft das Zeichen "$-$" und meint damit die Addition des Inversen, z.B. $a - b = a + (-b)$.

In der Sprache der Gruppen verwendet man auch häufig "$\cdot$" statt "$\circ$" als Verknüpfung. Und entsprechend schreibt man dann auch statt $a \circ a \circ a$ gerne aaa oder direkt a^3 oder noch besser: statt $\underbrace{aa\ldots a}_{n\text{-mal}}\underbrace{bb\ldots b}_{n\text{-mal}}$ gleich $a^n b^n$.

9. Ringe und Körper

9.1. Ringe

Ein *Ring* (engl. *ring*) ist eine algebraische Struktur, in der man addieren und multiplizieren kann, aber nicht notwendigerweise dividieren. Ein Ring muss nicht kommutativ sein und muss keine 1 enthalten. Er ist damit eine Verallgemeinerung eines Körpers. $\mathbb{Z}$, $\mathbb{Q}$ und $\mathbb{R}$ sind Ringe, aber nur $\mathbb{Q}$ und $\mathbb{R}$ sind auch Körper.

Definition 9.1 (Ring)**:** *Eine Menge R mit zwei Operationen "$+$" und "$\cdot$" heißt* Ring *genau dann, wenn gilt:*

- $(R, +, 0)$ *ist eine abelsche Gruppe. Das neutrale Element der Addition heißt 0.*
- $(R, \cdot)$ *ist eine Halbgruppe, d.h., die Menge R mit der Verknüpfung "$\cdot$" ist assoziativ.*
- *die Distributivgesetze: für a, b, $c \in R$ gilt:*

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

Ist die Multiplikation kommutativ, heißt der Ring kommutativ*, existiert ein neutrales Element der Multiplikation, heißt er* Ring mit 1 *oder* unitär*.*

Beispiel 9.1: $\mathbb{Z}$ ist ein kommutativer Ring mit 1, ebenso $\mathbb{R}[x]$, die Menge aller Polynome über $\mathbb{R}$. Beides sind keine Körper. ◁

Definition 9.2 (Nullring)**:** *Der* Nullring *ist ein kommutativer Ring mit 1 und enthält als einziges Element die 0.*

Für den Nullring gilt $1 = 0$!

Definition 9.3 (Einheit)**:** *Sei R ein Ring mit 1. Dann heißt ein $x \in R$ eine* Einheit *(engl.* unit*) genau dann, wenn ein x^{-1} existiert, sodass $x \cdot x^{-1} = x^{-1} \cdot x = 1$.*

Die Menge aller Einheiten in einem Ring R bildet eine Gruppe, die Einheitengruppe *(engl.* group of units*). Oft schreibt man die Einheitengruppe als R^*.*

Beispiel 9.2: Die Einheitengruppe in $\mathbb{Z}$ ist $\mathbb{Z}^* = \{-1, 1\}$. ◁

Definition 9.4 (Nullteiler)**:** *In einem Ring R ist ein* Nullteiler *ein $0 \neq a \in R$, für welches ein $b \neq 0$ existiert, sodass $ab = 0$.*

Ein Nullteiler ist keine Einheit.

9.2. Division mit Rest

Dies ist eine gute Stelle, um die *Division mit Rest* einzuführen. Sie findet ihre Anwendung z.B. in den Restklassenringen $\mathbb{Z}/n\mathbb{Z}$, $n \in \mathbb{N}$, Zahlentheorie, aber auch in vielen informatischen Themen, speziell in der Kryptografie.

Als das Ergebnis der *Modulo Division* bezeichnen wir den nicht-negativen Rest der Division zweier ganzer Zahlen a und q. Im folgenden berechnen wir a/q und erhalten als Ergebnis den Ganzzahlquotienten n und den Rest r.

Definition 9.5 (Division mit Rest)**:** *Sei der Dividend $a \in Z$ und Divisor $d \in \mathbb{Z}$, $d \neq 0$ gegeben. Dann existiert ein eindeutig bestimmter Ganzzahlquotient $n \in \mathbb{Z}$ und Rest $r \in \mathbb{N}_0$, sodass gilt:*

$$a = n \cdot d + r, \ \textit{mit} \ 0 \leq r < |d|.$$

Sind wir nur am Rest der Division interessiert, so schreiben wir $r = a \bmod d$.

Beispiel 9.3: Einige Zahlenbeispiele:

$$\begin{aligned} 5 \bmod 3 &= 2 && (5 = 1 \cdot 3 + 2) \\ 100 \bmod 99 &= 1 && (100 = 1 \cdot 99 + 1) \\ 100 \bmod 5 &= 0 && (100 = 20 \cdot 5 + 0) \\ -5 \bmod 3 &= 1 && (-5 = -2 \cdot 3 + 1) \\ -5 \bmod -3 &= 1 && (-5 = 2 \cdot -3 + 1) \\ 7 \bmod 9 &= 7 && (7 = 0 \cdot 9 + 7) \end{aligned}$$

$\lhd$

Die Modulo-Division wird z.B. zum Test der Teilbarkeit benutzt: wenn $a \bmod q = 0$ gilt, dann ist per Definition a durch d (ohne Rest) *teilbar*, wir schreiben dafür auch $d \mid a$, gelesen "d teilt a", ansonsten schreiben wir $d \nmid a$.

Offensichtlich ist $a \bmod 1 = 0$ für alle $a \in \mathbb{Z}$.

Wenn wir $d = 0$ erlauben würden, gäbe es keine Lösung, da $r < |d|$ verlangt ist. Entsprechend ist es auch hier (wie bei der normalen Division) nicht erlaubt.

Wir benutzen hier die Konvention, dass der Rest r nicht-negativ ist, sowie $0 \leq r < |d|$. Es gibt auch andere Konventionen, speziell in Programmiersprachen, z.B. $|r| < |d|$. Wenn Sie beim Programmieren die Modulo-Division auf möglicherweise negativen a oder d benutzen, ist es eine gute Idee, Test Code zu schreiben, der diese Fälle überprüft.

In C/C++ und Python schreibt sich der Modulo-Operator als `%`, also z.B. `r = a % d`.

Sind wir jedoch nur am Ganzzahlquotienten n interessiert, so schreiben wir $n = \lfloor a/d \rfloor$. Die Zeichen "$\lfloor\ \rfloor$" nennen sich *Gauß-Klammer* oder *Abrundungsfunktion* (engl. *floor function*). Es gilt für alle $k \in \mathbb{Z}$ und $x \in \mathbb{R}$:

$$k \leq \lfloor x \rfloor \Leftrightarrow k \leq x.$$

Analog gibt es auch eine *Aufrundungsfunktion* (engl. *ceiling function*): es gilt für alle $k \in \mathbb{Z}$ und $x \in \mathbb{R}$:

$$k \geq \lceil x \rceil \Leftrightarrow k \geq x.$$

Die Gauß-Klammern werden in C/C++ und Python als `floor(x)` bzw. `ceil(x)` geschrieben. Beachten Sie, dass bei einer Konvertierung von Floating Point auf Integer ohne weiteres nicht klar ist, ob negative Zahlen nach unten (`floor()`) oder zur 0 hin gerundet werden. Auch wieder ein guter Punkt für Test Code.

9.3. Polynome

Auch die *Polynome* bilden einen Ring, den *Polynomring* über einem Zahlbereich.

Definition 9.6 (Polynom)**:** *Sei $n \in \mathbb{N}_0$ und $a_0, a_1, \ldots, a_n \in \mathbb{R}$. Einen Term der Form*

$$a_0 x^0 + a_1 x^1 + \ldots + a_n x^n$$

bezeichnen wir als Polynom *(engl.* polynomial*).*

Durch Nutzung des Summenzeichen $\sum$ können wir ein Polynom auch wie folgt formulieren:

$$P(x) = \sum_{i=0}^{n} a_i x^i.$$

Die Polynome in der Unbestimmten x über dem Zahlbereich $\mathbb{R}$ bilden einen kommutativen Ring mit 1 (Polynomring), der $\mathbb{R}[x]$ geschrieben und "$\mathbb{R}$ adjungiert x" gesprochen wird.

Definition 9.7 (Grad eines Polynoms)**:** *Der* Grad *(engl.* degree*) eines Polynoms (in einer Unbestimmten) ist der größte Exponent der Summanden des Polynoms. Den Grad eines Polynoms P schreibt man als* $\deg(P)$*. Der Grad des Nullpolynoms $P(x) = 0$ wird entweder als -1 oder als $-\infty$ definiert.*

Wenn das Polynom $P(x)$ durch seine Nullstellen (mit *Vielfachheit*) $b_1, b_2, \ldots, b_n$ gegeben ist, können wir es als Produkt

$$P(x) = (x - b_1) \cdot (x - b_2) \cdot \ldots \cdot (x - b_n)$$

schreiben.

Die Kurzschreibweise mittels Produktzeichen ist

$$P(x) = \prod_{i=1}^{n} (x - b_i). \qquad (*)$$

Denken Sie an Punkt-vor-Strichrechnung! Da das $\prod$-Zeichen eine Abfolge von Faktoren darstellt, ist folgende Darstellung von $P(x)$ falsch:

$$\prod_{i=1}^{n} x - b_i, \qquad \lightning\lightning$$

da sie ausgeschrieben $x - (b_1 \cdot x) - (b_2 \cdot x) - \ldots - (b_{n-1} \cdot x) - b_n$ ergäbe. Richtig ist die Darstellung (*).

Beachten Sie weiterhin, dass ein Polynom n-ten Grades aus $n+1$ *Monomen* (engl. *monomial*) (Summanden) besteht. Umgekehrt heißt das, dass ein Polynom aus n Monomen, deren Grad von 0 sukzessive aufsteigt, den Grad $n - 1$ hat.

Beispiel 9.4:

$$\begin{aligned}
P_1(x) &= x^2 - x + 5 \text{ besteht aus 3 Monomen und } \deg(P_1) = 2,\\
P_2(x) &= 2x - 1 \text{ besteht aus 2 Monomen und } \deg(P_2) = 1,\\
P_3(x) &= 8 \text{ besteht aus einem Monom und } \deg(P_3) = 0,\\
P_4(x) &= x^3 - 1 \text{ besteht aus 2 Monomen und } \deg(P_4) = 3.
\end{aligned}$$

9.4. Polynomdivision

Der Polynomring $\mathbb{R}[x]$ ist ein kommutativer Ring mit 1, der aber kein Körper ist. D.h., es existiert nicht zu jedem $a(x) \in \mathbb{R}[x]$ ein inverses Element $a(x)^{-1}$, sodass $a(x) \cdot a(x)^{-1} = 1$ gilt. Trotzdem können wir eine Division mit Rest ausführen (siehe Abschnitt 9.2).

Seien $P(x)$, $d(x)$, $a(x)$ und $r(x) \in \mathbb{R}[x]$. Dann können wir $P(x)$ folgendermaßen zerlegen:

$$P(x) = a(x) \cdot d(x) + r(x),$$

mit $\deg(r(x)) < \deg(d(x))$.

Wir benutzen die Polynomdivision beispielsweise, wenn wir ein nicht-konstantes Polynom in seine Linearfaktoren zerlegen. Ein *Linearfaktor* $F(x) \in \mathbb{R}[x]$ ist $F(x) = x - b$, wobei $b \in \mathbb{R}$ eine Nullstelle des Polynoms $P(x)$ ist, d.h. $P(b) = 0$ gilt.

Die Idee hinter der Nullstellenberechnung mittels Polynomdivision ist einfach. Man versucht, ein Polynom der Form

$$P(x) = a_0 x^0 + a_1 x^1 + \ldots + a_n x^n$$

in ein Produkt der Form

$$P(x) = (x - b_1) \cdot (x - b_2) \cdot \ldots \cdot (x - b_n)$$

umzuschreiben. Aus der Schule kennen Sie bereits folgende Regel: "Ein Produkt ist dann 0, wenn mindestens ein Faktor 0 ist". Somit sind b_1, b_2, $\ldots$, b_n die Nullstellen des Polynom n-ten Grades.

Allgemein gibt es keine geschlossene Formel, mit der man Nullstellen von beliebigen Polynomen finden kann. [43] Wenn aber eine Nullstelle b bekannt ist, [44] kann man das Polynom durch $(x - b)$ dividieren und erhält ein neues Polynom kleineren Grades. Dies kann man fortführen, so lange man weitere Nullstellen findet. [45]

Das Vorgehen bei der Polynomdivision ist ähnlich dem bei der schriftlichen Division. Wir veranschaulichen das Verfahren zur Polynomdivision an einem Beispiel.

Beispiel 9.5: Sei

$$P(x) = x^3 + 3x^2 - x - 3.$$

Irgendwie ermitteln wir eine Nullstelle bei $x = 1$. Jetzt dividieren wir das Polynom durch $(x - 1)$:

$$\begin{array}{rl} \left(x^3 + 3x^2 - x - 3 \right) : \left(x - 1 \right) = x^2 + 4x + 3 \\ \underline{-x^3 + x^2} \\ \quad 4x^2 - x \\ \quad \underline{-4x^2 + 4x} \\ \quad\quad 3x - 3 \\ \quad\quad \underline{-3x + 3} \\ \quad\quad\quad 0 \end{array}$$

[43] Für Polynome zweiten, dritten und vierten Grades gibt es allgemeine Formeln, für höhere Grade bewiesenermaßen nicht mehr.

[44] Wir sagen hier nichts darüber, *wie* man diese Nullstelle findet. Das ist ein anderes und schwieriges Thema.

[45] Polynome können auch vielfache Nullstellen haben, z.B. hat x^2 eine doppelte Nullstelle bei $x = 0$.

Wie zu erwarten, geht die Division auf. Jetzt können wir das Verfahren wiederholen: ermittle eine weitere Nullstelle bei $x = -1$ und dividiere das Restpolynom x^2+4x+3 durch $(x+1)$:

$$\begin{array}{rl} \Big(\quad x^2 + 4x + 3 \Big) : \Big(x + 1 \Big) = x + 3 \\ \underline{- x^2 \quad - x} \\ \qquad\quad 3x + 3 \\ \qquad\quad \underline{- 3x - 3} \\ \qquad\qquad\quad 0 \end{array}$$

Wir haben erfolgreich dividiert und die letzte Nullstelle bei $x = -3$ lässt sich einfach ablesen. Somit haben wir drei Nullstellen $x_1 = 1$, $x_2 = -1$, $x_3 = -3$ gefunden. ◁

Beachten Sie, dass die Division durch den Linearfaktor einer Nullstelle immer aufgeht. Dass allerdings das Polynom vollständig in Linearfaktoren über $\mathbb{R}$ zerfällt, ist nicht selbstverständlich. Z.B. hat das Polynom $x^2 + 1$ keine Nullstelle in $\mathbb{R}$. Man sagt, es ist *irreduzibel*. Erst in $\mathbb{C}$ zerfällt jedes Polynom in Linearfaktoren. In unserem Beispiel wäre das $x^2 + 1 = (x - i)(x + i)$. [46]

9.5. Körper

Ein *Körper* (engl. *field*) ist ein kommutativer Ring mit 1, der nicht der Nullring ist und in dem es zu jedem Element $\neq 0$ ein Inverses gibt. Unsere "normalen" Zahlen, $\mathbb{R}$, bilden einen Körper, ebenso $\mathbb{Q}$ und $\mathbb{C}$.

Definition 9.8 (Körper)**:** *Eine Menge K mit zwei Operationen "$+$" und "$\cdot$" heißt* Körper *genau dann, wenn gilt:*

- $(K, +, 0)$ *ist eine abelsche Gruppe. Das neutrale Element der Addition heißt 0.*
- $(K \setminus \{0\}, \cdot, 1)$ *ist eine abelsche Gruppe. Das neutrale Element der Multiplikation heißt 1.*
- *die Distributivgesetze: für a, b, $c \in K$ gilt:*

$$a \cdot (b + c) = a \cdot b + a \cdot c \quad \text{und} \quad (a + b) \cdot c = a \cdot c + b \cdot c. \quad [47]$$

In einem Körper K muss zu jedem Element $a \in K \setminus \{0\}$ ein multiplikatives Inverses existieren, also ein x, für das gilt: $a \cdot x = 1$. Damit ist in einem Körper K die Einheitengruppe $K^* = K \setminus \{0\}$.

Ein Körper kann *Unterkörper* beinhalten, das sind Teilmengen, die ihrerseits wieder Körper sind. Zum Beispiel gilt $\mathbb{Q} \subset \mathbb{R} \subset \mathbb{C}$.

Ein Körper mit endlich vielen Elementen heißt *endlicher Körper* (engl. *finite field*). Ist p eine Primzahl und $n \in \mathbb{N}$, so existiert ein Körper mit p^n Elementen. Alle Körper mit p^n Elementen sind (bis auf eine Umbenennung ihrer Elemente) gleich (*isomorph*). Man nennt diese Körper auch $\mathbb{F}_{p^n}$ oder $GF(p^n)$ (für *general field*).

[46] Zur Definition von $\mathbb{C}$ siehe Kapitel 4.

[47] Eines der Distributivgesetze hätte auch gereicht. Das andere folgt aus der Kommutativität der Multiplikation.

Endliche Körper spielen eine wichtige Rolle in der Kryptographie, z.B. beim RSA-Algorithmus oder beim diskreten Logarithmus.

Im Falle $n = 1$ existiert eine bijektive Abbildung zwischen $\mathbb{F}_p$ und der Menge $F = \{a \in \mathbb{N}_0 \mid a < p\} \simeq \mathbb{Z}/n\mathbb{Z}$ mit den Verknüpfungen

$$+ : F \times F \to F \text{ mit } (x, y) \mapsto (x + y) \bmod p, \quad \text{sowie}$$

$$\cdot : F \times F \to F \text{ mit } (x, y) \mapsto (x \cdot y) \bmod p.$$

Beispiel 9.6: Betrachten wir $\mathbb{F}_3$, den Körper mit drei Elementen. Wir benutzen zur Darstellung der Elemente die Menge $F := \{0, 1, 2\}$. Addition und Multiplikation sind definiert wie in $\mathbb{N}_0$, nur mit einer mod 3-Operation nach der Addition bzw. Multiplikation. Dann ergeben sich folgende Verknüpfungstafeln:

$+$	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

Abbildung 11: Verknüpfung "+"

$\cdot$	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

Abbildung 12: Verknüpfung "·"

◁

Die endlichen Körper $\mathbb{F}_{p^n}$ mit $n > 1$ sind schwieriger aufzubauen als der Fall $n = 1$ und sind auch nicht isomorph zu $\mathbb{Z}/p^n\mathbb{Z}$, welches ja eben kein Körper ist, da z.B. p ein Nullteiler ist.

9.6. Bruchrechnung

Jetzt haben wir die theoretische Grundlage, auf so etwas "einfaches" wie Bruchrechnung zurückzuschauen. Eine kurze Wiederholung:

Die Menge der rationalen Zahlen $\mathbb{Q}$ ist definiert als:

$$\mathbb{Q} = \left\{ \frac{a}{b} \mid a, b \in \mathbb{Z}, b \neq 0 \right\}.$$

Elemente der Menge $\mathbb{Q}$ können wir als Bruch schreiben, das heißt als $\frac{a}{b}$ oder a/b. Dabei nennen wir a den *Zähler* (engl. *numerator*) und b den *Nenner* (engl. *denominator*).[48]

Wir schreiben zwar a/b, aber erinnern uns an Abschnitt 9.1, dass es sich hierbei um eine Kurzschreibweise handelt. Sei a, b, $x \in \mathbb{Z}$ mit $b \neq 0$. Lösen wir folgende Gleichung:

$$\begin{aligned} a &= x \cdot b \\ a \cdot b^{-1} &= x \cdot b \cdot b^{-1} \\ a \cdot b^{-1} &= x. \end{aligned}$$

Das heißt, die Lösung heißt $x = a \cdot b^{-1}$. Um es uns einfacher zu machen, schreiben wir $a/b := a \cdot b^{-1}$. [49]

[48] Gedruckt schreibt man oft auch a/b, da es sich besser liest. Dabei bedeutet $a/b \cdot c = \frac{a}{b \cdot c}$. Vermeiden Sie jedoch zu schreiben $a/b + c$ (also ohne Klammern!). Für den Leser ist unklar, was Sie damit meinen.

[49] Wir werden die Potenz x^{-1} in Abschnitt 10.2 wieder treffen.

Multiplikation von Brüchen

Seien a, b, c, $d \in \mathbb{Z}$, $b \neq 0 \neq d$. Dann gilt:

$$\begin{aligned}\frac{a}{b} \cdot \frac{c}{d} &= a \cdot b^{-1} \cdot c \cdot d^{-1} \\ &= (a \cdot c) \cdot (b^{-1} \cdot d^{-1}) \\ &= (a \cdot c) \cdot (d \cdot b)^{-1} \\ &= \frac{a \cdot c}{b \cdot d}.\end{aligned}$$

Wir hatten in Kapitel 8 gesehen, warum $(a \cdot b)^{-1} = b^{-1} \cdot a^{-1}$ ist. Hier könnten wir die Faktoren natürlich vertauschen, da die Multiplikation in $\mathbb{R}$ kommutativ ist.

Division von Brüchen

Seien a, b, c, $d \in \mathbb{Z}$, $b \neq 0 \neq d$. Man dividiert durch einen Bruch, indem man mit dem Kehrwert multipliziert. Warum? Darum:

$$\begin{aligned}\frac{a}{b} : \frac{c}{d} &= (a \cdot b^{-1}) : (c \cdot d^{-1}) \\ &= a \cdot b^{-1} \cdot (c \cdot d^{-1})^{-1} \\ &= a \cdot b^{-1} \cdot (d^{-1})^{-1} \cdot c^{-1} \\ &= a \cdot b^{-1} \cdot d \cdot c^{-1} \\ &= a \cdot d \cdot b^{-1} \cdot c^{-1} \\ &= a \cdot d \cdot (c \cdot b)^{-1} \\ &= \frac{a \cdot d}{c \cdot b} \\ &= \frac{a \cdot d}{b \cdot c}.\end{aligned}$$

□

Erweitern von Brüchen

Sei a, $b \in \mathbb{Z}$, $b \neq 0$, $\mathbb{Z} \ni k \neq 0$:

$$\frac{a}{b} = \frac{a}{b} \cdot 1 = \frac{a}{b} \cdot \frac{k}{k} = \frac{ak}{bk}.$$

Wir multiplizieren den Bruch a/b hier geschickt mit 1, um ihn auf den Nenner $b \cdot k$ zu bringen. Dies wird sich noch als nützlich erweisen. Wir nennen das *Erweitern* des Bruchs.

Kürzen von Brüchen

Sei x, y, $k \in \mathbb{Z}$, $y \neq 0 \neq k$. Sei $a = xk$, $b = yk$. Dann gilt:

$$\frac{a}{b} = \frac{xk}{yk} = \frac{x}{y} \cdot \frac{k}{k} = \frac{x}{y} \cdot 1 = \frac{x}{y}.$$

Den beschriebenen Vorgang nennen wir das *Kürzen* (engl. *to cancel out*) des Bruches. Da a und b beides ganzzahlige Vielfache von k sind, können wir beide durch k teilen.

Den maximalen Faktor, den man kürzen kann, nennt man den größten gemeinsamen Teiler (engl. *greatest common denominator*), oder kurz *ggT* (engl. *gcd*). Berechnen kann man

diesen mit dem *Euklidischen Algorithmus*, [50] laut Knuth dem ältesten Algorithmus der Menschheit.

Beispiel 9.7:

$$\frac{3}{7} = \frac{3}{7} \cdot \frac{4}{4} = \frac{12}{28} \qquad \text{(Erweitern mit 4)}$$

$$\frac{6}{8} = \frac{3 \cdot 2}{4 \cdot 2} = \frac{3}{4} \cdot \frac{2}{2} = \frac{3}{4} \cdot 1 = \frac{3}{4} \qquad \text{(Kürzen von 2)}$$

◁

Addition von Brüchen

Wir addieren Brüche, indem wir sie auf den gleichen Nenner bringen (*Hauptnenner* genannt) und dann die jeweiligen Zähler addieren. Seien a, b, c, $d \in \mathbb{Z}$, $b \neq 0 \neq d$. Dann gilt:

$$\frac{a}{b} + \frac{c}{d} = \frac{ad}{bd} + \frac{cb}{db} = \frac{ad + bc}{bd}.$$

[50] Euklid von Alexandria, griechischer Mathematiker, der wahrscheinlich im 3. Jahrhundert v. Chr. in Alexandria gelebt hat.

10. Was Sie noch wissen sollten

The greatest shortcoming of the human race
is our inability to understand the exponential function.

— Albert A. Bartlett [51]

10.1. Betrag

Definition 10.1 (Betrag)**:** *Der Betrag (engl.* absolute value*) einer Zahl $a \in \mathbb{R}$ ist definiert als:*

$$|a| = \begin{cases} a & \text{falls } a \geq 0, \\ -a & \text{falls } a < 0. \end{cases}$$

Beispiel 10.1:

$$\begin{aligned} |4| &= 4 \\ |-2.718| &= 2.718 \\ |x| = \pi &\Leftrightarrow x = \pm\pi \\ |x+2| = 1 &\Leftrightarrow \pm(x+2) = 1 \Leftrightarrow x = -1 \vee x = -3 \end{aligned}$$

◁

Für den Betrag gelten folgende Gesetze:

Seien a, $b \in \mathbb{R}$. Dann gilt:

$$\begin{aligned} |a| &\geq 0, && \text{(Positivität)} \\ |a \cdot b| &= |a| \cdot |b|, && \text{(Homogenität)} \\ |a+b| &\leq |a| + |b|. && \text{(Dreiecksungleichung)} \end{aligned}$$

Umformungen ist bei Beträgen schwierig, da z.B. Quadrieren keine Äquivalenzumformung ist. Daher muss man oft eine Fallunterscheidung machen, siehe Kapitel 6.

Damit hängt zusammen, dass das Auflösen eines Quadrats durch das Ziehen den Wurzel Betragsstriche erfordert. Schauen Sie:

$$\begin{aligned} (x+1)^2 &= 1 \\ |x+1| &= \sqrt{1} \end{aligned}$$

1. Fall: $(x+1) \geq 0$. Wir können die Betragsstriche einfach weglassen:

$$\begin{aligned} x+1 &= 1 \\ x &= 0. \end{aligned}$$

2. Fall: $(x+1) < 0$. Wir müssen das Argument des Betrags negieren:

$$\begin{aligned} -(x+1) &= 1 \\ -x-1 &= 1 \\ x+1 &= -1 \\ x &= -2. \end{aligned}$$

D.h., ohne eine Fallunterscheidung kommen wir hier nicht weiter.

[51] Amerikanischer Physiker, 1923–2013. Zu sehen auf Youtube.

10.2. Potenzen

Wie wir Multiplizieren als mehrfaches Addieren auffassen, so wird Potenzieren als mehrfaches Multiplizieren verstanden.

Definition 10.2 (Potenzen für nicht-negative ganzzahlige Exponenten)**:** *Sei $a \in \mathbb{R}$, $n \in \mathbb{N}_0$. Die n-te Potenz (engl. n-th* power*) von a ist definiert als*

$$a^n := 1 \cdot \underbrace{a \cdot a \cdot \ldots \cdot a}_{n\text{-mal}} = \prod_{i=1}^{n} a.$$

Wir nennen a die *Basis* (engl. *basis*) und n den *Exponenten* (engl. *exponent*). Wenn $n = 0$, dann stehen dort 0 Multiplikationen mit a, also ist $a^0 = 1$. Weiterhin ist $0^n = 0$, für $n > 0$. Der Wert von 0^0 ist allgemein nicht definiert. [52] Das Potenzieren bindet stärker als die Multiplikation, es gilt also $ab^c = a(b^c)$. Die Operation ist nicht kommutativ, im allgemeinen gilt also $a^b \neq b^a$. Potenzen werden von rechts ausgewertet, also $a^{b^c} = a^{(b^c)}$.

Daraus lassen sich sofort einige Gesetze ableiten: Seien a, $b \in \mathbb{R}$ und m, $n \in \mathbb{N}_0$, dann gilt:

$$\begin{aligned}
a^n \cdot a^m &= a^{n+m}, \\
a^n \cdot b^n &= (a \cdot b)^n, \\
\frac{a^n}{b^n} &= \left(\frac{a}{b}\right)^n && \text{für } b \neq 0, \\
\frac{a^n}{a^m} &= a^{n-m} && \text{für } a \neq 0, n \geq m, \\
(a^n)^m &= a^{n \cdot m}.
\end{aligned}$$

Definition 10.3 (Potenzen für ganzzahlige Exponenten)**:** *Für den Fall $n - m = -1$ ergibt sich bei $a \neq 0$:*

$$\frac{a^n}{a^m} = \frac{1}{a} =: a^{-1}.$$

Darüber erweitern wir den Begriff der Potenz auf ganzzahlige Exponenten.

Die oben genannten Gesetze gelten dann ohne Einschränkung auch für n, $m \in \mathbb{Z}$.

Ist $a < 0$, dann ist a^n bei geradzahligem Exponenten n positiv, bei ungeradzahligem Exponenten negativ.

Daraus ergibt sich für $a = -1$ und $n \in \mathbb{Z}$, dass $(-1)^{2n} = 1$ und $(-1)^{2n-1} = -1$ ist. [53]

10.3. Wurzeln

Definition 10.4 (Wurzeln für natürliche Exponenten)**:** *Seien $a \in \mathbb{R}_{\geq 0}$, $x \in \mathbb{R}$, $n \in \mathbb{N}$. Dann besitzt die Gleichung $x^n = a$ eine eindeutig bestimmte nicht-negative Lösung in $x = \sqrt[n]{a}$. Damit lässt sich das Wurzelziehen als Umkehrung des Potenzierens auffassen. Wir nennen dies die n-te* Wurzel *(engl. n-th* root*) von a. Die Zahl a heißt auch* Radikand *(engl.* radicand*). Ist $n = 2$, sprechen wir von einer* Quadratwurzel *(engl.* square root*), bei $n = 3$ von einer* Kubikwurzel *(engl.* cube root*). Ist n nicht angegeben, gilt $n = 2$.*

[52] Mehr dazu finden Sie auf Wikipedia.
[53] Beachten Sie die Klammerung! $-1^n = -(1^n)$!

Offensichtlich gilt $\sqrt[1]{a} = a$. Weiterhin gilt $\sqrt[n]{1} = 1$ und $\sqrt[n]{0} = 0$.

Ebenso gilt $\sqrt[n]{a^n} = a$. Das Ziehen der n-ten Wurzel wirkt also wie das Potenzieren mit $1/n$, daher schreiben wir auch

$$\sqrt[n]{a} := a^{1/n}.$$

Mit dieser Definition können wir aus den Potenzgesetzen direkt die Wurzelgesetze ableiten: Seien a, $b \in \mathbb{R}_{\geq 0}$, n, $k \in \mathbb{N}$, $m \in \mathbb{Z}$. Dann gilt:

$$\begin{aligned}
\sqrt[n]{a^m} &= (\sqrt[n]{a})^m = a^{m/n}, \\
\sqrt[n]{a^n} &= a = (\sqrt[n]{a})^n, \\
\sqrt[n]{a} \cdot \sqrt[n]{b} &= \sqrt[n]{a \cdot b}, \\
\frac{\sqrt[n]{a}}{\sqrt[n]{b}} &= \sqrt[n]{\frac{a}{b}} && \text{für } b \neq 0, \\
\sqrt[n]{\sqrt[k]{a}} &= \sqrt[n \cdot k]{a} = \sqrt[k]{\sqrt[n]{a}}.
\end{aligned}$$

Die Quadratwurzel ist immer nicht-negativ. Insofern ist sie nur die Umkehrung des Quadratfunktion für $x \in \mathbb{R}_{\geq 0}$.

Sei $a \in \mathbb{R}_{<0}$, $x \in \mathbb{R}$ und $n \in \mathbb{N}$ und ungerade, so besitzt $x^n = a$ eine eindeutige *negative* Lösung. Zum Beispiel ist für $x^3 = -125$ die Lösung $x = -5$. Somit kann man den Wurzelbegriff noch weiter verallgemeinern für ungerade ganzzahlige Exponenten.

In der Literatur wird teilweise trotzdem gefordert, dass eine n-te Wurzel immer positiv ist und damit folgende Aussage wahr wäre: $\sqrt[3]{-8} = 2$. Sollten solche Sachverhalte in einer Argumentation wichtig sein, helfen einige erklärende Worte.

Mittels $\sqrt[n]{a} = a^{1/n}$ haben wir die Potenzgesetze schon auf Exponenten aus $\mathbb{Q}$ verallgemeinert. Man kann zeigen, dass sie sich noch weiter auf $\mathbb{R}$ verallgemeinern lassen.

Eine Liste der Potenzgesetze mit allen Bedingungen findet sich in Anhang D.

Schauen Sie mal, ob Sie den Fehler in dieser Rechnung finden:

$$-27 = (-27)^{(2/3)\cdot(3/2)} = ((-27)^{2/3})^{3/2} = (729^{1/3})^{3/2} = 9^{3/2} = 27. \quad \lightning^{54}$$

10.4. Fakultät

Definition 10.5 (Fakultät)**:** *Sei $n \in \mathbb{N}_0$, dann ist die Fakultät von n (engl.* factorial*) (gesprochen "n Fakultät") definiert als:*

$$n! := \prod_{i=1}^{n} i.$$

Weiterhin ist $0! = 1$, das ergibt sich aus der Definition des leeren Produkts.

[54] Hinweise zur Lösung siehe siehe Wikipedia.

Beispiel 10.2:

$$6! = \prod_{i=1}^{6} i = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 = 720.$$

◁

Die Fakultätsfunktion wird häufig in der Kombinatorik benötigt (Permutation, Binomialkoeffizient). Ebenso kommt sie vor bei der Berechnung von Taylorreihen, die damit Formeln zur Berechnung von trigonometrischen Funktionen oder der Exponentialfunktion liefern, siehe nächster Abschnitt.

10.5. Exponentialfunktion

Definition 10.6 (Exponentialfunktion)**:** *Die* Exponentialfunktion *(engl.* exponential function*) bezeichnet die Funktion $x \mapsto a^x$ für $a \in \mathbb{R}_{>0}$, $a \neq 1$.*

Als natürliche *(oder auch nur* ***die****) Exponentialfunktion bezeichnen wir die Funktion $x \mapsto e^x$, wobei e die Eulersche Zahl [55] (2.71828...) ist. Wir schreiben auch $e^x = \exp(x)$.*

Es gelten die gleichen Gesetze wie wir sie schon in Abschnitten 10.2 und 10.3 gesehen haben, nur liegt jetzt der Fokus auf den Exponenten, nicht auf den Basen.

Man kann $\exp(x)$ schreiben als eine unendliche Reihe:

$$\exp(x) \coloneqq \sum_{i=0}^{\infty} \frac{x^i}{i!},$$

wobei $n!$ die Fakultätsfunktion ist, siehe Abschnitt 10.4.

Damit können wir den Wert von e errechnen, also $e = \sum_{i=0}^{\infty} \frac{1}{i!}$.

Die Exponentialfunktion erfüllt die Funktionalgleichung $\exp(x + y) = \exp(x) \cdot \exp(y)$, was wir schon bei den Potenzgesetzen als $a^{n+m} = a^n \cdot a^m$ gesehen hatten.

Man kann zeigen, dass a^x für $a > 1$ und $x \to \infty$ schneller wächst als jede Potenz x^b, $b \in R$ beliebig. D.h., es gilt

$$\lim_{x \to \infty} \frac{a^x}{x^b} = \infty.$$

Das wird noch wichtig werden in Kapitel 11.

10.6. Logarithmus

Die Umkehrung der Exponentialfunktion ist der *Logarithmus* (engl. *logarithm*).

Definition 10.7 (Logarithmus)**:** *Seien a, $b \in \mathbb{R}_{>0}$ mit $b \neq 1$ und $x \in \mathbb{R}$. Dann hat die Gleichung $b^x = a$ die eindeutige Lösung $x = \log_b a$. Wir nennen x den* Logarithmus *von a zur Basis b.*

[55] Nach dem Schweizer Mathematiker Leonhard Euler, 1707–1783

Seien weiterhin $c, d \in \mathbb{R}_{>0}$ mit $d \neq 1$ und $y \in \mathbb{R}$. Da Exponentialfunktion und Logarithmus voneinander die Umkehrfunktionen sind, gilt:

$$b^{\log_b y} = y = \log_b(b^y).$$

Damit ergeben sich direkt aus den Potenzgesetzen folgende Gesetze für Logarithmen:

$$\begin{aligned} \log_b(a \cdot c) &= \log_b a + \log_b c, \\ \log_b\left(\frac{a}{c}\right) &= \log_b a - \log_b c, \\ \log_b a &= \frac{\log_d a}{\log_d b}, \\ \log_b(a^y) &= y \cdot \log_b a. \end{aligned} \tag{*}$$

Ist aus dem Kontext klar, welche Basis gemeint ist, schreiben wir oft statt $\log_b x$ nur $\log x$. In der Informatik wird als Basis häufig 2 verwandt, daher lassen wir sie oft weg. In der Mathematik und Physik ist der *natürliche Logarithmus* (engl. *natural logarithm*) zur Basis e sehr wichtig und definiert $\ln x := \log_e x$. Logarithmen eines Arguments x zu verschiedenen Basen unterscheiden sich nur um einen konstanten Faktor voneinander, wie man in (*) sieht.

Wenn wir im Argument der log-Funktion eine Variable mit nur *einem* Zeichen schreiben, schreiben wir oft nur $\log x$ (ohne Klammern). Um Unklarheiten zu vermeiden, sollte man aber klammern, wenn mehrere Zeichen dort stehen: denn bedeutet $\log a \cdot b$ nun $\log(a) \cdot b$ oder $\log(a \cdot b)$? Ebenso: was bedeutet $\log x^2$? Entweder $(\log x)^2$ oder $\log(x^2)$?

Es gibt die Schreibweise $\log^2 x$, die bedeutet $(\log x)^2$.

Anschaulich kann man sagen, dass der Logarithmus einer ganzen Zahl > 1 etwa ihrer Länge in Ziffern entspricht. Genau gilt: Sei $n \in \mathbb{N}$ zur Basis $b \in \mathbb{N}_{\geq 2}$ geschrieben, dann braucht es $\lfloor \log_b n \rfloor + 1$ Ziffern, um n zur Basis b auszuschreiben. [56] Damit ist $\log x$ eine der langsamst-wachsenden einfachen Funktionen. Das wird in Kapitel 11 noch wichtig werden.

Eine ganz praktische Anwendung des Logarithmus kennen wir schon, nämlich die Einheitenpräfixe. Die wichtigsten wollen wir in Folge einmal auflisten. In der Informatik werden oft die binären Werte benutzt, obwohl sie nicht ganz mit den dezimalen übereinstimmen, z.B. ist k $= 10^3 = 1000 \neq 1024 = 2^{10}$. [57] Für Kilo und Mega sind die Abweichungen zwischen den dezimalen und binären Präfixen noch gering, aber pro Schritt werden sie um den Faktor $2^{10}/10^3 = 1.024 = 2.4\%$ größer. Um die binären Präfixe klar zu kennzeichnen, wird den Dezimalpräfixen manchmal [58] ein "i" nachgestellt, z.B. 1Ki $= 1024$.

[56] Die $\lfloor x \rfloor$ Notation wurde im Abschnitt 9.2 eingeführt.
[57] Beachten Sie, dass der (dezimale) Präfix für Kilo der einzige Präfix > 1 ist, der mit einem kleinen Buchstaben abgekürzt wird.
[58] Speziell unter Linux, selten unter Windows. GNU style!

Präfix	Name	(dezimaler) Wert	binärer Wert
E	Exa	10^{18}	2^{60}
P	Peta	10^{15}	2^{50}
T	Tera	10^{12}	2^{40}
G	Giga	10^{9}	2^{30}
M	Mega	10^{6}	2^{20}
k	Kilo	10^{3}	2^{10}
m	Milli	10^{-3}	
μ	Mikro	10^{-6}	
n	Nano	10^{-9}	
p	Pico	10^{-12}	

11. O-Notation

> If you find that you're spending almost all your time on theory,
> start turning some attention to practical things; it will improve your theories.
> If you find that you're spending almost all your time on practice,
> start turning some attention to theoretical things; it will improve your practice.
>
> — Donald E. Knuth [59]

Bevor wir eine formale Definition der Groß-O-Notation (engl. *big O notation*) (auch Landau-Notation genannt) geben, wollen wir die Idee motivieren.[60]

Wenn ein Algorithmus entwickelt oder implementiert wird, stellt sich evtl. die Frage nach der Laufzeit oder dem Speicherbedarf (häufiger ist die Frage nach der Laufzeit, daher betrachten wir fortan nur diese; die Theorie ist unabhängig davon, was man modelliert). Natürlich kann die Laufzeit in Sekunden gemessen werden, aber dieser Wert ist abhängig von vielen Details (Rechnergeschwindigkeit, Compiler, Caches, ...). Ebenso kann die Rechenzeit theoretisch angegeben werden, also auf einer hypothetischen Architektur wie z.B. Knuths `MIX` oder Schönhages TP. Das ist allerdings sehr aufwändig und teilweise auch nicht exakt möglich. Bei beiden Arten der Beschreibung erhält man auch nur Messpunkte, die sich evtl. nur schwer extrapolieren lassen.

Das Problem wird weiterhin dadurch erschwert, dass die Laufzeit häufig von der Eingabe abhängt, also variiert, selbst wenn die Länge gleich ist. So ist evtl. die Sortierung einer aufsteigenden Liste von Zahlen in einer anderen Größenordnung von Zeit möglich, als wenn die Liste durchgewürfelt ist. Wir sprechen hier von dem besten Fall (engl. *best case*), dem durchschnittlichen Fall (engl. *average case*), sowie dem schlechtesten Fall (engl. *worst case*). Wenn nichts anderes angegeben ist, ist normalerweise der schlechteste Fall gemeint.

Ein erstes Maß für den *Aufwand* an Zeit, den ein Algorithmus benötigt, kann der Zusammenhang zwischen der Länge der Eingabe und der Rechenzeit des Algorithmus darstellen. Also: Wenn sich die Länge der Eingabe verdoppelt, wie entwickelt sich die Laufzeit? Bleibt sie gleich? Verdoppelt sie sich? Oder vielleicht vervierfacht sie sich sogar? Wir sagen auch: in welcher *Größenordnung* (engl. *order*) liegt die Rechenzeit?

Diese Betrachtung machen wir für den asymptotischen Fall, d.h., für den Fall, dass die Länge der Eingabe $n \to \infty$ geht. Damit kann man die Größenordnung des Aufwandes als mathematische Funktion f ausdrücken, die als Parameter die Länge n der Eingabe nimmt. Dabei werden konstante Faktoren weggelassen und nur der asymptotisch stärkste Term geschrieben. Alle anderen Terme werden von diesem bei $n \to \infty$ dominiert und können einfach durch eine größere Konstante ausgedrückt werden. Wir sagen dann die Laufzeit "liegt in" $O(f(n))$.

Zur asymptotischen Laufzeit betrachten wir zuerst Abbildung 13. Es scheint so, als sei der Aufwand $f(n)$ grundsätzlich kleiner als $g(n)$. Für die dargestellte Länge n der Eingabe stimmt das auch. Wenn n aber weiter steigt, siehe Abbildung 14, sehen wir, dass es sich nur um ein kurzes "Tal" gehandelt hat. Die beiden Funktionen sind $f(n) = n^3 + 20n^2$ und $g(n) = 100n^2$ und man sieht leicht, dass für $n \to \infty$ die Funktion $f(n)$ größer wird als $g(n)$.

[59] Amerikanischer Mathematiker und Informatiker, geb. 1938, Schutzpatron aller Programmierer.
[60] Dieses Kapitel orientiert sich an [KW05, Kap. 10.4].

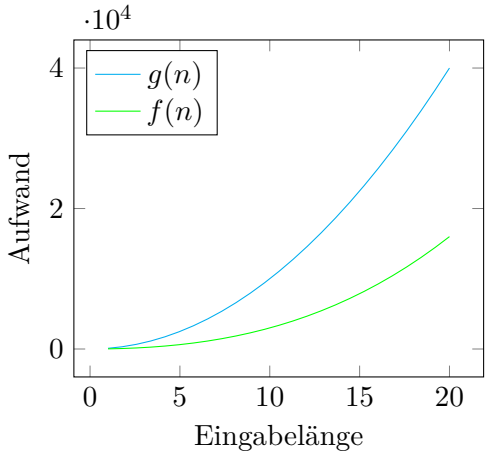


Abbildung 13: Aufwand für kurze Eingaben

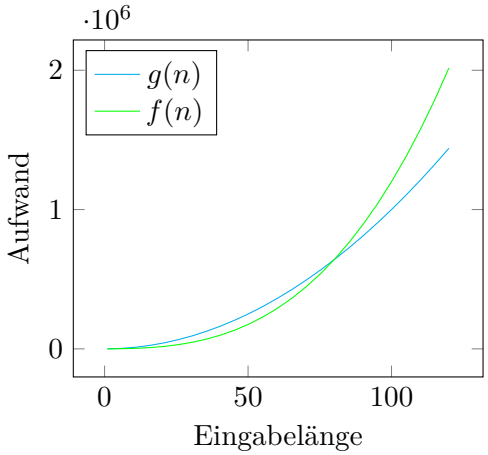


Abbildung 14: Aufwand längerer Eingaben

Definition 11.1 ("Groß-O")**:** *Sei* $f : \mathbb{N} \to \mathbb{R}_{>0}$*, dann ist die* Größenordnung *von* f *eine Menge von Funktionen und definiert als*

$$O(f(n)) = \{t : \mathbb{N} \to \mathbb{R}_{\geq 0} \mid \exists c \in \mathbb{R}_{>0}, n_0 \in \mathbb{N}_0 : \forall n \geq n_0 : t(n) \leq c \cdot f(n)\}.$$

Beispiel 11.1: Sei M eine Menge mit totaler Ordnung "$\geq$" und sei das maximalen Element

$$x = \max M := a \in M \mid \forall b \in M : a \geq b.$$

Der Aufwand zum Finden von $\max M$ liegt in $O(n)$, da man jedes Element einmal vergleichen muss. Einen solchen Aufwand nennt man auch *linear*. ◁

Beispiel 11.2: Seien Zahlen a, $b \in N_0$ gegeben in Dezimaldarstellung, also $a = \sum_{i=0}^{n} a_i$ und $b = \sum_{j=0}^{m} b_j$ mit $0 \leq a_i, b_j \leq 9$. Die a_i und b_j sind also Dezimalziffern.

Die Schulmethode zum Multiplizieren von a und b multipliziert jede Ziffer a_i mit jeder Ziffer b_j und addiert dann die Produkte stellenwertgerecht auf. [61] Dieser Algorithmus benötigt $nm + n + m$ Schritte. Gehen wir im worst case davon aus, dass $n = m$ ist, dann benötigt der Algorithmus $n^2 + 2n$ Schritte und liegt damit in $O(n^2)$. Diesen Aufwand nennt man *quadratisch*. ◁

Beispiel 11.3: Das Gaußsche Eliminationsverfahren zum Lösen von einem Gleichungssystem mit n Unbekannten hat eine Laufzeit von $O(n^3)$. Das nennt sich auch *kubische* Laufzeit. ◁

Beispiel 11.4: Die Suche in einer geordneten Liste der Länge n benötigt maximal $\lceil \log_2 n \rceil$ Vergleiche, wenn man die Liste sukzessive halbiert und das gesuchte Element mit dem Mittelelement vergleicht. Ist das Mittelelement größer als das gesuchte, sucht man im "linken" Teil der Liste weiter, ist es größer, dann im "rechten", sonst hat man es schon gefunden. Also hat diese Suche eine Laufzeit von $O(\log n)$ und wir nennen sie. *logarithmische* Laufzeit. ◁

[61] Es gibt einige weitere Verfahren zum Multiplizieren, z.B. Karatsuba (Laufzeit von $O(n^{1.585})$) oder Schönhage-Strassen (mit Aufwand $O(n \cdot \log(n) \cdot \log(\log(n)))$).

A. Symbole

$\neg A$	Negation, `NOT`, “Nicht”
$A \wedge B$	Konjunktion, `AND`, “Und”
$A \vee B$	Disjunktion, `OR`, “(inklusives) Oder”
$A \oplus B$	Exklusives Oder, `XOR`
$A \Rightarrow B$	Implikation: Aus A folgt B
$A \Leftrightarrow B$	Äquivalenz: A “genau dann” oder “dann und nur dann”, wenn B
$\forall x : A(x)$	Allquantor: für alle x gilt: $A(x)$
$\exists x : A(x)$	Existenzquantor: es gibt mindestens ein x, für das gilt: $A(x)$
$\exists! x : A(x)$	Einzigkeitsquantor: es gibt genau ein x, für das gilt: $A(x)$
$a \in A$	a ist Element oder enthalten in der Menge A
$\emptyset$	Die leere Menge $\{\}$
$A \cup B$	Vereinigung der Mengen A und B: $\{x \mid x \in A \vee x \in B\}$
$A \cap B$	Der Schnitt der Mengen A und B: $\{x \mid x \in A \wedge x \in B\}$
$A \setminus B$	Die Differenz der Mengen A und B: $\{x \mid x \in A \wedge x \notin B\}$
$\overline{A}$	Das Komplement von A bezüglich einer Obermenge B: $B\ A$
$A \subseteq B$	A ist Teilmenge von B: $\forall x : x \in A \Rightarrow x \in B$
$A \subset B$, $A \subsetneq B$	A ist echte Teilmenge von B: $A \subseteq B \wedge A \neq B$
$\mathbb{N}$	Menge der natürlichen Zahlen: $\{1, 2, 3, \ldots\}$
$\mathbb{N}_0$	Menge der natürlichen Zahlen mit 0: $\{0, 1, 2, \ldots\}$
$\mathbb{Z}$	Ring der ganzen Zahlen: $\{0, \pm 1, \pm 2, \ldots\}$
$\mathbb{Q}$	Körper der rationalen Zahlen: $\{a/b \mid a \in \mathbb{Z}, b \in \mathbb{N}\}$
$\mathbb{R}$	Körper der reellen Zahlen
$\mathbb{R}_{>0}$	$\{x \in \mathbb{R} \mid x > 0\}$
$\mathbb{C}$	Körper der komplexen Zahlen: $\{a + bi \mid a, b \in \mathbb{R}\}$ mit $i^2 = -1$
$[a, b]$	Abgeschlossenes Intervall: $\{x \in \mathbb{R} \mid a \leq x \leq b\}$, mit a, $b \in \mathbb{R}$
(a, b)	Offenes Intervall: $\{x \in \mathbb{R} \mid a < x < b\}$, mit a, $b \in \mathbb{R}$
$\lfloor a \rfloor$	Abrundungsfunktion: $\max\{x \in \mathbb{Z} \mid x \leq a\}$, mit $a \in \mathbb{R}$
$\lceil a \rceil$	Aufrundungsfunktion: $\min\{x \in \mathbb{Z} \mid x \geq a\}$, mit $a \in \mathbb{R}$
$\mathrm{dom}(R)$	Definitionsbereich der Relation R: $\{x \mid \exists y(x, y) \in R\}$
$\mathrm{ran}(R)$	Wertebereich der Relation R: $\{y \mid \exists x(x, y) \in R\}$
$f : A \to B$	Funktion f mit Definitionsbereich A und Zeilbereich B
$x \mapsto f(x)$	x wird abgebildet auf $f(x)$
$a \circ b$	Komposition von Funktionen oder allgemeine Verknüpfung
a^{-1}	Multiplikatives Inverses zu a
$\mathbb{R}[x]$	Polynomring in x über $\mathbb{R}$, d.h. die Menge der Funktionen, die darstellbar sind als $\sum_{i \in \mathbb{N}_0} a_i x^i$ mit $a_i \in \mathbb{R}$
$a \bmod d$	Modulo: Rest r der Division a/d mit a, n, d, $r \in \mathbb{Z}$, $d \neq 0$: $a = n \cdot d + r$, $0 \leq r < \lvert d \rvert$
$a \mid b$	$a \in \mathbb{Z} \setminus \{0\}$ teilt $b \in \mathbb{Z}$, also $a \bmod b = 0$
$a \nmid b$	$a \in \mathbb{Z} \setminus \{0\}$ teilt $b \in \mathbb{Z}$ nicht, also $a \bmod b \neq 0$
$\log_b(x)$	Logarithmus von x zur Basis b
e	Eulersche Zahl $e = \sum_{i=0}^{\infty} 1/i! = 2.718281828459\ldots$
$\ln x$	Natürlicher Logarithmus von x zur Basis e: $\log_e x$
$O(f(n))$	Asymptotischer Aufwand: $\{t : \mathbb{N} \to \mathbb{R}_{\geq 0} \mid \exists c \in \mathbb{R}_{>0}, n_0 \in \mathbb{N} : \forall n \geq n_0 : t(n) \leq c \cdot f(n)\}$

B. Griechisches Alphabet

Alpha	α	A
Beta	β	B
Gamma	γ	Γ
Delta	δ	Δ
Epsilon	ε, ϵ	E
Zeta	ζ	Z
Eta	η	H
Theta	θ, ϑ	Θ
Iota	ι	I
Kappa	κ, $\varkappa$	K
Lambda	λ	Λ
My	μ	M
Ny	ν	N
Xi	ξ	Ξ
Omikron	o	O
Pi	π	Π
Rho	ρ, ϱ	P
Sigma	σ	Σ
Tau	τ	T
Ypsilon	υ	Υ
Phi	ϕ, φ	Φ
Chi	χ	X
Psi	ψ	Ψ
Omega	ω	Ω

Es hilft beim Lesen und Verstehen von wissenschaftlichen Texten, die griechischen Buchstaben aussprechen und schreiben zu können.

C. Rechenregeln

Brüche

$$\frac{a}{b} \cdot \frac{c}{d} = \frac{a \cdot c}{b \cdot d}$$
$$\frac{a}{b} : \frac{c}{d} = \frac{a}{b} \cdot \frac{d}{c} = \frac{ad}{cb}$$

Potenzgesetze[2]

(Siehe auch Anhang D.)

$$a^0 = 1, \quad a^1 = a, \quad 0^0 \text{ undefiniert}$$
$$a^m \cdot a^n = a^{m+n}$$
$$\frac{a^m}{a^n} = a^{m-n}$$
$$\frac{1}{a^m} = a^{-m} \Rightarrow \frac{1}{a} = a^{-1}$$
$$\left(\frac{1}{a}\right)^m = \frac{1^m}{a^m} = \frac{1}{a^m}$$
$$a^{\frac{m}{n}} = \sqrt[n]{a^m} = (\sqrt[n]{a})^m$$
$$a^{m \cdot n} = (a^m)^n$$
$$a^m \cdot b^m = (a \cdot b)^m$$
$$\left(\frac{a}{b}\right)^m = \frac{a^m}{b^m}$$

$\sqrt{\text{Wurzelgesetze}}$

$$\sqrt[n]{a} = a^{1/n} \Rightarrow \sqrt{a} = a^{1/2}$$
$$\sqrt[n]{a} \cdot \sqrt[n]{b} = \sqrt[n]{a \cdot b}$$
$$\frac{\sqrt[n]{a}}{\sqrt[n]{b}} = \sqrt[n]{\frac{a}{b}}$$
$$\sqrt[n]{x} = x^{1/n} \Rightarrow \sqrt[2]{x} = x^{1/2}$$
$$\sqrt[k]{\sqrt[n]{a}} = \sqrt[k \cdot n]{a}$$
$$a^{m/n} = \sqrt[n]{a^m} = (\sqrt[n]{a})^m$$
$$a^{-m/n} = \frac{1}{a^{m/n}}$$

Binome

$$(a+b)^2 = a^2 + 2ab + b^2$$
$$(a-b)^2 = a^2 - 2ab + b^2$$
$$(a+b) \cdot (a-b) = a^2 - b^2$$
$$\binom{n}{k} = \frac{n!}{(n-k)! \cdot k!}$$
$$\binom{n}{0} = \binom{n}{n} = 1$$
$$\binom{n}{1} = \binom{n}{n-1} = n$$
$$\binom{n}{k} = \binom{n}{n-k}$$

Exp, Log

$$e = 2.718281828459\ldots$$
$$e^x = \exp(x) = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$
$$a^b = \exp(b \cdot \log(a))$$
$$\log_b(1) = 0$$
$$\log_n(n) = 1$$
$$\log_b(x \cdot y) = \log_b(x) + \log_b(y)$$
$$\log_b\left(\frac{x}{y}\right) = \log_b(x) - \log_b(y)$$
$$\log_b(x+y) = \log_b(x) + \log_b(1 + \frac{y}{x})$$
$$\log_b(x^r) = r \cdot \log_b(x)$$
$$\log_b\left(\frac{1}{x}\right) = -\log_b(x)$$
$$\log_b(\sqrt[n]{x}) = \log_b(x^{1/n}) = \frac{1}{n}\log_b(x)$$
$$\log_b(x) = \frac{\log_a(x)}{\log_a(b)}$$
$$a^{\log_n(b)} = b^{\log_n(a)}$$
$$\log_b(b^a) = a = b^{\log_b(a)}$$

D. Potenzgesetze, vollständig

Der folgende Text stammt aus Wikipedia:

Um die nachfolgende Tabelle nicht zu überladen, betrachten wir nur Potenzen mit reellen Basen, die ungleich 0 sind. Betrachtet man aber eines der unten aufgeführten Gesetze mit nur positiven Exponenten, dann ist es auch für Potenzen zur Basis 0 gültig. Wenn von rationalen Zahlen mit geraden oder ungeraden Nennern gesprochen wird, dann sind stets die Nenner ihrer gekürzten Bruchdarstellungen gemeint.

$a^0 = 1$	für alle $a \neq 0$
$a^{-r} = \frac{1}{a^r}$	für beliebige reelle r, falls $a > 0$ ist; für beliebige rationale r mit ungeraden Nennern, falls $a < 0$ ist.
$a^{\frac{m}{n}} = \sqrt[n]{a^m} = (\sqrt[n]{a})^m$	für beliebige natürliche n und ganze m, falls $a > 0$ ist; für beliebige natürliche ungerade n und ganze m, falls $a < 0$ ist.
$a^{r+s} = a^r \cdot a^s$	für beliebige reelle r, s, falls $a > 0$ ist; für beliebige rationale r, s mit ungeraden Nennern, falls $a < 0$ ist.
$a^{r-s} = \frac{a^r}{a^s}$	für beliebige reelle r, s, falls $a > 0$ ist; für beliebige rationale r, s mit ungeraden Nennern, falls $a < 0$ ist.
$(a \cdot b)^r = a^r \cdot b^r$	für beliebige natürliche r, und für ganze r, wenn $a \cdot b \neq 0$; für beliebige reelle r, falls $a > 0$, $b > 0$ sind; für beliebige rationale r mit ungeraden Nennern, falls mindestens eine der Zahlen a, b negativ ist.
$(\frac{a}{b})^r = \frac{a^r}{b^r}$	für beliebige ganze r mit $r \geq 0$ und $b \neq 0$ oder $r \leq 0$ und $a \neq 0$; für beliebige reelle r, falls $a > 0$, $b > 0$ sind; für beliebige rationale r mit ungeraden Nennern, falls mindestens eine der Zahlen a, b negativ ist.
$(a^r)^s = a^{r \cdot s}$	für beliebige ganze r, s, falls $a \neq 0$ ist; für beliebige reelle r, s, falls $a > 0$ ist; für beliebige rationale r, s, mit ungeraden Nennern, falls $a < 0$ ist.
$(a^r)^s = -a^{r \cdot s}$	für $a < 0$ und beliebige rationale r, s, falls r und $r \cdot s$ ungerade Nenner haben und $r \cdot s$ einen ungeraden Zähler hat.

Ist mindestens einer der Exponenten r, s irrational oder sind beide rational, aber hat mindestens eine der Zahlen r oder $r \cdot s$ einen geraden Nenner, dann ist einer der Ausdrücke $(a^r)^s$ oder $a^{r \cdot s}$ für $a < 0$ undefiniert. Ansonsten sind beide definiert und stimmen entweder überein oder unterscheiden sich nur um ihr Vorzeichen. Für beliebige r, s, falls $a > 0$ ist, und für ganze r, s, falls $a \neq 0$ ist, stimmen sie immer überein. Für $a < 0$ und nicht ganzzahlige, aber rationale r, s sind diese beiden Fälle möglich. Welcher Fall eintritt, hängt von der Anzahl der Zweien in der Primzahlzerlegung des Zählers von r und des Nenners von s ab. Um das richtige Vorzeichen auf der rechten Seite der Formel $(a^r)^s = \pm a^{r \cdot s}$ zu erkennen, ist es hinreichend, in diese Formel $a = -1$ einzusetzen. Das Vorzeichen, mit dem sie dann bei $a = -1$ gültig ist, bleibt richtig für alle $a < 0$ und gegebenem r, s. Gilt $(a^r)^s = -a^{r \cdot s}$ für $a < 0$, dann gilt $(a^r)^s = |a|^{r \cdot s}$ für alle $a \neq 0$ (und auch für $a = 0$, falls alle Exponenten positiv sind).

Zum Beispiel gilt $((-1)^2)^{\frac{1}{2}} = 1$ und $(-1)^{2 \cdot \frac{1}{2}} = -1$. Darum ist $\sqrt{a^2} = (a^2)^{\frac{1}{2}} = -a^{2 \cdot \frac{1}{2}} = -a$ für alle $a < 0$ und somit $\sqrt{a^2} = |a|$ für alle reellen a gültig.

E. Programmieraufgaben

Falls Sie auf der Suche nach Übungsaufgaben zum Programmieren sind, dann finden Sie hier einige Anregungen. Die Liste ist grob sortiert nach steigendem Schwierigkeitsgrad.

- Umrechnung EUR in USD und zurück
- Body-Mass-Index ausrechnen und Bewertung ausgeben
- Primfaktorzerlegung einer ganzen Zahl
- Primzahlen finden mit dem Sieb des des Eratosthenes
- Verifizierung eines Datums (Schaltjahre, etc.)
- Pseudo-Zufallszahlen generieren mit der LCM Methode
- ggT berechnen mit dem (erweiterten) Euklidischen Algorithmus
- Binäre Suche in einer schon sortieren Liste
- Zahl in römischen Ziffern ausdrücken und zurück
- Sortierter binärer Baum (für beliebige Datentypen)
- Datum umrechnen in Sekunden seit 1.1.1970 und zurück (mit Schaltjahren!)
- Prüfziffernberechnung oder Überprüfung einer IBAN
- Eigene Berechnung der Quadratwurzel durch Intervallschachtelung
- Sortierfunktion schreiben wie Bubble Sort oder Merge Sort
- Gedichte generieren (das können Sie beliebig aufwändig machen)
- Deterministischen endlichen Automaten implementieren
- Taschenrechner mit Punkt-vor-Strichrechnung, der “10+2*5.5” rechnen kann
- Klasse zur komplexen Arithmetik (+, −, ·, :)
- Addition und Multiplikation von langen Zahlen
- Tic-Tac-Toe Spiel mit optimaler Strategie
- Karatsuba-Multiplikation von langen Zahlen
- Klasse zur Matrix-Arithmetik (+, −, ·, Inverses)

Viele weitere Ideen finden Sie auch hier. Wenn Sie es gerne mathematisch haben, dann gibt es bei Projekt Euler massenhaft Aufgaben. Die Hochschule Karlsruhe hat auch eine schöne Liste mit Aufgaben samt Tipps und Lösungen.

Literatur

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[BS89] I. N. Bronstein and K. A. Semendjajew. *Taschenbuch der Mathematik*. Harri Deutsch, 24th edition, 1989.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[Fil07] A. Filler. Einführung in die Gruppentheorie, 2007. http://www.mathematik.hu.2berlin.de/~filler/lv_ph/algebra2/Skript.2Algebra2.pdf.

[Fis11] Gerd Fischer. *Lehrbuch der Algebra*. Vieweg & Teubner, 2nd edition, 2011.

[Fis14] Gerd Fischer. *Lineare Algebra*. Springer, 18 edition, 2014.

[Gat10] Andreas Gathmann. Algebraische Strukturen, 2010. http://www.mathematik.uni.2kl.de/agag/mitglieder/professoren/gathmann/notes/agstr/.

[GKP94] Roland L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete mathematics*. Addison-Wesley, 2nd edition, 1994.

[Gri12] Daniel Grieser. *Mathematisches Problemlösen und Beweisen*. Springer Vieweg, 2012.

[Hoe14] Georg Hoever. *Vorkurs Mathematik*. Springer, 2014.

[JL01] Gordon James and Martin Liebeck. *Representations and Characters of Groups*. Cambridge University Press, 2nd edition, 2001.

[Jun10] Markus Junker. Einführung in Sprache und Grundbegriffe der Mathematik, December 2010. http://home.mathematik.uni.2freiburg.de/junker/skripte/Grundlagen.2WS1011.pdf.

[KEM80] *Kleine Enzyklopädie Mathematik*. Verlag Harri Deutsch, 2nd edition, 1980.

[Knu97a] Donald E. Knuth. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley, 3rd edition, 1997.

[Knu97b] Donald E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley, 3rd edition, 1997.

[Kon98] Konfuzius. *Gespräche*. Reclam, 1998.

[KW05] Wolfgang Küchlin and Andreas Weber. *Einführung in die Informatik: Objektorientiert mit Java*. Springer, 3rd edition, 2005.

[LL04] Eric Lehman and Tom Leighton. Mathematics for Computer Science, 2004. https://www.cs.princeton.edu/courses/archive/spring10/cos433/mathcs.pdf.

[Rö13] Heiko Röglin. Skript zur Vorlesung Logik und diskrete Strukturen, March 2013. http://www.roeglin.org/teaching/WS2012/LuDS/LuDS.pdf.

[Rud09] Walter Rudin. *Analysis*. Oldenbourg, 4th edition, 2009.

[Sch92] Uwe Schöning. *Theoretische Informatik kurz gefaßt*. B.I.-Wissenschaftsverlag, 1992.

[Sed92] Robert Sedgewick. *Algorithms in C++*. Addison-Wesley, 1992.

[vdW71] B. L. van der Waerden. *Algebra I*. Springer, 1971.