Vorkurs

Formale Methoden der Informatik

David Kübel, Barbara Schwarzwald

basierend auf Material von Christoph Lüders

Institut für Informatik

Wintersemester 2020/21

28.09. bis 09.10.2020

Version 7.0

Inhaltsverzeichnis

1. Intro **1**
- 1.1. Danksagung . . . 1
- 1.2. Organisatorisches . . . 1
- 1.3. Raison d'être . . . 2
- 1.4. Selbsthilfe . . . 2
- 1.5. Literatur . . . 4

2. Mathematische Sprache **5**
- 2.1. Term, Gleichung, Ungleichung . . . 6
- 2.2. Rechengesetze . . . 6
- 2.3. Konventionen . . . 7
- 2.4. Summen- und Produktschreibweise . . . 7

3. Logik **9**
- 3.1. Aussagenlogik . . . 9
- 3.2. Operationen auf Aussagen . . . 10
- 3.3. Gesetze für Aussagen . . . 13
- 3.4. Prädikatenlogik . . . 15
- 3.5. Quantoren . . . 15
- 3.6. Quantorenregeln . . . 17

4. Mengen **19**
- 4.1. Beschreibung . . . 19
- 4.2. Vereinigung und Schnitt . . . 21
- 4.3. Teilmengenbeziehungen . . . 22
- 4.4. Weitere Operationen auf Mengen . . . 25

5. Datentypen **27**
- 5.1. Zahlensysteme . . . 27
- 5.2. Skalare Datentypen . . . 29
- 5.3. Mengen in C/C++ und Python . . . 31
- 5.4. Boolesche Operationen in Programmiersprachen . . . 32

6. Beweistechniken **34**
- 6.1. Umformen von Gleichungen . . . 34
- 6.2. Direkte Beweise . . . 35
- 6.3. Fallunterscheidung . . . 36
- 6.4. Indirekte Beweise . . . 38
- 6.5. Ringschluss . . . 39
- 6.6. Vollständige Induktion . . . 40

A. Lösungen zu den Aufgaben **44**

B. Symbole **47**

C. Griechisches Alphabet **48**

D. Rechenregeln **49**

E. Potenz- und Wurzelgesetze **50**

F. Programmieraufgaben 51

G. Versionsgeschichte 52

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 großartige Arbeit in den letzten 6 Jahren, die uns erlaubt, bei diesem Vorkurs auf umfangreiches Material und Erfahrung zurückzugreifen, obwohl wir ihn zum ersten Mal halten, danken wir besonders besonders herzlich Christoph Lüders.

Auch möchte ich mich jetzt schon herzlich bei allen diesjährigen Tutoren bedanken: Lena Berster, Jonas Cremer, Bettina Esser, Julius Gummersbach, Matthias Neidhardt, Johanna Ockenfels, Benedikt Stratmann und Vincent Wieland.

Teile dieses Vorkurses orientieren sich an dem Skript zur Vorlesung "Logik und diskrete Strukturen" von Heiko Röglin [Rö17]. Ebenso folgen einige Abschnitte Teilen aus "Einführung in die Informatik" von Wolfgang Küchlin und Andreas Weber [KW05]. Herzlichen Dank für die Inspiration und Vorlage.

Der Vorkurs basiert in Teilen auf einem Skript und Übungszetteln von Leif Thiemann und Christopher Voss. Sarah Sturm hat die Übungen weiter ergänzt und durchgesehen. Auch an diese meinen herzlichen Dank.

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

Bonn, im September 2020
B. S. und D. K.

1.2. Organisatorisches

Der Vorkurs findet statt von Montag, dem 28.09., bis Freitag, dem 09.10., jeweils von 10h–12h.

Aufgrund der Covid-19-Sicherheitsmaßnahmen findet der Vorkurs erstmals komplett online statt. In der ersten Woche (vom 28.09. bis 02.10.) wird die Vorlesung von Barbara Schwarzwald gehalten und Sie können unter diesem Zoom-Konferenz-Zugangslink teilnehmen. In der zweiten Woche (vom 05.10. bis 09.10.) wird die Vorlesung von David Kübel gehalten.

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.

Die Übungen sind unterteilt in 8 Gruppen und finden von 13h–15h statt. Manche der Gruppen werden über Zoom, andere über die Plattform BigBlueButton gehalten. Zur Teilnahme treten Sie ab dem 28.09. einer der Übungsgruppen im precampus-Kurs bei. Dort finden Sie dann die entsprechenden Zugangsdaten.

[1] Sie haben trotzdem noch einen Fehler gefunden? Am Besten sagen Sie uns direkt in oder nach der Vorlesung Bescheid. Aber auch Email wird gelesen. Vielen Dank!

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

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. Weiterhin kann es am nächsten Tag bei einem neuen Thema ganz anders aussehen.

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. 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.

  Das Skript enthält auch einige Aufgaben, deren Lösungen dann in Anhang A zu finden sind.

- 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.

- Videos im Netz, z.B. von Christian Spannagel von der PH Heidelberg (auf seinem Youtube Channel).

  Weiterhin hervorragend ist 3Blue1Brown mit seinem Youtube-Kanal. Ansehen!

  Es gibt zu vielen Themen gute Videos, suchen Sie mal danach.

- 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 SageMath an, auch online als *SageMathCell*. SageMath 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!

[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.

- 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 Express für C/C++, C#, Visual Basic, Python und F# kostenlos und sehr leistungsfähig. Nicht zuletzt kann man Python für alle Betriebssysteme völlig frei laden und benutzen. Prima Sprache!

  In Anhang F 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.

- 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 vorletzter 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.

- Last but not least: schlafen Sie genug. "Was für ein lamer Tipp", werden Sie denken, aber es lohnt sich, auch akademisch! Ausgeschlafen können Sie sich Dinge besser merken, sind emotional ausgeglichener und kommen nachweislich zu besseren Noten, siehe auch "College students: getting enough sleep is vital to academic success". Wer gerne mehr dazu wissen will, dem sei das Buch "Why we sleep" von Matthew Walker [Wal17] empfohlen. It's a good and easy read.

1.5. Literatur

Teile dieses Vorkurses orientieren sich an der Vorlesung "Logik und diskrete Strukturen" von Heiko Röglin aus dem WS 2012/13 [Rö17].

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 "etwas 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? Echt kleiner oder kann es auch gleich sein?
- Ist vielleicht eine Länge gemeint? Wenn ja, in welcher Richtung gemessen?

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 B) 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 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, Schreiben, Hören, 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 C 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 kennenlernen, 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 2.1.1 (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). ⧖[3]

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

2.2. Rechengesetze

Vorab seien einige Rechengesetze wiederholt, die Sie aus $\mathbb{R}$ kennen. In Kapitel **??** 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 **??** 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 "$\cdot$", welches meist 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 meist für die Addition steht, gilt Punkt-vor-Strichrechnung. D.h., ohne Angabe von Klammern bindet das Zeichen "$\cdot$" 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 **??** 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 2.4.1:

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

⧖

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 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 2.4.2:

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

⧖

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 2.4.3:

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

⧖

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 **??**. Ähnliche Schreibweisen für andere Operationen lernen wir schon in Kapitel 3 kennen.

Aufgabe 2.4.1: Schreiben Sie mit Summenzeichen:

a) $-1 + 4 + 9 + 14 + 19$

b) $60 + 30 + 20 + 15 + 12 + 10$

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 3.1.1 (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 3.1.1: 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. Trotzdem sind es valide Aussagen.

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 es gerne multimedial mögen: Vorlesung über Aussagenlogik als Video von Christian Spannagel.
[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.

Manche Aussagen sind elementare Aussagen ("Es regnet.", "Der Boden ist nass.", "Es ist dunkel."). Alle anderen Aussagen werden aus solchen elementaren Aussagen und Operatoren zusammengesetzt ("Wenn es regnet, ist der Boden nass.").

Wahrheitswerte bzw. Aussagen können wir durch verschiedene Operationen (deren Operatoren heißen Junktoren) miteinander verknüpfen (die dadurch eine boolesche Algebra[9] bilden können). Das heißt, eine oder mehrerere mit einem Operator verknüpfte Aussagen bilden einen neue Aussage. Ob diese neue Aussage wahr oder falsch ist, hängt von den Werten der beiden verknüpften Aussagen und dem Operator ab. Da es nur endlich viele mögliche Werte für die ursprünglichen Aussagen gibt, können wir diese einfach alle auflisten. Dadurch entsteht eine Wahrheitstabelle (engl. *truth table*). Ein Operator ist durch seine Wahrheitstabelle eindeutig bestimmt.

Wenn zwei Aussagen A und B für alle möglichen Wahrheitswerte gleich sind, schreiben wir $A \equiv B$. Wir verwenden hier diese besondere Schreibweise, damit sie nicht mit den Zeichen "$=$" und "$\Leftrightarrow$" verwechselt werden kann. Eine andere verbreitete Schreibweise ist $\simeq$.

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 "$\neg$" 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 "$\wedge$". 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!**

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

Unser normalsprachliches "Oder" ist meist ein *exklusives* Oder (siehe unten). Der Satz "Trinkst du Bier *oder* Wein?" bedeutet eben meist nicht, dass man beides möchte.

Das mathematische Symbol für `OR` ist "$\vee$" und dies ist seine Wahrheitstabelle:

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

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 "$A \oplus A \equiv \mathsf{f}$". Man kann `XOR` auch aus `AND`, `OR` und `NOT` zusammensetzen:

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

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:

$$A \Rightarrow B \equiv \neg A \vee B.$$

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: $A \Leftrightarrow B \equiv (A \Rightarrow B) \wedge (A \Leftarrow B)$.

Zum Beschreiben von allen 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, $\neg$; Quantoren, $\forall$, $\exists$ (siehe Kapitel 3.5)
- Konjunktion, $\wedge$,
- Disjunktion, $\vee$,
- Implikation, $\Rightarrow$,
- Äquivalenz, $\Leftrightarrow$,
- Aussagenlogische Äquivalenz, $\equiv$.

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.

[10] Sie sind hinreichend, aber nicht notwendig! Alleine mit der `NAND` oder `NOR` Operation geht es auch. Wissen Sie, wie?

Aufgabe 3.2.1: Gegeben seien die folgenden Aussagen:
A: Es ist eiskalt.
B: Es schneit.

Drücken Sie die nachfolgenden Sätze als aussagenlogische Formeln mit Hilfe der Aussagenvariablen A und B aus.

a) Es ist eiskalt und es schneit.

b) Es ist eiskalt, aber es schneit nicht.

c) Es ist nicht eiskalt und es schneit nicht.

d) Entweder es schneit oder es ist eiskalt (oder beides).

e) Entweder es schneit oder es ist eiskalt, aber es schneit nicht, wenn es eiskalt ist.

f) Wenn es schneit, ist es eiskalt.

Aufgabe 3.2.2: Zeigen Sie mittels Wahrheitstabelle:

1. $A \oplus \mathsf{w} \equiv \neg A$
2. $A \oplus A \equiv \mathsf{f}$

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 $A \equiv B$.[11]

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 Gesetze zu Aussagen aufgeführt.

[11] siehe auch [KW05, Kapitel 16].

Konstanz:	$A \wedge \neg A \equiv \mathsf{f}$ $A \vee \neg A \equiv \mathsf{w}$
Doppelte Negation:	$\neg\neg A \equiv A$
Assoziativität:	$(A \vee B) \vee C \equiv A \vee (B \vee C)$ $(A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$
Kommutativität:	$A \vee B \equiv B \vee A$ $A \wedge B \equiv B \wedge A$
Idempotenz:	$A \vee A \equiv A$ $A \wedge A \equiv A$
Absorption:	$A \vee (A \wedge B) \equiv A$ $A \wedge (A \vee B) \equiv A$
Neutralität:	$A \vee \mathsf{f} \equiv A$ $A \wedge \mathsf{w} \equiv A$
Distributivität:	$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$ $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$
De Morgansche Gesetze: [12]	$\neg(A \vee B) \equiv \neg A \wedge \neg B$ $\neg(A \wedge B) \equiv \neg A \vee \neg B$

Hier sind noch einige weitere Gesetze und Definitionen, die man auch häufiger braucht:

Konstanz:	$A \oplus A \equiv \mathsf{f}$
Assoziativität:	$(A \oplus B) \oplus C \equiv A \oplus (B \oplus C)$
Kommutativität:	$A \oplus B \equiv B \oplus A$ $A \Leftrightarrow B \equiv B \Leftrightarrow A$
Neutralität:	$A \oplus \mathsf{f} \equiv A$
Äquivalenz: Exklusives Oder: Implikation: Prinzip der Kontraposition:	$A \Leftrightarrow B \equiv (A \Rightarrow B) \wedge (B \Rightarrow A)$ $A \oplus B \equiv (A \vee B) \wedge \neg(A \wedge B)$ $A \Rightarrow B \equiv \neg A \vee B$ $A \Rightarrow B \equiv \neg B \Rightarrow \neg A$

Aufgabe 3.3.1: `NOT`, `AND` und `OR` reichen zum Ausdrücken von allen möglichen zweistelligen logischen Operation. Aber könnte man alle auch mit weniger als drei elementaren Operation ausdrücken? Welchen?

Aufgabe 3.3.2: Zeigen Sie:

a) $(A \Rightarrow B) \vee (A \Rightarrow C) \equiv A \Rightarrow (B \vee C)$

b) $(A \Rightarrow B) \wedge (A \Rightarrow C) \equiv A \Rightarrow (B \wedge C)$

Aufgabe 3.3.3: Zeigen Sie, dass die beiden Definitionen für `XOR` aussagenlogisch äquivalent sind. Benutzen Sie dazu die Gesetze zu Aussagen.

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

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 3.4.1 (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 3.4.1: 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.

3.5. Quantoren

Jetzt definieren wir Quantoren (engl. *quantifiers*).

Definition 3.5.1 (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 *(engl.* universal quantifier*).*

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

$$\forall x : P(x) \equiv \bigwedge_i P(x_i) \equiv \underbrace{P(x_1) \wedge P(x_2) \wedge P(x_3) \wedge \dots}_{\text{alle } x_i}$$

Definition 3.5.2 (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 *(engl.* existential quantifier*).*

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

$$\exists x : P(x) \equiv \bigvee_i P(x_i) \equiv \underbrace{P(x_1) \vee P(x_2) \vee P(x_3) \vee \dots}_{\text{alle } x_i}$$

Bemerken Sie, dass beide Quantoren eine Aussage *über alle* x machen, also alle Elemente aller Mengen [13] (inklusive Zahlen, Studierender und Fahrräder). Meist möchte man spezifischere Aussagen machen und gibt die Grundmenge direkt mit an. Sei M eine Menge, dann ist

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

[13] Siehe nächstes Kapitel.

Und für den Existenzquantor:

$$\exists x \in M : P(x) \;:=\; \exists x : (x \in M \wedge P(x)).$$

Ebenso kann man direkt Bedingungen angeben: $\forall \varepsilon > 0 : \frac{1}{n} < \varepsilon$. Auch das schreibt 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.

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:

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

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

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

□

Umgekehrt ist $\exists x \in \emptyset : A(x)$ falsch, da kein x existiert, für das die Aussage wahr wäre. Der 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 beziehen sich auf so wenig wie möglich (solange und stehen damit auf der Höhe der Negation in der Rangfolge der Operatoren (siehe Seite 12). D.h., $\forall x : P(x) \Leftrightarrow Q(y)$ ist das Gleiche wie $(\forall x : P(x)) \Leftrightarrow Q(y)$. Sonst müssen Sie klammern: $\forall x : (P(x) \Leftrightarrow Q(y))$. Im Zweifel sollten Sie hier (möglicherweise überflüssige) Klammern setzen, um Ihre Intention klar zu machen.

Beispiel 3.5.1: 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 $\epsilon > 0$ existiert ein n_0 aus $\mathbb{N}$, sodass für alle $n \geq n_0$ gilt: $\frac{1}{n} < \varepsilon$."
- $\forall x \in \mathbb{N} : (x > 4 \Rightarrow x^2 < 2^x)$

⧖

Es gibt auch einen Quantor, der aussagt, dass ein Prädikat *genau einmal* wahr ist (Einzigkeitsquantor). Er ist so definiert, für ein Prädikat $B(x)$:

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

Aufgabe 3.5.1: Schreiben Sie mittels Quantoren:

a) Die Monotonie der Addition: für x, y, $z \in \mathbb{N}$ gilt: aus $x < y$ folgt $x + z < y + z$.

b) Den großen Fermatschen Satz:[14] Für kein $\mathbb{N} \ni n > 2$ existieren x, y, $z \in \mathbb{N}$ für die gilt: $x^n + y^n = z^n$.

c) Die Goldbachsche Vermutung: Jede gerade Zahl, die größer als 2 ist, ist Summe zweier Primzahlen.

Aufgabe 3.5.2: Definieren Sie einen Quantor $\exists^{=2} a : B(a)$, der ausdrückt, dass ein Prädikat für *genau zwei* Elemente gilt.

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":

$$\neg(\forall x : A(x)) \equiv \exists x : \neg A(x)$$
$$\neg(\exists x : A(x)) \equiv \forall x : \neg A(x)$$

Zur Motivation: Die Aussage $\neg(\forall x : A(x))$ bedeutet, dass $\forall x : A(x)$ falsch ist. Das bedeutet aber, dass es *mindestens ein* x geben muss, sodass $A(x)$ falsch ist. Das können wir mittels Quantor schreiben als $\exists x : \neg A(x)$.

Umgekehrt gilt für den Existenzquantor: Die Aussage $\neg(\exists x : A(x))$ heißt, dass es kein x gibt, für das $A(x)$ gilt. Also gilt für alle x die Aussage $A(x)$ *nicht*. Daher: $\forall x : \neg A(x)$.

Wenn man annimmt, dass es nur zwei verschiedene Werte für x gibt, dann ergeben sich aus den Quantorenregeln die De Morganschen Gesetze. [15]

Aufgabe 3.6.1: Negieren Sie folgende Aussagen logisch:

a) Alle Studenten, die nicht Informatik studieren, sind doof.

b) Es existiert eine gerade Zahl, die nicht die Summe zweier Primzahlen ist. Für alle geraden Zahlen gilt: Sie sind Summe zweier Primzahlen.

Aufgabe 3.6.2: Gegeben sei die Aussage "Jeder blaue Zwerg mag Schokolade". Welche der folgenden Behauptungen widerlegt die Aussage?

[14] im 17. Jahrhundert von Pierre de Fermat (1607–1665) formuliert, aber erst 1994 von Andrew Wiles (geb. 1953) bewiesen.

[15] Übersicht über Quantorenregeln im Netz: http://www.reisz.de/qa.htm und http://www.reisz.de/qa2.htm (secco).

a) Kein Zwerg mag Schokolade.

b) Kein Zwerg mag Schokolade, und es gibt einen Zwerg der blau ist.

c) Kein Zwerg ist blau.

d) Kein Zwerg ist blau oder mag Schokolade.

e) Es gibt einen Zwerg, der nicht blau ist.

f) Es gibt einen Zwerg, der blau ist und keine Schokolade mag.

g) Es gibt einen Zwerg, der nicht blau ist und Schokolade mag.

h) Es gibt einen Zwerg, der nicht blau ist oder keine Schokolade mag.

i) $\star$ Zeigen Sie a) mittels Prädikatenlogik.

4. Mengen

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

— Leopold Kronecker [16] (1893)

4.1. Beschreibung

Nach Georg Cantor [17] 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:

$$\begin{aligned} B &= \{0, 1\}, \\ F &= \{\text{rot}, \text{grün}, \text{blau}\}, \\ A &= \{\alpha, \omega\}, \\ P &= \{1, x, x^2, x^3\}.^{18} \end{aligned}$$

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:

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

[16] Deutscher Mathematiker, 1823–1891
[17] 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.[19] Beachten Sie, dass die Seite mit dem "∞"-Zeichen eine offene Grenze beschreibt, also runde Klammern zu benutzen sind.

[18] Kapitel **??** beschäftigt sich mit Mengen von Funktionen.

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

Beispiel 4.1.1:

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

⧖

Aufgabe 4.1.1: Geben Sie die Elemente der folgenden Mengen an:

a) $x \in \mathbb{N} : x < 4$

b) $x \in \mathbb{R} : x^2 = 1$

c) $x \in \mathbb{R} : \exists y \in \mathbb{Q} : xy = 1$

d) $x \in \mathbb{Z} : x < 100 \wedge \exists y \in \mathbb{Z} : y^2 = x$

4.2. Vereinigung und Schnitt

Wir können auf Mengen diverse Operationen anwenden.[20] Die Operationen können wir sehr schön mit den Methoden der Logik aus dem letzten Kapitel beschreiben und beweisen.

Oft benutzt man auch *Venn-Diagramme*, [21] um Beziehungen von Mengen darzustellen. Sie sind sehr intuitiv, ersetzen aber keinen formalen Beweis.

Definition 4.2.1 (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.

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

4.3. Teilmengenbeziehungen

Definition 4.3.1 (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:*

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

- *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: $A \cap B = \emptyset$.*

Die Teilmengenbeziehungen für die uns wohlbekannten Mengen sehen so aus:

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

Beispiel 4.3.1: Seien die Mengen $A = \{2, 3, 5, 7\}$, $B = \{1, 2, 4, 8\}$, $C = \{5, 7\}$ gegeben. Dann gilt:

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

⧖

Aufgabe 4.3.1: Sei $M = \{-1, 0, 1\}$. Welche der folgenden Aussagen sind wahr?

a) $M \subseteq \mathbb{N}$

b) $M \subseteq \mathbb{Z}$

c) $M \subseteq M$

d) $M \cap \mathbb{Z} = M$

e) $M \cup \mathbb{Z} = M$

f) $M \cup \{0,1\} = \{0,1\}$

g) $M \cap \mathbb{R} = \mathbb{R} \cap M$

h) $M \subset ((\mathbb{N} \setminus \mathbb{Z}) \cup \{0,1\})$

i) $M \subset ((\mathbb{Z} \setminus \mathbb{N}) \cup \{0,1\})$

j) $M \subset ((\mathbb{Z} \setminus \mathbb{N}) \cup \{-1,0\})$

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: $A \subseteq A$. Die leere Mengen und die Menge selbst 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:

  $$(A \cup B) \cup C = A \cup (B \cup C),$$
  $$(A \cap B) \cap C = A \cap (B \cap C).$$

- 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:

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

  Ebenso:

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

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

Sei A eine Menge. Zu zeigen:

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

Beweis: Nach Definition von Schnitt ist das:

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

Nun beweisen wir die Assoziativität des Schnitts:

Seien A, B, C Mengen. Zu zeigen:

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

Beweis: Nach Definition von Mengengleichheit heißt das:

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

Wähle ein x aus: fest, aber beliebig. Dann gilt:

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

Nach Definition ist $\wedge$ assoziativ:

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

Auf beiden Seiten steht das Gleiche, daher gilt die Äquivalenz. Da x beliebig war, gilt die Aussage für alle x und ist damit bewiesen. □

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:

$$\begin{aligned}\{x \mid x \in C \wedge (x \notin A \wedge x \notin B)\} &= \\ \{x \mid x \in C \wedge x \notin A \wedge x \notin B\} &=\end{aligned}$$

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 & Schnitt:

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

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 $A \not\subseteq B$. [22]

Aufgabe 4.3.2: Zeigen Sie die Absorptionsgesetze für Mengen:

a) $M_1 \cup (M_1 \cap M_2) = M_1$

b) $M_1 \cap (M_1 \cup M_2) = M_1$

4.4. Weitere Operationen auf Mengen

Definition 4.4.1 (Kardinalität)**:** *Die Anzahl der Elemente einer endlichen Menge M heißt* Kardinalität *(engl.* cardinality*) oder* Mächtigkeit*. Wir schreiben dafür $|M|$, manchmal findet sich auch $\#M$.*[23]

Hat eine unendliche Menge M die gleiche Mächtigkeit wie $\mathbb{N}$, d.h., existiert eine Bijektion [24] 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). Die Mengen $\mathbb{Z}$ und $\mathbb{Q}$ sind abzählbar.

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

Beispiel 4.4.1: Sei $A = \{1, 2, 3\}$, dann ist $|A| = 3$.

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

Definition 4.4.2 (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 4.4.2: 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 $|\mathcal{P}(A)| = 2^{|A|} = 8$. ⧖

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

[23] Die Kardinalität einer unendlichen Menge ist nicht so einfach anzugeben. Erstaunlicherweise gibt es hier verschiedene Mächtigkeiten.

[24] Zum Begriff der Bijektivität siehe Abschnitt **??**.

Aufgabe 4.4.1: Zeigen Sie: $A \subseteq B \Rightarrow \mathcal{P}(A) \subseteq \mathcal{P}(B)$.

Definition 4.4.3 (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.* [25] *Formal gilt:*

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

Beispiel 4.4.3: 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.

[25] 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 [26]

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 5.1.1 (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* [27]

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

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:

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

Definition 5.1.2 (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++ [28]	`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`.

Wie können Sie einfach eine Zahl nach binär konvertieren? Der folgende Python Code soll das verdeutlichen:

```
from __future__ import print_function, division

def to_bin(z):
    """
    Funktion zum Konvertieren nach Binaer.

    Eingabe: Zahl z, nicht-negativ.
    Ausgabe: Zahl zur Basis 2.
    """
    s = ""
    p = 2**(z.bit_length() - 1)   # höchstes gesetztes Bit
    while p > 0:
        if p <= z:
            z = z - p
            s += "1"
        else:
            s += "0"
        p = p // 2
    return s

if __name__ == "__main__":
    import sys
    print(to_bin(int(sys.argv[1])))
```

[28] 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) { ... }`. ☺

Der Algorithmus geht alle Zweierpotenzen durch, von hoch nach niedrig und schreibt dann jeweils eine “1” oder eine “0”.

Aufgabe 5.1.1: Berechnen Sie den Binärwert folgender Dezimalzahlen:

a) 42_{10}

b) 123_{10}

c) 408_{10}

d) 230_{10}

e) 169_{10}

f) 3141_{10}

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: $W = \{\mathsf{wahr}, \mathsf{falsch}\}$.

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. [29] Das wird nötig z.B. für Vergleiche: $\frac{1}{2} = \frac{2}{4}$.
- eine Funktion, die auf das kleinste gemeinsame Vielfache erweitert, damit man zwei Zahlen addieren kann: $\frac{1}{2} + \frac{1}{3} = \frac{5}{6}$.

[29] Siehe Abschnitt **??**

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. [30]

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:

	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`

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

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.[31] 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.

[31] Siehe Zahlensysteme, Abschnitt 5.1.

6. Beweistechniken

Math answers aren't determined by votes.

— Marilyn vos Savant[32]

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. [33] 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 nur einem Zahlenbereich ohne negative Zahlen eine Äquivalenzumformung ist (also in $\mathbb{N}$, $\mathbb{N}_0$, $\mathbb{Q}_+$ oder $\mathbb{R}_+$)! Eine hinreichendes Kriterium für Äquivalenzumformungen ist die *Injektivität* der Operation, siehe Abschnitt **??**.

Beispiel 6.1.1:

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

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

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

⧖

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

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

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

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*. [34]

Ein direkter Beweis (engl. *direct proof*) 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 dann 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 6.2.1: 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.

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

□

[35]

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

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

Beispiel 6.2.2: Als weiteres Beispiel beweisen wir jetzt die Partialsumme der geometrischen Reihe.

Sei $\mathbb{R} \ni a \neq 1$ und $n \in \mathbb{N}_0$. Dann gilt

$$\sum_{i=0}^{n} a^i = \frac{1 - a^{n+1}}{1 - a}$$

Beweis:

$$\begin{aligned}
&\Leftrightarrow && a^0 + a^1 + \ldots + a^n = \frac{1 - a^{n+1}}{1 - a} \\
&\Leftrightarrow && (1 + a + \ldots + a^n)(1 - a) = 1 - a^{n+1} \\
&\Leftrightarrow && (1 + a + \ldots + a^n) - a(1 + a + \ldots + a^n) = 1 - a^{n+1} \\
&\Leftrightarrow && 1 + a + \ldots + a^n - a - a^2 - \ldots - a^{n+1} = 1 - a^{n+1}
\end{aligned}$$

Alle Summanden außer 1 und a^{n+1} kürzen sich raus.

$$\Leftrightarrow \qquad 1 - a^{n+1} = 1 - a^{n+1} \qquad \square$$

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

Aufgabe 6.2.1: Beweisen Sie durch einen direkten Beweis:

a) Das Quadrat einer geraden Zahl ist gerade.

b) Das Quadrat einer ungeraden Zahl ist ungerade.

Aufgabe 6.2.2: Beweisen Sie den Satz des Pythagoras grafisch.

Aufgabe 6.2.3: Das Kreuzprodukt zweier Vektoren a und b ist definiert durch

$$a \times b = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \times \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{pmatrix}$$

Beweisen oder widerlegen Sie:

a) Das Kreuzprodukt ist assoziativ.

b) Das Kreuzprodukt ist kommutativ.

6.3. Fallunterscheidung

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

Beispiel 6.3.1:

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 $k, \ell \in \mathbb{N}$ mit den Eigenschaften

$$a = 2k, \quad b = 2\ell.$$

Das Produkt $a \cdot b$ ist dann

$$a \cdot b = 2k \cdot 2\ell = 2(2k\ell).$$

Somit ist das Produkt gerade.

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

$$a = 2k - 1, \quad b = 2\ell - 1.$$

Das Produkt $a \cdot b$ ist dann

$$a \cdot b = (2k - 1) \cdot (2\ell - 1) = 4k\ell - 2k - 2\ell + 1 = 2(2k\ell - k - \ell) + 1.$$

Da $2k\ell - k - \ell \in \mathbb{N}$, ist $2(2k\ell - k - \ell)$ 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 $k, \ell \in \mathbb{N}$ und wir können a und b schreiben als

$$a = 2k - 1, \quad b = 2\ell.$$

Das Produkt $a \cdot b$ ist dann

$$a \cdot b = (2k - 1) \cdot (2\ell) = 4k\ell - 2\ell = 2(2k\ell - \ell).$$

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.

Beispiel 6.3.2: Für welche $x \in \mathbb{R}$ gilt:

$$x \cdot x = x + x \ ?$$

Wir rechnen es einfach aus:

$$x^2 = 2x.$$

Damit wir im nächsten Schritt durch x dividieren können, müssen wir den $x \neq 0$ fordern. Wir betrachen den Fall $x = 0$ später getrennt.

$$x = 2$$

Der Fall $x = 0$ löst die Gleichung aber auch, wovon wir uns durch Einsetzen überzeugen können. Also sind die Lösungen 0 und 2. □

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 6.4.1: 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. $\lightning$ [36]

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 $\sqrt{2} \notin \mathbb{Q}$. □

Aufgabe 6.4.1: Zeigen Sie durch Widerspruch: Wenn n^3 durch 2 teilbar ist, dann ist auch n durch 2 teilbar.

6.5. Ringschluss

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 Ringschlusses (auch zyklisches Beweisverfahren genannt) 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

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

Beispiel 6.5.1: Seien A, B Mengen. Dann sind folgende Aussagen äquivalent:

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

Beweis: Wir zeigen, dass folgende Implikationen gelten:

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

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

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

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

  Nach Voraussetzung besteht A nur aus Elementen, die auch in B enthalten sind. Daher fügt die Vereinigung mit A der Menge B keine weiteren Elemente hinzu.

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

  Da B gleich der Vereinigung von A und B ist, sind alle Elemente aus A auch in B enthalten. Das ist die Definition von $A \subseteq B$.

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 führen und erzeugen damit eine Beweiskette für alle $n \geq n_0$.

Sie können sich das vorstellen wie den Dominoeffekt: [37] 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:

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.

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

3. Dann folgt der *Induktionsschritt* (IS) (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 6.6.1: Wir beweisen die Gaußsche Summenformel: [38]

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.

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

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

Beispiel 6.6.2: 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.

[38] 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.

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. □

Beispiel 6.6.3: Jetzt zeigen wir noch, dass für $n \in \mathbb{N}_0$ der Ausdruck $n^5 - n$ durch 5 teilbar ist.

Induktionsanfang: Für $n = 0$ steht da: $0^5 - 0 = 0 = 5 \cdot 0$. 0 ist durch 5 teilbar.

Induktionsschritt: Wir können annehmen, dass $n^5 - n$ durch 5 teilbar ist und wollen zeigen, dass $(n+1)^5 - (n+1)$ ebenfalls durch 5 teilbar ist.

$$\begin{aligned} (n+1)^5 - (n+1) = \\ n^5 + 5n^4 + 10n^3 + 10n^2 + 5n + 1 - n - 1 = \\ n^5 - n + 5n^4 + 10n^3 + 10n^2 + 5n = \\ \underbrace{n^5 - n}_{\text{nach IV}} + \underbrace{5(n^4 + 2n^3 + 2n^2 + n)}_{\text{durch 5 teilbar}} \end{aligned}$$

Da beide Summanden durch 5 teilbar sind, ist die Summe durch 5 teilbar. □

Beispiel 6.6.4: In diesem Beispiel beweisen wir eine Ungleichung.

Zu zeigen: $n^2 < 2^n$ für $n \in \mathbb{N}$, $n > 4$.

Induktionsanfang bei $n = 5$: $25 < 32$ — stimmt schon mal.

Induktionsvoraussetzungen:

$$n^2 < 2^n,$$
$$n > 4.$$

Induktionsschritt $n \to (n+1)$:

$$(n+1)^2 < 2^{n+1}$$
$$(n+1)^2 < 2 \cdot 2^n$$

Wir verschärfen die Aussage, indem wir die IV benutzen:

$$\begin{aligned}(n+1)^2 &< 2 \cdot n^2 \\ n^2 + 2n + 1 &< n^2 + n^2 \\ 2n + 1 &< n^2 \\ 1 &< n^2 - 2n \\ 2 &< n^2 - 2n + 1 \\ 2 &< (n-1)^2 \\ \sqrt{2} &< n - 1 \\ \sqrt{2} + 1 &< n\end{aligned}$$

Das gilt nach Voraussetzung. □

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.

Aufgabe 6.6.1: Zeigen Sie mittels vollständiger Induktion:

a) $1^3 + 2^3 + 3^3 + ... + (n-1)^3 + n^3 = \left(\frac{1}{2}n \cdot (n+1)\right)^2$

b) $1 + 3 + 5 + ... + (2n-1) = n^2$

c) $\prod_{i=2}^{n} \left(1 - \frac{2}{i \cdot (i+1)}\right) = \frac{1}{3} \cdot \left(1 + \frac{2}{n}\right)$

A. Lösungen zu den Aufgaben

Aufgabe 2.4.1:

a) $\sum_{i=0}^{4}(-1+5\cdot i)$

b) $\sum_{i=1}^{6}\frac{60}{i}$

Aufgabe 3.3.1:

Mit einem `NAND` oder `NOR`: daraus kann man ein `NOT` machen ($\text{NOT}(a) = \text{NAND}(a,a)$), daraus dann ein `AND` ($\text{AND}(a,b) = \text{NOT}(\text{NAND}(a,b))$ und daraus dann ein `OR` ($\text{OR}(a,b) = \text{NOT}(\text{AND}(\text{NOT}(a), \text{NOT}(b)))$). Es geht analog mit einem `NOR` als elementare Operation.

Aufgabe 3.3.2:

a)

$$\begin{aligned}(A \Rightarrow B) \vee (A \Rightarrow C) &\equiv\\ (\neg A \vee B) \vee (\neg A \vee C) &\equiv\\ \neg A \vee C \vee B &\equiv\\ \neg A \vee (C \vee B) &\equiv\\ A \Rightarrow (B \vee C)\end{aligned}$$ □

b)

$$\begin{aligned}(A \Rightarrow B) \wedge (A \Rightarrow C) &\equiv\\ (\neg A \vee B) \wedge (\neg A \vee C) &\equiv\\ \underbrace{((\neg A \vee B) \wedge \neg A)}_{\neg A} \vee ((\neg A \vee B) \wedge C) &\equiv\\ \underbrace{\neg A \vee (\neg A \wedge C)}_{\neg A} \vee (B \wedge C) &\equiv\\ \neg A \vee (B \wedge C) &\equiv\\ A \Rightarrow (B \wedge C)\end{aligned}$$ □

Aufgabe 3.2.1:

a) $A \wedge B$

b) $A \wedge \neg B$

c) $\neg A \wedge \neg B$

d) $A \vee B$

e) $(A \vee B) \wedge (A \Rightarrow \neg B)$ oder $(A \vee B) \wedge \neg(A \wedge B)$

f) $B \Rightarrow A$

Aufgabe 3.5.1:

a) $\forall x \in \mathbb{N} : \forall y \in \mathbb{N} : \forall z \in \mathbb{N} : (x < y \Rightarrow x + z < y + z)$

b) $\neg(\exists n > 2 : \exists x \in \mathbb{N} : \exists y \in \mathbb{N} : \exists z \in \mathbb{N} : x^n + y^n = z^n)$

c) Sei P die Menge der Primzahlen. $\forall x > 2 : (2 \mid x \Rightarrow \exists p \in P : \exists q \in P : x = p + q)$

Aufgabe 3.5.2: $\exists^{=2} a : B(a) \coloneqq \exists x : \exists y : (B(x) \wedge B(y) \wedge x \neq y \wedge \forall z : (B(z) \Rightarrow (z = x \vee z = y)))$

Aufgabe 3.6.1:

a) Es existiert ein Student, der nicht Informatik studiert und nicht doof ist.

b) Für alle geraden Zahlen gilt: Sie sind Summe zweier Primzahlen.

Aufgabe 3.6.2:

a) – h) Behauptungen a), b), c), d) und f) widerlegen die Aussage.

i) Wir negieren die Aussage “Jeder blaue Zwerg mag Schokolade” und zeigen, dass dies unter der Voraussetzung “Kein Zwerg mag Schokolade”, gilt.

$$\begin{aligned}&\neg(\text{Jeder blaue Zwerg mag Schokolade})\\ &\neg(\forall z : (B(z) \Rightarrow S(z)))\\ &\neg(\forall z : (\neg B(z) \vee S(z)))\\ &\exists z : \neg(\neg B(z) \vee S(z))\\ &\exists z : (B(z) \wedge \underbrace{\neg S(z)}_{\text{n. V. wahr}})\\ &\exists z : (B(z) \wedge \text{wahr})\\ &\exists z : B(z)\end{aligned}$$ □

D.h., entgegen der Intuition ist diese Aussage nur wahr, falls es blaue Zwerge überhaupt gibt.

Aufgabe 4.1.1:

a) $\{1, 2, 3\}$

b) $\{-1, 1\}$

c) $\mathbb{Q} \setminus \{0\}$

d) $\{0, 1, 4, 9, 16, 25, 36, 49, 64, 81\}$

Aufgabe 4.3.1:

a) falsch

b) wahr

c) wahr

d) wahr

e) falsch

f) falsch

g) wahr

h) falsch

i) wahr

j) falsch

Aufgabe 4.3.2:

a) "$\subseteq$": $M_1 \cup (M_1 \cap M_2) \subseteq M_1 \cup M_1 = M_1$
"$\supseteq$": $M_1 \cup (M_1 \cap M_2) \supseteq M_1$

b) $M_1 \cap (M_1 \cup M_2)$
$= (M_1 \cap M_1) \cup (M_1 \cap M_2)$
$= M_1 \cup (M_1 \cap M_2)$
$= M_1$ nach a).

Aufgabe 4.4.1:

Sei $M \in \mathcal{P}(A)$. Dann ist $M \subseteq A$ nach Definition der Potenzmenge. Nach Voraussetzung ist aber $A \subseteq B$, also auch $M \subseteq A \subseteq B$, also auch $M \subseteq B$. Das wiederum bedeutet, dass $M \in \mathcal{P}(B)$ laut Definition der Potenzmenge.

Aufgabe 5.1.1:

a) 101010_2

b) 1111011_2

c) 110011000_2

d) 11100110_2

e) 10101001_2

f) 110001000101_2

Aufgabe 6.2.1:

a) Sei n gerade $\Rightarrow \exists k \in \mathbb{Z}$, sodass $n = 2k$. $(2k)^2 = 4k^2$. Das ist offensichtlich gerade.

b) Sei n ungerade $\Rightarrow \exists k \in \mathbb{Z}$, sodass $n = 2k-1$. $(2k-1)^2 = 4k^2 - 4k + 1 = \underbrace{4(k^2-k)}_{\text{gerade}} + 1$

Aufgabe 6.2.3:

a) Falsch. Gegenbeispiel:
$a = \begin{pmatrix}1\\0\\0\end{pmatrix}, b = \begin{pmatrix}0\\1\\0\end{pmatrix}, c = \begin{pmatrix}1\\1\\1\end{pmatrix}$
$(a \times b) \times c$
$= \begin{pmatrix}0\\0\\1\end{pmatrix} \times \begin{pmatrix}1\\1\\1\end{pmatrix} = \begin{pmatrix}-1\\1\\0\end{pmatrix} \neq \begin{pmatrix}0\\1\\0\end{pmatrix} =$
$a \times (b \times c)$

b) Nein. Nachrechnen für beliebige $x, y \in \mathbb{R}^3$, $x \neq 0 \neq y$, $x \neq y$.

Aufgabe 6.4.1:

Annahme: $\exists k \in \mathbb{Z} : (2k-1)^3$ ist durch 2 teilbar.
$\Rightarrow 8k^3 - 8 + 12k^2 + 6k - 1$ ist gerade. $\lightning$

Aufgabe 6.6.1:

a) IA: $n_0 = 1 : 1^3 = 1 = \left(\frac{1\cdot(1+1)}{2}\right)^2$ ✓
IV: $\sum_{i=1}^{n} i^3 = \left(\frac{n\cdot(n+1)}{2}\right)^2$
IS:

$$\begin{aligned}\sum_{i=1}^{n+1} i^3 &= \sum_{i=1}^{n} i^3 + (n+1)^3 \\ &= \left(\frac{n\cdot(n+1)}{2}\right)^2 + (n+1)^3 \\ &= \frac{n^2\cdot(n+1)^2}{4} + (n+1)^3 \\ &= \frac{n^2\cdot(n+1)^2 + 4(n+1)(n+1)^2}{4} \\ &= \frac{(n^2+4(n+1))\cdot(n+1)^2}{4} \\ &= \frac{(n^2+4n+4)\cdot(n+1)^2}{4} \\ &= \frac{(n+2)^2\cdot(n+1)^2}{4} \\ &= \left(\frac{(n+2)\cdot(n+1)}{2}\right)^2 \qquad \square\end{aligned}$$

b) IA: $n_0 = 1 : 1 = 1^2$✓
IV: $\sum_{i=1}^{n} 2i - 1 = n^2$
IS:

$$\begin{aligned}\sum_{i=1}^{n+1} 2i - 1 &= \sum_{i=1}^{n} 2i - 1 + 2(n+1) - 1 \\ &= n^2 + 2n + 2 - 1 \\ &= n^2 + 2n + 1 \\ &= (n+1)^2 \qquad \square\end{aligned}$$

c) IA: $n_0 = 2 : (1 - \frac{2}{6}) = \frac{1}{3}$✓

IV: $\prod_{i=2}^{n} \left(1 - \frac{2}{i \cdot (i+1)}\right) = \frac{1}{3} \cdot \left(1 + \frac{2}{n}\right)$

IS:

$$
\begin{aligned}
&\prod_{i=2}^{n+1} \left(1 - \frac{2}{i \cdot (i+1)}\right) \\
&= \prod_{i=2}^{n} \left(1 - \frac{2}{i \cdot (i+1)}\right) \cdot \left(1 - \frac{2}{(n+1)(n+2)}\right) \\
&= \frac{1}{3} \cdot \left(1 + \frac{2}{n}\right) \cdot \left(1 - \frac{2}{(n+1)(n+2)}\right) \\
&= \frac{1}{3} \cdot \left(1 - \frac{2}{(n+1)(n+2)} + \frac{2}{n} - \frac{4}{n(n+1)(n+2)}\right) \\
&= \frac{1}{3} \cdot \left(1 - \frac{2n}{(n+1)(n+2)n} + \right. \\
&\qquad \left. \frac{2(n+1)(n+2)}{n(n+1)(n+2)} - \frac{4}{n(n+1)(n+2)}\right) \\
&= \frac{1}{3} \cdot \left(1 + \frac{-2n + 2(n+1)(n+2) - 4}{n(n+1)(n+2)}\right) \\
&= \frac{1}{3} \cdot \left(1 - \frac{-2n + 2n^2 + 6n + 4 - 4}{n(n+1)(n+2)}\right) \\
&= \frac{1}{3} \cdot \left(1 - \frac{2n^2 + 4n}{n(n+1)(n+2)}\right) \\
&= \frac{1}{3} \cdot \left(1 - \frac{2n + 4}{(n+1)(n+2)}\right) \\
&= \frac{1}{3} \cdot \left(1 - \frac{2}{n+1}\right) \qquad \square
\end{aligned}
$$

B. 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
$A \equiv B$	Äquivalenz logischer Aussagen, d.h. gleiche Wahrheitstabellen
$\forall x : A(x)$	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 (unechte) 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: $\{0, 1, 2, 3, \ldots\}$
$\mathbb{Z}$	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 $a \in \mathbb{R}$
$\mathrm{preim}(R)$	Urbildbereich der Relation R: $\{x \mid \exists y : (x, y) \in R\}$
$\mathrm{im}(R)$	Bildbereich der Relation R: $\{y \mid \exists x : (x, y) \in R\}$
$\mathrm{dom}(R)$	Vorbereich der Relation R
$\mathrm{codom}(R)$	Nachbereich der Relation 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: $\{t : \mathbb{N} \to \mathbb{R}_{\geq 0} \mid \exists c \in \mathbb{R}_{>0}, n_0 \in \mathbb{N} : \forall n \geq n_0 : t(n) \leq c \cdot f(n)\}$
$\square$	Quod erat demonstrandum: Ende eines Beweises
$\between$	Ende eines Beispiels

C. 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.

D. Rechenregeln

Brüche

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

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

$$e = 2.7182818284590\ldots$$
$$e^x = \exp(x) = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$
$$a^b = \exp(b \cdot \log(a))$$
$$\log_b(1) = 0$$
$$\log_n(n) = 1$$
$$\log_b(x \cdot y) = \log_b(x) + \log_b(y)$$
$$\log_b\left(\frac{x}{y}\right) = \log_b(x) - \log_b(y)$$
$$\log_b(x+y) = \log_b(x) + \log_b(1 + \frac{y}{x})$$
$$\log_b(x^r) = r \cdot \log_b(x)$$
$$\log_b\left(\frac{1}{x}\right) = -\log_b(x)$$
$$\log_b(\sqrt[n]{x}) = \log_b(x^{1/n}) = \frac{1}{n}\log_b(x)$$
$$\log_b(x) = \frac{\log_a(x)}{\log_a(b)}$$
$$a^{\log_n(b)} = b^{\log_n(a)}$$
$$\log_b(b^a) = a = b^{\log_b(a)}$$

E. Potenz- und Wurzelgesetze

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 $a \neq 0$
$a^{-r} = \frac{1}{a^r}$	für $r \in \mathbb{R}$, falls $a > 0$ ist; für $r \in \mathbb{Q}$ mit ungeradem Nenner, falls $a < 0$ ist.
$a^{\frac{m}{n}} = \sqrt[n]{a^m} = (\sqrt[n]{a})^m$	für $n \in \mathbb{N}$ und $m \in \mathbb{Z}$, falls $a > 0$ ist; für $m \in \mathbb{Z}$ und ungerade $n \in \mathbb{N}$, falls $a < 0$ ist.
$a^{r+s} = a^r \cdot a^s$ $a^{r-s} = \frac{a^r}{a^s}$	für $r, s \in \mathbb{R}$, falls $a > 0$ ist; für $r, s \in \mathbb{Q}$ mit ungeraden Nennern, falls $a < 0$ ist.
$(a \cdot b)^r = a^r \cdot b^r$	für $r \in \mathbb{N}$, und für $r \in \mathbb{Z}$, wenn $a \cdot b \neq 0$; für $r \in \mathbb{R}$, falls $a, b > 0$ sind; für $r \in \mathbb{Q}$ mit ungeraden Nennern, falls mindestens eine der Zahlen a, b negativ ist.
$(\frac{a}{b})^r = \frac{a^r}{b^r}$	für $r \in \mathbb{Z}$ mit $r \geq 0$ und $b \neq 0$ oder $r \leq 0$ und $a \neq 0$; für $r \in \mathbb{R}$, falls $a, b > 0$ sind; für $r \in \mathbb{Q}$ mit ungeraden Nennern, falls mindestens eine der Zahlen a, b negativ ist.
$(a^r)^s = a^{r \cdot s}$	für $r, s \in \mathbb{Z}$, falls $a \neq 0$ ist; für $r, s \in \mathbb{R}$, falls $a > 0$ ist; für $r, s \in \mathbb{Q}$ mit ungeraden Nennern, falls $a < 0$ ist.
$(a^r)^s = -a^{r \cdot s}$	für $a < 0$ und $r, s \in \mathbb{Q}$, 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.

F. 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 ($+$, $-$, $\cdot$, $:$)
- Addition und Multiplikation von langen Zahlen
- Tic-Tac-Toe Spiel mit optimaler Strategie
- Karatsuba-Multiplikation von langen Zahlen
- Klasse zur Matrix-Arithmetik ($+$, $-$, $\cdot$, 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.

G. Versionsgeschichte

Diese Übersicht beschreibt die Veränderungen zwischen den einzelnen Versionen dieses Skripts. Rein sprachliche oder kosmetische Änderungen sind nicht extra aufgeführt.

- v7.0
  Intiale Version mit den Inhalten der ersten Woche

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-berlin.de/~filler/lv_ph/algebra2/Skript-Algebra2.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-kl.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-freiburg.de/junker/skripte/Grundlagen-WS1011.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.

[Rö17] Heiko Röglin. Skript zur Vorlesung Logik und diskrete Strukturen, March 2017. http://www.roeglin.org/teaching/Skripte/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.

[Wal17] Matthew P Walker. *Why we sleep : unlocking the power of sleep and dreams*. Scribner, 2017.