Kapitel 1
Einführung

Was ist »theoretische Informatik« und wofür braucht man Theorie überhaupt, wenn man doch eigentlich praktisch arbeiten möchte?

Kaum jemand kann sich heutzutage noch der Allgegenwärtigkeit von digitalen Systemen entziehen – und immer weniger möchten das überhaupt, weil wir uns eine Welt ohne Smartphone, Computer und Internet kaum noch vorstellen können. Da Sie dieses Buch in den Händen halten, gehören Sie vermutlich zu den Menschen, die von der Informatik nicht nur als Nutzer in den Bann gezogen wurden: Vermutlich sind Sie genau wie ich selbst begeistert von den unendlichen Möglichkeiten und kleinen Erfindungen, die nur ein kurzes Computerprogramm entfernt liegen.

Zum Alltag von Informatikern gehört zwar auch, stundenlang im Programmcode nach winzigen Fehlern mit großen Auswirkungen zu suchen – aber die wunderbaren Erfolgsmomente, wenn eine eigene Idee zum ersten Mal auf dem Bildschirm funktionstüchtig zum Leben erwacht, machen das allemal wett. Leider können sich viele Studierende nicht im gleichen Maße für die theoretische Informatik begeistern, wie sie das für die praktische Seite tun:

»Ich möchte Software entwickeln, keine Beweise schreiben.«

Theorie vs. Praxis

Solche und ähnliche Aussagen höre ich oft, wenn die Sinnhaftigkeit der Ausbildung in den theoretischen Grundlagen der Informatik infrage gestellt wird. Rückblickend erkennen die meisten dann zwar, dass ihnen genau diese theoretischen Grundlagen eine große Hilfe bei der Entwicklung von Software sind. Die Lehrveranstaltung »Theoretische Informatik« ist dennoch für viele ein unbeliebtes bis zuweilen gefürchtetes Fach. Die Gründe hierfür sind vielfältig. Oft hängen sie aber damit zusammen, dass man zum Meistern der theoretischen Informatik viele Kompetenzen ausbilden und trainieren muss.

1.1 Kompetenzen für die theoretische Arbeit

Viele Lehrveranstaltungen konzentrieren sich in erster Linie auf die Sachinhalte des Fachs. In diesem Buch möchte ich neben diesen reinen Sachinhalten einen Schwerpunkt auf die Vermittlung und das Training der Kompetenzen legen, die zum theoretischen Arbeiten notwendig sind. Ganz unabhängig von den Erkenntnissen der theoretischen Informatik lohnt es sich ohnehin, die folgenden Fähigkeiten zu fördern:

1.1.1 Abstraktionsvermögen

Modelle analysieren

Auch wenn die Informatik am Ende das Ziel hat, einen Mehrwert für die echte Welt zu schaffen, so arbeiten wir doch immer nur mit *Modellen* der Wirklichkeit. Ein Modell ist ein abstraktes Abbild der Wirklichkeit, das relevante Eigenschaften abbildet und irrelevante weglässt. Die Entscheidung, was relevant ist und was nicht, ist alles andere als trivial und hängt zudem immer vom Kontext ab – wir können also für unterschiedliche Problemstellungen nicht zwingend dasselbe Modell verwenden. Hat man jedoch erst einmal den Dreh raus, hilft abstraktes Denken auch ganz unabhängig von der Informatik beim Lösen vieler kleiner und großer Alltagsprobleme.

1.1.2 Präzises Arbeiten

Aussagen beweisen

Zwei der wichtigsten Themen aus der mathematischen Grundausbildung, die für die theoretische Informatik benötigt werden, sind Logik und Beweistechnik, denn sie ermöglichen uns präzises Argumentieren. Ein einziger Fehler in einer Argumentationskette – und sei er auch noch so klein – kann die gesamte Beweisführung in sich zusammenfallen lassen. Da hilft es auch nichts, wenn in diesem speziellen Fall eine fehlerfreie Argumentation zum selben Ergebnis gekommen wäre.

Kommt Ihnen das bekannt vor? Richtig: Genauso unbarmherzig, wie die mathematische Logik mit Denk- oder Notationsfehlern umgeht, zeigt sich ein Compiler angesichts von Tippfehlern im Code. Eigentlich ist präzises Arbeiten also ein Heimspiel für Informatiker*innen, für viele stellt die Schönheit einer eleganten Beweisführung jedoch nicht denselben Ausgleich für die harte Arbeit dar, wie es ein funktionstüchtiges Programm vermag.

Vielleicht kann ich Sie in diesem Buch davon überzeugen, dass die theoretische Arbeit Sie auch für die praktische Softwareentwicklung voranbringen wird. Ich möchte Ihnen zeigen, dass die theoretischen Themen große Relevanz für die praktische Informatik haben. Zudem werde ich Ihnen dabei helfen, ähnlich wie beim Debuggen eines Programms, Fehler in Argumentationsketten zu erkennen und zu vermeiden.

1.1.3 Frustrationstoleranz und Kreativität

Lösungen finden

Theoretische Fragestellungen und deren Lösungen lassen sich oft in wenigen Zeilen vollständig formulieren. Die kompakten Notationen verstecken dabei, dass der Weg von der Fragestellung zur Lösung alles andere als geradlinig und kurz ist. Im Gegenteil: Bei der Lösung einer spannenden Aufgabe gehört es dazu, einige Umwege und Sackgassen zu erkunden, bevor man die tatsächliche Lösung entdecken kann. Die abschließende schriftliche Darstellung entspricht daher meist nicht dem ursprünglichen Lösungsweg.

Es ist wichtig, dabei den Wert von gescheiterten Versuchen zu erkennen und aus allen erforschten Umwegen neues Wissen herauszuziehen. Ganz genau zu verstehen, *warum* eine Lösungsidee nicht funktioniert hat, ist oftmals der entscheidende Schritt, um eine funktionstüchtige Lösung zu entwickeln. Zusätzlich zu mathematisch präzisem Arbeiten ist hierfür nun kreatives Denken und Knobeln gefordert – und eine hohe Frustrationstoleranz.

Aus diesem Grund zeige ich Ihnen in diesem Buch für einige Fragestellungen nicht nur die Lösung, sondern auch den umständlichen Weg zu dieser Lösung. So erfahren Sie direkt an Beispielen, wie Sie sich Schritt für Schritt der Lösung nähern und von Sackgassen den korrekten Weg zeigen lassen können.

1.1.4 Kommunikationsfähigkeit

Lösungen kommunizieren

Ist die Lösung erst einmal gefunden, so ist die Arbeit aber noch lange nicht getan. Neben der eigenen Erkenntnis ist eine der wichtigsten Tätigkeiten jeder Wissenschaft, die Erkenntnis für andere aufzubereiten. Im Studienalltag dient der Lösungsaufschrieb oft leider nur der Leistungsbewertung. Tatsächlich geht es aber eigentlich darum, verständlich die eigenen Gedanken an eine andere Person zu übermitteln. Ob man sich dabei mathematischer Formeln, erklärender Texte, anschaulicher Grafiken oder eines anderen Mediums bedient, ist unerheblich – am Ende muss die Sprache von Beweisen genauso erlernt und geübt werden wie jede andere Sprache auch.

Auch diese Sprache hat eine Art Dialekte: Über die Verwendung vieler Symbole und Formulierungen herrscht einigermaßen Einigkeit. Manche Notationen bedeuten jedoch in verschiedenen Teildisziplinen der theoretischen Informatik etwas ganz Unterschiedliches und spätestens, wenn es um den Notationsstil geht, werden Sie zu jeder Empfehlung auch eine Person finden können, die Ihnen genau davon abrät.

Die Notation in diesem Buch folgt selbstverständlich meinen persönlichen Vorlieben, die ich mir wiederum von anderen Wissenschaftler*innen abgeschaut habe. Ich versuche es Ihnen so leicht wie möglich zu machen, die Inhalte auch auf andere Notationsstile zu übertragen.

1.2 Themen der theoretischen Informatik

Die Informatik als Gesamtes beschäftigt sich mit der automatisierbaren Verarbeitung von Informationen aller Art und gliedert sich in drei Bereiche. Während in der technischen und der praktischen Informatik die Realisierung von Maschinen beziehungsweise von Anwendungen für solche Maschinen im Fokus stehen, beschäftigt sich die theoretische Informatik mit den grundlegenden Rahmenbedingungen für diese Informationsverarbeitung. Ich gruppiere in diesem Buch die Grundlagenthemen der theoretischen Informatik grob in drei Gebiete: Berechenbarkeitstheorie, Algorithmik und Komplexitätstheorie.

1.2.1 Berechenbarkeitstheorie

Eine der wichtigsten Fragen, die die theoretische Informatik beantworten möchte, lautet: Für welche Problemstellungen lassen sich algorithmisch Lösungen berechnen – und für welche nicht? Um diese Frage anzugehen, muss zuvor geklärt werden, was *berechnen* überhaupt bedeutet. Wir werden sehen, dass es für die Mächtigkeit eines Berechnungsmodells unerheblich ist, ob dieses auf Zahlen, Texten oder ganz anderen Daten arbeiten kann, und dass es ebenso egal ist, ob eine komplizierte Ausgabe produziert wird oder das Modell nur zwischen »ja« und »nein« unterscheiden kann.

Berechnungsmodelle

Es gibt unzählige solcher *Berechnungsmodelle*, von denen bei Weitem nicht alle in diesem Buch Platz finden können. Prominente Vertreter, die wir uns in Teil I des Buches anschauen werden, sind *Formale Sprachen*, *Automaten* und *Grammatiken*, *Turingmaschinen* und *Loop/While-Sprachen*.

Grenzen der Berechenbarkeit

Mit diesen Modellen werden wir für sehr einfache Probleme Berechnungsvorschriften entwickeln und deren Korrektheit beweisen. Umgekehrt werden wir aber auch die Grenzen des Machbaren erkunden und zeigen, dass einige für die praktische Arbeit in der Softwareentwicklung sehr wichtige Fragestellungen leider algorithmisch unlösbar sind. Als Kernaussage fungiert hier die *Unentscheidbarkeit* des *Halteproblems*; diese Aussage werden Sie zudem mithilfe von *Reduktionen* auf andere Probleme zu übertragen lernen.

1.2.2 Algorithmik

Effiziente Algorithmen

Die Algorithmik beschäftigt sich damit, für Probleme Lösungsverfahren zu entwickeln und zu analysieren. Im Gegensatz zur Berechenbarkeitstheorie interessieren wir uns in Teil II des Buches nicht mehr für einen beliebigen Lösungsalgorithmus, sondern für einen möglichst »guten« – wobei wir im Allgemeinen davon ausgehen, dass ein Algorithmus dann besonders gut ist, wenn er zu einer Eingabe möglichst *schnell* eine korrekte Ausgabe liefert.

Entwicklungsparadigmen

Die Einführung von vier Paradigmen – *Brute Force*, *Greedy*, *Divide and Conquer* sowie *Dynamische Programmierung* – wird Ihnen dabei helfen, für eine Vielzahl von Problemen schnell passende und effiziente Algorithmen zu konstruieren.

Analysetechniken

Zu jedem Algorithmus gehört eine Analyse der *Laufzeit* und *Korrektheit*, damit wir Garantien für die Qualität des entwickelten Algorithmus bekommen. Für jedes Paradigma lernen Sie, solche Analysen durchzuführen. Mit der *Landau-Notation* werden Sie für die Analyse ein Maß für die Effizienz eines Algorithmus kennenlernen. Als Beweistechniken für die Korrektheit der Lösungsverfahren werden *Invarianten* und *(strukturelle) Induktion* zum Einsatz kommen.

1.2.3 Komplexitätstheorie

Schwierige Probleme

Die Analyse eines Algorithmus ermöglicht zwar Rückschlüsse auf die Effizienz dieses speziellen Verfahrens, lässt aber die Frage offen, ob es effizientere Algorithmen geben könnte. Im Teil III des Buches beschäftigen wir uns deshalb mit der Analyse der zugrunde liegenden Problemstellung: Gibt es Probleme, die inhärent schwierig zu lösen sind, für die wir also niemals einen effizienten Algorithmus finden können?

P vs. NP

Mithilfe von *unteren Laufzeitschranken* können wir für manche Probleme beweisen, dass wir bereits optimale Algorithmen gefunden haben. Für

viele Probleme ist dies bislang jedoch niemandem gelungen. In der Komplexitätstheorie untersucht man unter anderem zwei bedeutende Gruppen von Problemen, *P* und *NP* genannt. Eine der wichtigsten offenen Fragen der Informatik lautet, ob diese Gruppen identisch sind, ob also die Probleme in NP genauso effizient lösbar sind wie die vermeintlich leichteren Probleme in P. Naheliegenderweise werden wir die Frage in diesem Buch nicht beantworten können. In Form von *Polynomialzeitreduktionen* werden Sie jedoch ein mächtiges Werkzeug kennenlernen, um Probleme anhand ihrer Komplexität zu vergleichen.

Im Ausblick werde ich Ihnen abschließend noch einen Einblick darin geben, wie man auch für sehr schwierige Probleme, die scheinbar keine effiziente Lösung zulassen, für die Praxis brauchbare Algorithmen entwickeln kann. Als Technik stelle ich Ihnen dafür die *Parametrisierte Analyse* vor.

1.3 Anleitung fürs Buch

Voraussetzungen

Dieses Buch richtet sich primär an Leser*innen, die mit den Grundbegriffen der Informatik vertraut sind und sich nun mit der theoretischen Seite des Fachgebiets beschäftigen möchten. Sie sollten daher bereits wissen, was ein *Algorithmus* ist und einfachen *(Pseudo-)Code* schreiben und lesen können. Typische Datenstrukturen (*Arrays*, *Listen* und *Graphen*) sollten Ihnen ebenso geläufig sein wie die wichtigsten Standardalgorithmen der Informatik für das *Sortieren* und *Suchen*. Für eine Einführung oder Auffrischung dieser Themen empfehle ich Ihnen einen Blick in:

> *Boockmeyer, Fischbeck, Neubert: Fit fürs Studium – Informatik. Rheinwerk Computing 2017.*

Grundlagen der diskreten Mathematik (*Zahlen*, *Mengen*, *Funktionen*, *Logik*, *einfache Beweistechnik*) werden in Kapitel 2 kurz besprochen. Dies kann jedoch keine mathematische Grundausbildung ersetzen, sondern soll lediglich Ihr Wissen auffrischen und die Begriffe und Notationen dieses Buches festlegen.

Ebenso wie diese Notation wird die Themenauswahl des Buches sich selten genau mit dem Inhalt einer Einführungsvorlesung in theoretischer Informatik decken. Mein Ziel ist es, Ihnen ein Verständnis zu vermitteln, das Sie dann auch auf andere Modelle und Themenschwerpunkte übertragen können.

Dabei können Sie selbst entscheiden, wie tief Sie in ein Thema einsteigen wollen: Genügt Ihnen die Kenntnis von den Grundaussagen, möchten Sie

diese konzeptuell verstehen, im Detail formal kennenlernen oder selbst mit den Modellen weiterarbeiten können?

Wissen und Intuition

Zu Beginn jedes Kapitels gebe ich Ihnen eine Einführung ins Thema und eine intuitive Erklärung der Aussagen: Worin liegt die praktische Relevanz des Themas? Was sind grundlegende Konzepte, Aussagen und deren Auswirkungen? Welche konkreten Sätze gelten, warum gelten diese und welche Idee steckt hinter deren Beweis?

Formale Definition

An einigen Stellen vereinfacht diese Einführung. Im Anschluss gehe ich daher stets auf die formale Definition der Inhalte ein: Wie sind die zugrunde liegenden Modelle genau definiert, wie notieren wir die angesprochenen Aussagen präzise und formal korrekt?

Werkzeuge

Um mit den Modellen und Aussagen arbeiten zu können, zeige ich Ihnen dann die dazugehörigen Werkzeuge, jeweils in allgemeiner Erklärung und an einem Beispiel demonstriert: Welche Fragestellungen kann man jetzt mit dem Modell beantworten? Welche Werkzeuge gibt es dafür und wie benutzt man sie?

Ausblick

Selbstredend finden in diesem Buch nicht alle spannenden Konzepte der theoretischen Informatik Platz. Am Ende einiger Kapitel gebe ich Ihnen daher in einem Ausblick eine kurze Zusammenfassung von verwandten oder darüber hinausgehenden Inhalten: Welche Themen wurden hier nicht besprochen und wo finden Sie weiterführende Informationen dazu?

Übungsaufgaben und Lösungen

Zum Ende jedes Kapitels finden Sie Aufgaben und die dazugehörigen Lösungen. Zu einzelnen Aufgaben sind im Buch nur Lösungshinweise dargelegt, die ausführlichen Lösungen finden Sie als PDF-Download auf der Website des Rheinwerk-Verlags unter *https://www.rheinwerk-verlag.de/5092* auf der Registerkarte Materialien.

1.4 Danksagungen

Dieses Buch basiert auf den Erfahrungen, die ich in der Arbeit mit Studierenden in den Grundlagenvorlesungen zur Theoretischen Informatik sammeln konnte. Mein herzlicher Dank für viele anregende Gespräche, Konzepte und Experimente zu guter Lehre in theoretischen Veranstaltungen gilt Prof. Christoph Kreitz, Prof. Jürgen Dassow und insbesondere Prof. Tobias Friedrich und Dr. Timo Kötzing, deren Lehrveranstaltungen ich in den vergangenen Jahren mitgestalten durfte.

Immer wieder sprechen mich Studierende darauf an, ob ich aus den gesammelten Erfahrungen nicht ein Buch als Vorlesungsbegleiter schreiben könnte. Dass dies nun tatsächlich geklappt hat, ist dem Team des Rheinwerk Verlags zu verdanken. Besonders danken möchte ich meiner Lektorin Almut Poll, die das Projekt betreut hat und trotz aller Verzögerungen und späten Anpassungen nie das Vertrauen in meine Arbeit verloren hat, sowie Norbert Englert, der die Umsetzung des von mir gewünschten Formelformats möglich gemacht hat.

Torsten T. Will ist es als Reviewer nicht nur gelungen, vielerlei Verbesserungen ins Buch einzubringen, sondern er hat darüber hinaus seine Anmerkungen und Gedanken auch noch so formuliert, dass ich mich über jeden Kommentar freuen konnte und durch seine Randnotizen sogar regelmäßig zum Schmunzeln gebracht wurde. Danke dafür!

Eine ganze Reihe von Personen hat Teile des Manuskripts testgelesen, mir dabei geholfen, Erklärungen noch verständlicher zu machen, und auch den einen oder anderen Fehler entdeckt. Vielen Dank an Arne Boockmeyer, Jonas Chromik, Philipp Fischbeck, Martin Krejca, Jannik Peters, Martin Schirneck, Sebastian Serth und Jennifer Stamm.

Trotz aller Sorgfalt und Reviews gibt es sicherlich immer noch einiges, was am Buch verbessert werden kann. Lassen Sie mir gerne über die Rheinwerk-Website Ihr Feedback zum Buch zukommen – egal ob Lob oder Kritik, mein Dank gilt Ihnen in jedem Fall!