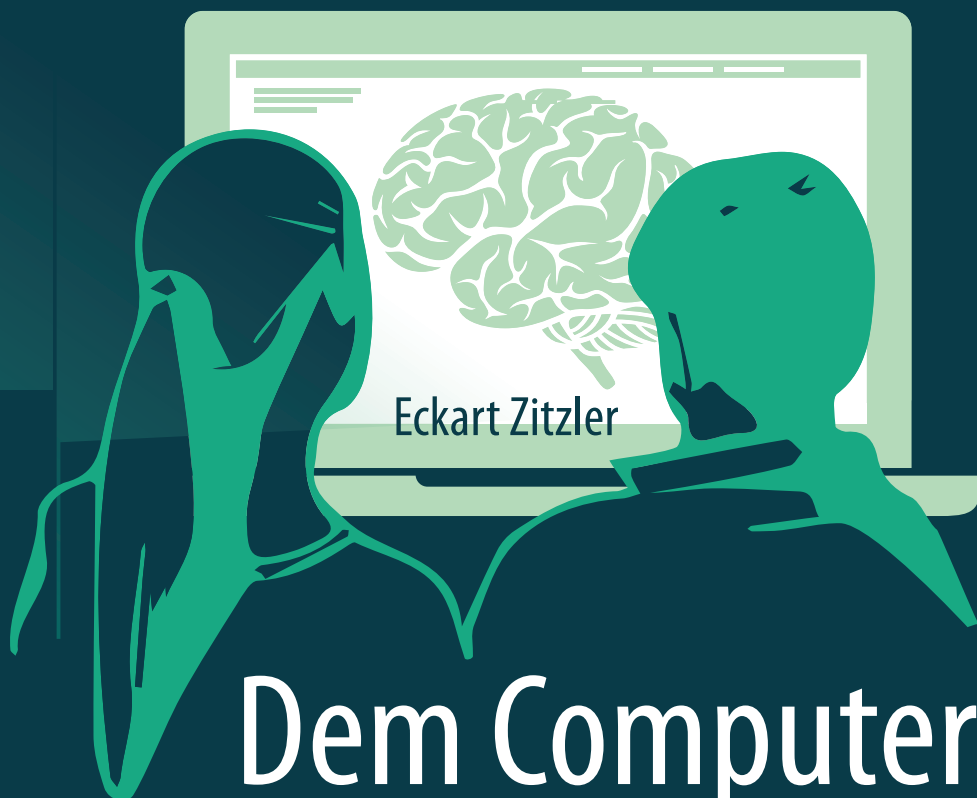




Dem Computer ins Hirn geschaut

Informatik entdecken, verstehen
und querdenken

EBOOK INSIDE



Springer

Dem Computer ins Hirn geschaut

Eckart Zitzler

Dem Computer ins Hirn geschaut

Informatik entdecken, verstehen
und querdenken

Mit Zeichnungen von Susanne Staubli



Springer

Eckart Zitzler
Pädagogische Hochschule Bern
Bern, Schweiz

ISBN 978-3-662-53665-0 ISBN 978-3-662-53666-7 (eBook)
DOI 10.1007/978-3-662-53666-7

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

© Springer-Verlag GmbH Deutschland 2017

Zeichnungen, wenn nicht anders erwähnt: Susanne Staubli, Zürich, nach Vorlagen von und in Zusammenarbeit mit Eckart Zitzler. Diese Zeichnungen sind von Autor und Zeichnerin unter die Creative Commons-Lizenz CC BY-NC 4.0 gestellt und stehen unter www.eckartzitzler.ch zum Download zur Verfügung.

Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Jede Verwertung, die nicht ausdrücklich vom Urheberrechtsgesetz zugelassen ist, bedarf der vorherigen Zustimmung des Verlags. Das gilt insbesondere für Vervielfältigungen, Bearbeitungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Der Verlag, die Autoren und die Herausgeber gehen davon aus, dass die Angaben und Informationen in diesem Werk zum Zeitpunkt der Veröffentlichung vollständig und korrekt sind. Weder der Verlag noch die Autoren oder die Herausgeber übernehmen, ausdrücklich oder implizit, Gewähr für den Inhalt des Werkes, etwaige Fehler oder Äußerungen. Der Verlag bleibt im Hinblick auf geografische Zuordnungen und Gebietsbezeichnungen in veröffentlichten Karten und Institutionsadressen neutral.

Planung: Dr. Andreas Rüdinger

Einbandabbildung: © Susanne Staubli, Eckart Zitzler; deblik, Berlin

Gedruckt auf säurefreiem und chlorfrei gebleichtem Papier

Springer ist Teil von Springer Nature

Die eingetragene Gesellschaft ist Springer-Verlag GmbH Deutschland

Die Anschrift der Gesellschaft ist: Heidelberger Platz 3, 14197 Berlin, Germany

Für e. g. k. j. – ein Glück, kostbar jederzeit

Inhaltsverzeichnis

Prolog: Über dieses Buch	XIII
1 Was ist ein Computer?	1
Maschinen zum Rechnen	2
Die Erscheinungsbilder des Computers sind vielfältig	2
Ein lebendes Fossil: der Abakus	5
Wenn es beim Rechnen knackt, knirscht und surrt	8
Die Geburt des modernen Computers	11
Drei Wesensmerkmale eines Computers	14
Des Computers Kern: die Turing-Maschine	15
Ein Maschinenkonstrukt, um das menschliche	
Rechnen nachzubilden	16
Rechnen als Textbearbeitung	17
Eine Rechenoperation wird über eine Tabelle beschrieben	18
Mehrere Tabellen können mehrere Rechenoperationen	
ausführen	21
Sinn und Unsinn von Turings Minimalcomputer	24
Natürliche Informationsverarbeitung	26
Das Band des Lebens: die DNA	27
Proteine beeinflussen den DNA-Ablesevorgang	30
Sind Kolibakterien Turing-Maschinen?	32
Rechnen mit DNA	36
Fazit	40
Zum Nach- und Weiterlesen	41

2 Schalten und Walten	43
Am Anfang ist die Zahl	44
Der Stromkreis als elementarer Zeichenträger	44
Von Lichtmustern zu Zahlen	47
Komm, wir bauen einen Taschenrechner!	50
Geniale Muster	52
Kleines Ding, große Wirkung	54
Elektrischer Staffellauf	55
Eine kleine Knüpfkunde	58
Viele mit vielen koppeln	61
Wie Proteine über Gene miteinander verknüpft sind	62
Was Lattentore mit Logik zu tun haben	65
Von Stromkreisen zu Bausteinen	65
Bausteine bauen	69
Vom Baustein zur Schaltung	73
Gibt es Bausteine in der Zelle?	75
Feedback erwünscht!	76
Der Taschenrechner wird erweitert	76
Was Feedback bewirken kann	78
Alles andere als ein Flop: das Flipflop	81
Wenn Viren Zustände haben	85
Das Feuerwerk im Kopf	87
Die Nervenzelle, das Super-Relais	88
Neuronale Verschaltungen	90
Fazit	92
Zum Nach- und Weiterlesen	93
 3 Verschlungene Pfade	 95
Vorneweg: das Küchen-Einmaleins	96
Eine Fabrik im Kleinen: die Küche	97
Was Prozessoren und Zellen mit Küchen gemein haben	98
Pfade für Daten	100
Ein Werk zum Rechnen	100
Zahlenbehälter	102
Transportgleise	106
Nicht Steuermann, sondern Steuerwerk	109
An den Fäden ziehen – und rechnen	109
Malnehmen mal umständlich	112
Der elektrische Datenkoch	115
Im Gleichschritt – Marsch!	117

Die Software betritt die Bühne	119
Befehle erfinden	120
Die Kontrolle haben	123
Maschinensprech	125
Programme ausführen	127
Wenn Zellen Stoffe wechseln	133
Alles eine Frage der Chemie	133
Regulieren, regulieren, regulieren	136
Fazit	138
Zum Nach- und Weiterlesen	139
 4 Computer-Anatomie	 141
Eingebettet sein: im Austausch mit der Welt	142
Anatomische Grundlagen	143
Der Urbauplan	146
Wie die Welt in den Computer kommt – und wieder hinaus	151
Ein Rückgrat für alle	156
Ein System, um den Betrieb zu meistern	160
Schneller sein: jeder Augenblick zählt	162
Die Relativität der Zeit	163
Vom aktiven zum passiven Warten	165
Die Last auf mehrere Schultern verteilen	167
Mehr ist nicht immer mehr	171
Allgegenwärtig sein: mehrere Dinge gleichzeitig erledigen	175
Der Prozess oder die Illusion der Gleichzeitigkeit	175
Teilen ganz praktisch	179
Zusammenarbeit will gelernt sein	181
One Program To Rule Them All	184
Fazit	188
Zum Nach- und Weiterlesen	189
 5 Speichern und Erinnern	 191
Vom Leben eines Bücherwurms	192
Wie man Informationen haltbar macht	195
Digitale Gravuren	195
Speicher organisieren	199
Die Eier legende Wollmilchsau	203
Ordnung muss sein: den Daten ihre Struktur	208

X Inhaltsverzeichnis

Von Stapeln und Feldern	209
Der Baum der Sucherkennntnis	213
Zeige mir deinen Nächsten	218
Welche Struktur ist die richtige?	223
Das Leben und die Einprägungen	226
Informationen einfangen	227
Informationen nutzen	230
Fazit	234
Zum Nach- und Weiterlesen	234
6 Netze spinnen	237
Zusammen ist man weniger allein	238
Hallo Du: eine Leitung für zwei	240
Die Familie wächst: eine Leitung für mehrere	244
Von der Familie zur Gemeinschaft:	
Leitungen überbrücken	248
Netze vernetzen	252
Auf Routen von Netz zu Netz	253
Weltweite Anschriften	256
Die Stockwerke des Internets	260
Alles wächst zusammen	263
Von Kunden und Dienstleistern	264
Einer für alle, alle für einen	268
Der Weltcomputer	272
Zwischennetze in der Natur	275
Voller Netze: der Mensch	275
Gut vernetzt: der Ameisenstaat	278
Fazit	281
Zum Nach- und Weiterlesen	282
7 Die Kunst der Darstellung	283
Die Welt in Nullen und Einsen	284
Was ist Information?	284
Von Zeichen und Codes	287
Grundlegende Darstellungsarten	289
Aufs Format kommt's an	291
Biologische Codes	296
Heiße Luft	298
Wie viel Information steckt da drin?	299
Komprimierung: Wie man aus Daten die Luft rauslässt	303

Steganografie: Luft hinzufügen	307
Genetische Datenakrobatik	310
Geheimniskrämereien	312
Alter Wein in neuen Schläuchen: Verschlüsseln mit Schlüssel	313
Öffentlich Geheimnisse teilen	320
Wenn der eine nicht ohne den anderen kann: zweiteilige Schlüssel	325
Yin und Yang im Alltag: Public-Key-Kryptografie	330
Duftcodes – und wie sie gehackt werden	334
Fazit	337
Zum Nach- und Weiterlesen	337
8 Denk!	339
Knifflige Probleme – bewährte Strategien	340
Von Verhaltensprogrammen und Verhaltensmustern	341
Wenn Informatiker Probleme haben	343
Der Weg ist das Ziel: Was es mit Algorithmen auf sich hat	346
Probieren geht über Studieren	350
Aus kleinen Lösungen werden große	354
Große Probleme in kleine zerlegen	356
Vom Rechenknecht zum Schlaumeier	360
Vom stereotypen zum intelligenten Verhalten	360
Lernen und Anpassungsfähigkeit	363
Spielen und Suche	367
Sprache und Kommunikation	371
Grenzen der Berechenbarkeit	378
Ich weiß, dass ich nichts weiß	379
Unentscheidbarkeit	382
Von praktischen und unpraktischen Algorithmen	385
P oder nicht P, das ist hier die Frage	389
Fazit	396
Zum Nach- und Weiterlesen	397
9 Im Licht der Evolution	399
Vom Entstehen	400
Schicht für Schicht zur Komplexität	400
Gefahren lauern überall	407

XII Inhaltsverzeichnis

Vom Entwickeln	414
Handwerk der Programmierung	416
Wie komplexe Software-Systeme gebaut werden	423
Vom Entdecken	428
Evolution verstehen	428
Evolution imitieren	436
Fazit	441
Zum Nach- und Weiterlesen	441
 Epilog: Zur Biologie der Informatik	 443
Die Natur der Informatik ist interdisziplinär	443
Dank	445
 Stichwortverzeichnis	 449

Prolog: Über dieses Buch

Der Mensch ist voller Rätsel. Möchten Sie nicht auch gelegentlich wissen, was im Hirn Ihres Gegenübers vor sich geht – oder in Ihrem eigenen? Beim Computer ist es ähnlich: Die Informationstechnologie durchdringt unseren Alltag, gleichzeitig durchschauen wir selten, was im Hintergrund abläuft. Wenn Sie ...

- verstehen möchten, wie Computer und das Internet funktionieren,
- eine erfrischende, bildhafte Einführung in die Informatik suchen, die fundamentale Ideen konkret erklärt, aber mathematische Details außen vor lässt, und
- gerne über den Tellerrand hinausschauen und verblüffende Parallelen zum Lebendigen entdecken wollen,

dann ist dieses Buch das Richtige für Sie. Ein Buch über den Umgang mit Informationen und das Zurechtfinden in der Welt. Denn: Information ist überall.

Zwei Welten treffen aufeinander

Informatik – ein Kunstwort, das die Begriffe Information und Automatik beinhaltet – steht für automatisierte Informationsverarbeitung. Diese zutreffende, aber schwer fassbare Umschreibung wird wohl kaum Vorstellungen von einem spannenden und aufregenden Fach hervorrufen, geschweige denn

die Lust, sich eingehender mit dem Thema zu beschäftigen. Informatik scheint naturgemäß eine trockene Disziplin zu sein, abstrakt und lebensfern. Ganz anders die Biologie: Viel greifbarer und näher wirkt sie, schließlich geht es um alles, was um uns herum kreucht und fleucht, und nicht zuletzt um uns selbst. Stellen wir Informatik und Biologie gegenüber, so könnten die Gegensätze wohl nicht größer sein: auf der einen Seite die Wissenschaft von der Informationsverarbeitung mittels Maschinen, auf der anderen Seite die Wissenschaft von den allgemeinen Gesetzmäßigkeiten des Lebendigen.

Doch was auf den ersten Blick so unvereinbar wirkt, ist sich näher als vermutet – Gegensätze ziehen sich bekanntermaßen an. Schon von Anfang an war die Informatik eng verwoben mit biologischer Erkenntnis. Die Querbezüge zur Biologie sind verblüffend und unzählig. Viele für die Informatik zentrale Fragen entpuppen sich als Fragen des Lebens an sich, vor allem wenn man sie über die Fächergrenzen hinweg in einem größeren Kontext betrachtet. Dabei dient das Studium des Lebendigen als unerschöpfliche Quelle der Inspiration. Das heißt für mich auch: Die Biologie kann helfen, der Informatik Leben einzuhauchen und deren vielseitigen Facetten zur Geltung zu bringen.

Was dieses Buch bietet – und was nicht

Das Buch verfolgt einerseits den Anspruch, fundamentale Ideen der Informatik verständlich und anschaulich zu vermitteln, ohne mit Formeln oder Programmcode hantieren zu müssen. Es geht um die zentralen Konzepte, die vernetzten Computersystemen und ganz allgemein der automatisierten Informationsverarbeitung zugrunde liegen. So gesehen richtet sich dieses Buch zunächst einmal an alle, die gerne wissen möchten, was sich hinter der Informatik verbirgt, und eine Einführung in dieses Fach ohne große Einstiegshürden suchen. Andererseits versucht es, Fragestellungen und Lösungsansätze aus der Informatik entsprechenden Phänomenen in der Lebenswelt gegenüberzustellen. Damit spricht es auch diejenigen an, die Interesse an interdisziplinären Themen haben, die gerne Querbezüge und Zusammenhänge entdecken und sich für die übergreifenden Fragen begeistern. Das Buch will somit eine unterhaltsame und anregende Einführung in die Welt der Informatik bieten und einladen zum Stöbern, Entdecken und Anregenlassen. Ob dieser Versuch gelungen ist, mögen dann Sie, liebe Leserin, lieber Leser, beurteilen.

Einen Computerkurs wird Ihnen dieses Buch allerdings nicht bieten können. Sie werden nach dem Lesen weder programmieren noch Ihren eigenen

Computer besser bedienen können. Sie werden auch keine Aufgaben finden, mit denen Sie die vorgestellten Konzepte in der Praxis umsetzen könnten. Das Buch ist nicht primär als Lehrmittel für die Schule konzipiert, wenngleich es als Anregung dienen kann, wie Informatikunterricht spannend und lebensnah gestaltet werden kann – die Zeichnungen in diesem Buch sind frei verfügbar und somit für den Unterricht nutzbar (www.eckartzitzler.ch). Und obwohl keine speziellen Vorkenntnisse, sei es nun in Informatik, Mathematik oder Biologie, vorausgesetzt werden und ich mich um einfache, nachvollziehbare Erklärungen bemüht habe, kann Ihnen zwischendurch schon mal der Kopf rauchen. Die Materie ist durchaus komplex, erwarten Sie also bitte keine gemütliche Bettlektüre (womit natürlich nicht ausgeschlossen ist, dieses Buch vor dem Schlafengehen zu lesen). Auch aktuelle, brandheiße Themen der Informatik werden Sie in diesem Buch vergeblich suchen. Es geht hier um die langfristigen, beständigen Konzepte, die sich in der Informatik etabliert haben. Es geht um die Hintergründe: Wie funktioniert das, was steckt dahinter?

Bevor es losgeht

Wenn Ihnen der hier skizzierte Ansatz zusagt und Sie sich entschließen, dieses Buch zu lesen oder zumindest schnuppernd zu erkunden, so lassen Sie mich noch kurz die Konzeption erläutern. Das Buch ist schichtenartig aufgebaut: Die neun Kapitel spiegeln verschiedene Betrachtungsebenen wider und sind so angeordnet, dass entwicklungsgeschichtlich ältere Konzepte zuerst behandelt werden. Schicht für Schicht arbeiten wir uns von der untersten Hardware-Ebene – dem Rechnen mit Strom über stromgesteuerte Schalter – hoch zu den Software-Ebenen, wo es um Künstliche Intelligenz und Anwendungen in der Molekularbiologie geht (Abb. P.1). Diese Herangehensweise mag auf den ersten Blick befremdlich erscheinen, da sie vom Unbekannten zum Bekannten führt – intuitiv naheliegender wäre der umgekehrte Weg. Doch wenn Sie diesem Aufbau folgen, lernen Sie Computersysteme quasi von der Pike auf kennen und können schrittweise nachvollziehen, wie alles aufeinander aufbaut und zusammenhängt. Letztlich ist das Vorgehen aber auch eine persönliche Geschmacksache und es spricht nichts dagegen, die Reihenfolge kurzerhand umzudrehen, denn:

Die Leseroute, die ich Ihnen mit der Reihenfolge der Kapitel vorgebe, ist als Vorschlag zu verstehen, den Sie selbstverständlich abändern und an Ihre Bedürfnisse anpassen können. Die Kapitel sind so konzipiert, dass jedes für sich ein abgeschlossenes Thema behandelt und separat studiert werden kann,

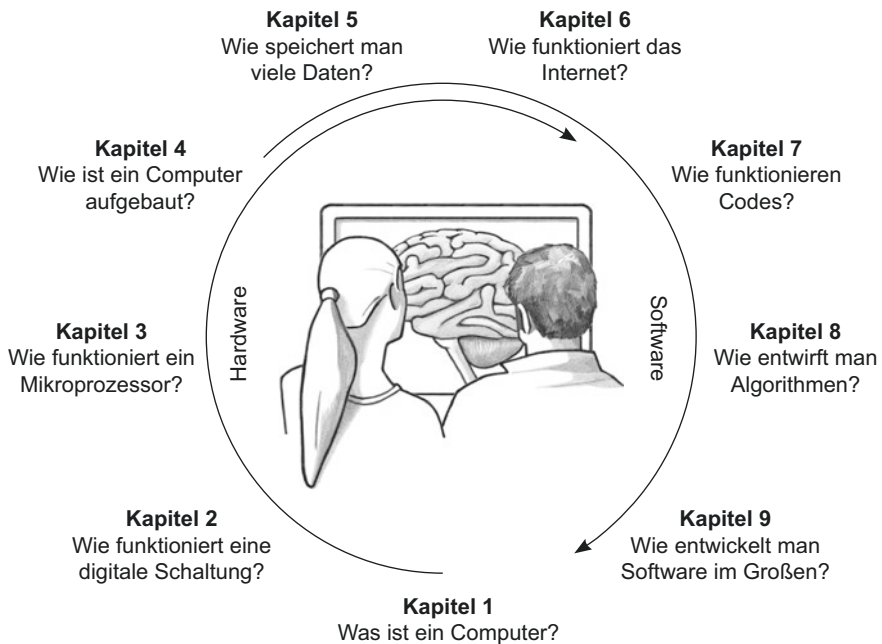


Abb. P.1 Dem Buch ins Innere geschaut. (© Susanne Staubli, Eckart Zitzler)

auch wenn sie inhaltlich aufeinander aufbauen. Jedes Kapitel ist in drei bis fünf Themenblöcke unterteilt, jeder von diesen wiederum in verdauliche Häppchen. Der Text ist angereichert mit veranschaulichenden Abbildungen, Beispielen und Anmerkungen; die Beispiele erläutern einzelne Aspekte vertieft und konkret, die Anmerkungen enthalten Hintergrundinformationen, die die Ausführungen ergänzen, aber für das Verständnis nicht essenziell sind. Ich möchte Sie ermuntern, das Buch ganz nach Interesse und Gusto durchzugehen, vielleicht Teile zu überspringen, andere nur blättern und dort vertieft einzusteigen, wo es Sie gerade packt. Als Einstieg empfehle ich Ihnen, das erste Kapitel zu lesen, da es das Prinzip des Buchs illustriert und einige Grundlagen darlegt. Noch ein Hinweis: Aus Gründen der einfacheren Lesbarkeit führe ich im Folgenden nicht immer die männliche und die weibliche Form auf, sondern verwende sie im Wechsel; es sind jedoch stets Menschen jeglichen Geschlechts gleichermaßen gemeint. Und jetzt wünsche ich Ihnen eine genussvolle und anregende Lektüre und hoffe, dass Ihnen das ein oder andere „Aha!“ über den Weg läuft. Los geht's!

1

Was ist ein Computer?

Überblick

Dieses Kapitel beschäftigt sich mit dem Wesen des Computers und stellt eine Einführung in die automatisierte Informationsverarbeitung dar. Zunächst gehen wir der Frage nach, was einen Computer ausmacht und welche Ideen ihm zugrunde liegen. Dazu unternehmen wir einen Streifzug durch die Geschichte und werfen einen Blick auf die wichtigsten Innovationen, die zum modernen Computer geführt haben. Zudem lernen wir die Turing-Maschine kennen, eine skurrile, minimalistische Rechenmaschine, die zunächst nur in der Vorstellung existiert und trotzdem für die Informatik von zentraler Bedeutung ist; an ihr studieren wir die Grundprinzipien eines Computers konkret und anschaulich. Schließlich wechseln wir die Perspektive und untersuchen anhand einer Bakterie, wie in der Natur Informationen verarbeitet werden. Wir stellen die Bakterie der Turing-Maschine gegenüber und entdecken verblüffende Analogien. Dass es sich dabei nicht nur um eine philosophische Betrachtung handelt, sehen wir am Schluss: wenn sich nämlich Biologie und Informatik in der Realität treffen und es darum geht, mit DNA-Molekülen zu rechnen.

Die Kapitelüberschrift mag harmlos daherkommen, doch die Frage hat es in sich. Zwar gehen wir tagtäglich mit Computern um und haben insofern eine intuitive Vorstellung davon, was mit dem Begriff des Computers gemeint ist. Das Verständnis in Worte zu fassen und auf den Punkt zu bringen, ist hingegen gar nicht so einfach – fällt Ihnen spontan eine eingängige Formulierung ein? Wie auch immer: Will man sich mit der Informatik beschäftigen, kommt man um eine genaue Antwort nicht herum, denn um den Computer dreht sich dabei ja alles. Wie man ihn baut, wie man ihn einsetzt und was man generell mit ihm bewerkstelligen kann. Nicht ohne

Grund spricht man im Englischen von *Computer Science*, also von einer Wissenschaft, deren Untersuchungsgegenstand der Computer ist. Was also ist ein Computer?

Maschinen zum Rechnen

Ein Auto ist ein Ding, das fährt, ein Flugzeug ist ein Ding, das fliegt, und ein Computer ist ein Ding, das rechnet – so oder so ähnlich könnte es in einem Lexikon für Kinder stehen. Wollen wir es genauer wissen, so lassen sich unterschiedlichste Definitionen finden. Egal ob kurz oder lang, im Kern sagen sie alle das Gleiche, nur leider verhält es sich mit ihnen ungefähr so wie mit einer Trockensuppe: Ohne Erläuterung bzw. Flüssigkeit sind sie ausgesprochen staubig. Leichter lässt sich das Wesen eines Computers anhand konkreter Beispiele veranschaulichen, insbesondere an Beispielen, die Meilensteine in seiner Entwicklungsgeschichte markieren. Wir werden also im Folgenden einige Vorläufer des Computers kennenlernen und drei fundamentale Prinzipien herauschälen. Doch beginnen wir zunächst mit den Bildern, die wir heute mit dem Wort Computer verbinden.

Die Erscheinungsbilder des Computers sind vielfältig

Wenn ich Ihnen sagen würde, dass in dem in Abb. 1.1 dargestellten Karton – über seine Größe sei nichts ausgesagt – ein Computer steckt, wie würden Sie sich dann den Inhalt vorstellen?

Sicherlich kommt Ihnen ein Bildschirm in den Sinn und ein entsprechendes Gehäuse, das über ein oder mehrere seitlich oder rückseitig angebrachte

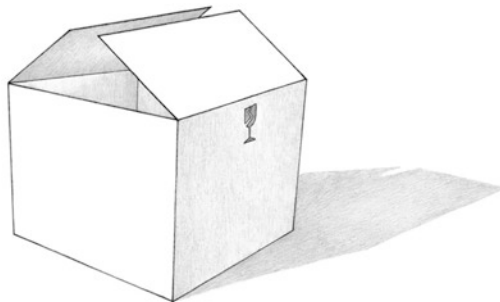


Abb. 1.1 Der Computer im Karton. (© Susanne Staubli, Eckart Zitzler)

Buchsen verfügt. Vielleicht ist noch eine Tastatur vorhanden, die entweder integriert oder als separate Komponente angeschlossen ist? Oder schwebt Ihnen ein Hochleistungscomputer vor, eine dieser riesigen Rechenanlagen, die den Platz eines einzelnen Gehäuses bei Weitem sprengt und stattdessen in unzähligen, mit aufwändigen Kühlsystemen ausgestatteten Schränken untergebracht ist, die wiederum ganze Hallen füllen? Diese Erscheinungsbilder haben sich im Laufe der Zeit herausgebildet und repräsentieren verschiedene Anwendungszwecke, siehe Anm. 1.1. Natürlich decken sie nur einen Teil eines Spektrums an Varianten ab, die sich hinsichtlich ihrer Leistungsfähigkeit, ihres Stromverbrauchs, ihrer Handlichkeit, ihrer Robustheit, ihrer Kosten usw. unterscheiden. Sie alle verkörpern die klassische Sicht auf den Computer: ein eigenständiger Apparat, mit dem Berechnungen oder ganz allgemein Programme ausgeführt werden können.

Anmerkung 1.1: Spielarten von Computern

Abb. 1.2 zeigt eine Auswahl historischer Erscheinungsformen von Computern. Der klassische Desktop-Computer (unten links), bestehend aus einem zentralen Gehäuse sowie weiteren, häufig separaten Komponenten wie Bildschirm und Tastatur, trat bereits in der 1970er-Jahren in Erscheinung. Gut 30 Jahre später sind tragbare Geräte sehr verbreitet: Laptops (oben Mitte), die die Komponenten eines Desktop-Computers in einem kompakten, leichten Gerät vereinen, sowie Tablet-Computer (oben links), die über ein flaches Gehäuse mit berührungsempfindlichem Bildschirm verfügen und per Fingerbewegungen gesteuert werden. Supercomputer (rechts) sind auf das Hochleistungsrechnen ausgerichtet und nur in großen Rechenzentren anzutreffen.

Die andere Sicht ist die des eingebetteten Computers, der in andere Gerätschaften oder Anlagen eingebunden ist und dort als Teil eines Ganzen weitestgehend unsichtbar seine Dienste verrichtet; ein Paradebeispiel hierfür ist die Raumfahrt (Abb. 1.3). Solche auf einen spezifischen technischen Kontext zugeschnittene Computertechnologie finden wir überall in unserem Alltag, z. B. im Fotoapparat und in der Waschmaschine. Auch im Krankenhaus ist sie nicht mehr wegzudenken; die Bandbreite reicht vom Computertomografen bis zum Blutdruckcomputer. Letztlich treffen wir eingebettete Computer überall dort an, wo etwas zu steuern ist – beispielsweise im Auto, in der Gepäckförderanlage oder im Wasserkraftwerk. Computer sind also allgegenwärtig und vor allem vielfältig in ihrer Erscheinung.

Das war natürlich nicht immer so. Der Begriff des Computers unterliegt einem ständigen Wandel, gemäß dem technologischen Fortschritt. Während um 1950 noch riesige Konstruktionen mit diesem Wort assoziiert wurden und

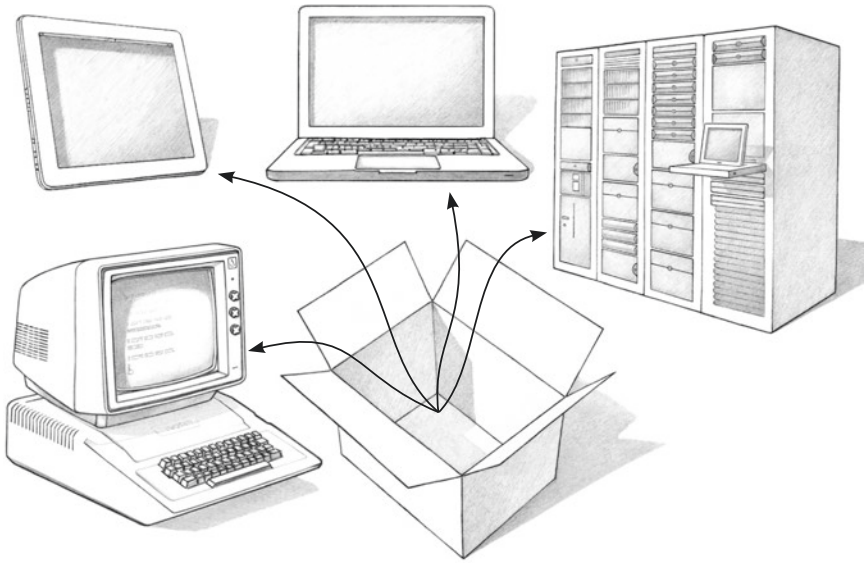


Abb. 1.2 Varianten von Computern. (© Susanne Staubli, Eckart Zitzler)

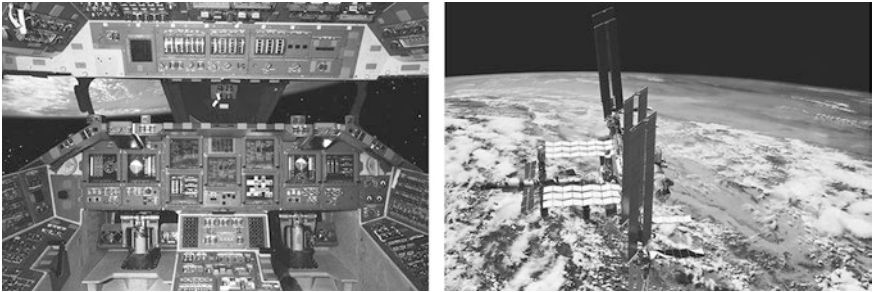


Abb. 1.3 Die computerbasierten Steuerungssysteme im Cockpit einer Raumfähre (links) und eine Raumstation (rechts), die von einer Vielzahl von Computern gesteuert wird. (© NASA)

die Vorstellung, dass einmal in jedem Haushalt ein persönlicher Computer stehen könne, absurd erschien, hatte sich dieses Bild 40 Jahre später grundlegend gewandelt. Nun verknüpfte man mit dem Begriff ein Gerät, das sich bequem zuhause auf dem Schreibtisch platzieren ließ: den Personal Computer (vgl. Abb. 1.2 links unten). Heutzutage ist die Bezeichnung aufgrund der Vielfalt und der rasanten Entwicklung viel verschwommener, und es ist anzunehmen, dass sich unsere Bilder in den kommenden Jahren wiederum stark verändern werden.

Die Grundideen sind hingegen immer noch dieselben. Ihre Wurzeln reichen weit in die menschliche Kulturgeschichte zurück – kaum verwunderlich, wenn wir das Wort Computer auf seine ursprüngliche Bedeutung zurückführen, nämlich Rechner von lateinisch *computare*, zu Deutsch berechnen/zusammenrechnen. Die Motivation war ursprünglich, immer ausgeklügeltere Hilfsmittel zum Rechnen zu ersinnen. Werfen wir also einen Blick auf die Anfänge und schauen uns an, wie sich die Computeridee stufenweise entwickelt hat.

Ein lebendes Fossil: der Abakus

Bereits vor schätzungsweise 3000 Jahren trat mit dem Abakus (Abb. 1.4) ein erstes Rechengerät in Erscheinung, das als Vorläufer des Rechnens mit Stift und Papier betrachtet werden kann – nicht von ungefähr, schließlich steht sein Name sinngemäß für „eine mit Sand oder Staub bedeckte Fläche zum Schreiben“. Entsprechend dient dieses altertümliche Rechenutensil dem Zweck, Zahlen festzuhalten und somit das Gehirn zu entlasten: Indem wir nicht alles im Kopf behalten müssen, sind wir ausdauernder und machen weniger Fehler beim Rechnen.

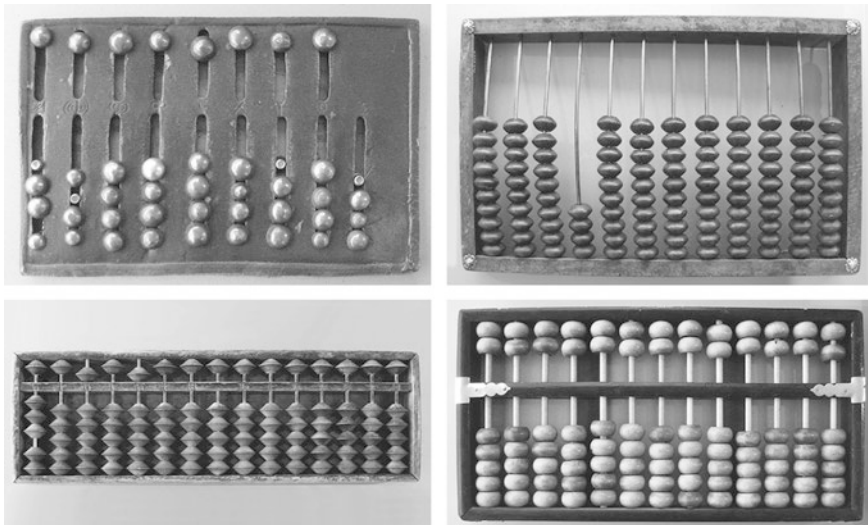


Abb. 1.4 Verschiedene Abakus-Varianten: ein römischer Handabakus (oben links), ein russischer Stschoty (oben rechts), ein japanischer Soroban (unten links) und ein chinesischer Suanpan (unten rechts). (© Arithmeum, Rheinische Friedrich-Wilhelms-Universität Bonn)

Auf dem Abakus wird eine Dezimalzahl, wie wir es gewohnt sind, als Folge von Ziffern dargestellt, und die Wertigkeit einer Ziffer wird über eine entsprechende Anzahl von Kugeln repräsentiert. Die Kugeln sind in der Regel aufgereiht an Stäben aus Metall oder Holz, die parallel zueinander in einen Holzrahmen eingespannt sind. Jeder Stab fungiert als Platzhalter für eine Ziffer, und indem wir die Kugeln auf dem Stab nach unten oder oben schieben, können wir beliebige Werte zwischen 0 und 9 einstellen. Die Gesamtzahl ergibt sich, indem wir die codierten Ziffern von den Stäben ablesen und aneinanderfügen; Abb. 1.5 oben visualisiert das Prinzip für einen polnischen Liczydło, die Leserichtung ist hier wie gewohnt von rechts nach links. Die größte Zahl, die der Abakus aufnehmen kann, ist demnach durch die Anzahl der Stäbe bestimmt: Bei zehn Stellen z. B. liegt die Darstellungsgrenze bei 9.999.999.999.

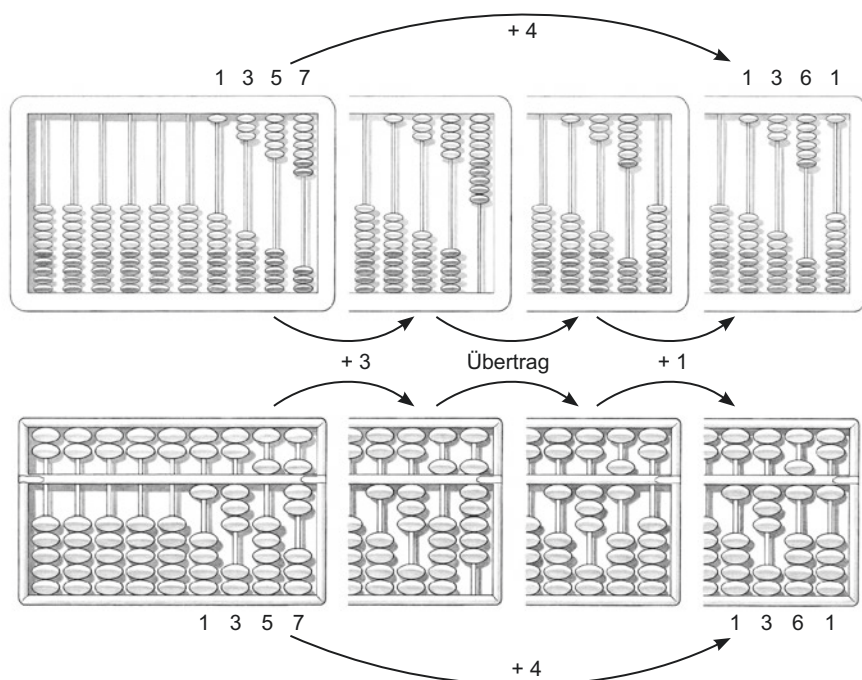


Abb. 1.5 Illustration, wie mit zwei Abakus-Varianten – einem polnischen Liczydło (oben) und einem chinesischen Suanpan (unten) – die Zahl 1357 dargestellt und anschließend die Zahl 4 dazuaddiert werden kann. (© Susanne Staubli, Eckart Zitzler)

Die genaue Ausgestaltung eines Abakus kann variieren, nicht nur hinsichtlich der verwendeten Materialien, sondern auch bezüglich der Bauart. Bei einem polnischen Abakus (Abb. 1.5 oben) z. B. besitzen die jeweils unteren fünf Kugeln eine andere Farbe als die oberen fünf Kugeln – zur besseren Übersichtlichkeit. Aus dem gleichen Grund ist bei einem chinesischen Abakus (Abb. 1.5 unten) eine Querleiste in den Rahmen eingebaut, der die Kugeln auf den Stäben in zwei Gruppen unterteilt. Die Kugeln der unteren Hälfte weisen wie beim polnischen Abakus eine Wertigkeit von eins auf, aber die oberen Kugeln zählen fünf. Summiert man die Wertigkeiten der Kugeln, die Richtung Mittelleiste geschoben sind, auf, dann kommt man auf die repräsentierte Ziffer. Abgesehen von diesen länderspezifischen Spielarten ist die Funktionsweise jedoch immer dieselbe.

Des Weiteren erlaubt der Abakus, grundlegende Operationen wie Addieren und Subtrahieren effizient und auf einfache Art und Weise auszuführen – die Vereinfachung wird erzielt, indem der Rechengvorgang in kleinere, schnell durchzuführende Einzelschritte zerlegt wird. Wir kennen diese Vorgehensweise vom Rechnen mit Papier und Stift: Beispielsweise werden zwei aus mehreren Ziffern bestehende Zahlen addiert, indem die Einzelziffern beider Zahlen positionsweise, von rechts nach links, aufsummiert und mögliche Überträge bei der nächsten Position dazugezählt werden. Das gleiche Schema findet beim Abakus Anwendung, wie Bsp. 1.1 zeigt. Dabei dient der Abakus genauso wie das Papier als Gedächtnisstütze, mit der sich nach jedem Einzelschritt das aktuelle Zwischenergebnis festhalten lässt; nach Durchführung aller Einzelschritte zeigt er das Endergebnis an.

Beispiel 1.1: Addieren mit dem Abakus

Nehmen wir an, wir wollten zur Zahl 1357 die Zahl 4 addieren (vgl. Abb. 1.5). Der Additionsvorgang beginnt bei der kleinstwertigen Position, also bei der Ziffer 7 bzw. dem Stab rechts außen. Zuerst versuchen wir dort vier Kugeln nach oben bzw. Richtung Mittelleiste zu verschieben. Da jedoch nur noch drei einwertige Kugeln zur Verfügung stehen, können wir an dieser Position nur drei dazuzählen. Jetzt repräsentiert der rechte Stab die Zahl 10, also einen Übertrag. Wir lösen den Übertrag auf, indem wir eine Kugel auf dem nächsten Stab links nach oben bzw. zur Mitte schieben und auf dem Stab rechts außen alle Kugeln in ihre Ausgangsstellung (die Ziffer 0 repräsentierend) zurückversetzen. Danach können wir den Restbetrag hinzufügen: Drei haben wir bereits addiert, es sollten aber vier sein; deswegen zählen wir noch eins zu rechten Position dazu, indem wir dort eine Kugel nach oben bzw. Richtung Mittelleiste verschieben.

Mit seiner Speicherfunktion besitzen der Abakus, der übrigens hier und da auch heute noch im Einsatz ist, und verwandte Rechenhilfen wie der Rechenschieber ein fundamentales Merkmal eines Computers. Allerdings sind sie nicht in der Lage, die Speicherinhalte selbstständig zu verändern, also automatisch ganze Berechnungen mit mehreren Rechenschritten durchzuführen. Die Einführung der Automatik – dieser Begriff ist ja in der zweiten Hälfte des Worts „Informatik“ enthalten – erfolgte erst später.

Wenn es beim Rechnen knackt, knirscht und surrt

Die ersten richtigen Rechenmaschinen wurden im 17. Jahrhundert entwickelt. Ihnen konnte man regelrecht beim Rechnen zuhören, denn ihr Inneres ähnelte einem mechanischen Uhrwerk. Ein in der Tat zutreffender Vergleich, schließlich handelt es sich bei einer Uhr um nichts anderes als um eine Zählmaschine: Jede Sekunde zählt sie eins zur aktuellen, gespeicherten Zeit hinzu, die sich selbst aus drei Zahlen – repräsentiert durch Sekunden-, Minuten- und Stundenzähler – zusammensetzt. Die drei Zähler sind miteinander gekoppelt, d. h., nach 60 s erhöht sich der Minutenzähler um eins und nach 60 min der Stundenzähler, während gleichzeitig der Sekunden- bzw. der Minutenzähler zurückgesetzt wird. Eine Uhr beherrscht demnach genau eine Rechenoperation, die Addition von eins. Das Gleiche gilt für einen Handzähler, ein per Tastendruck gesteuertes Zählwerk, mit dem sich Personen, Runden, Versuche usw. zählen lassen. Abgesehen vom verwendeten Zahlensystem besteht der einzige Unterschied zwischen den beiden Geräten darin, dass die Rechenoperation bei letzterem manuell über eine Taste ausgelöst wird, während ersteres autonom und zeitlich äußerst präzisiert hochzählt (Abb. 1.6).

Die Grundidee, Rechenschritte auf mechanische Vorgänge abzubilden, lässt sich am leichtesten am Beispiel derjenigen Grundrechenart erläutern und nachvollziehen, die wir als erste in der Schule erlernen: der Addition. Ein Taschenrechner aus dem 19. Jahrhundert, die Addiermaschine des Amerikaners Archibald M. Stephenson, soll uns als Anschauungsobjekt dienen; er war darauf ausgerichtet, seinen Benutzer beim stupiden Aufsummieren von Zahlenkolonnen zu entlasten und somit die Fehlerquote zu reduzieren.

Wie in Abb. 1.7 ersichtlich, besitzt diese Metallkonstruktion zwei mit Zacken versehene Drehscheiben, auf denen Zahlen eingraviert sind: rechts die Ziffern 0 bis 9, links die Werte 0 bis 19. Analog dem Abakus steht die rechte Scheibe für die Einer und die linke Scheibe für die Zehner; die größte

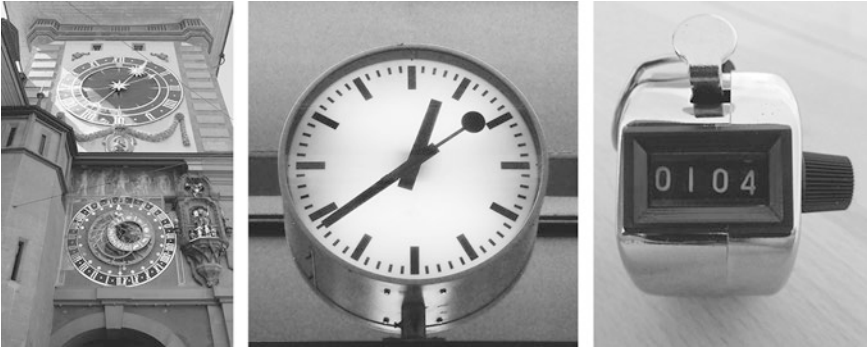


Abb. 1.6 Relikte aus der Zeit des mechanischen Rechnens: Turmuhr und Astrolabium der Zytglogge in Bern (links), eine Bahnhofsuhr (Mitte) und ein Handzähler mit Zähltaste und seitlichem Rückstellknopf (rechts). (© Eckart Zitzler)



Abb. 1.7 Mechanischer Taschenrechner von A. M. Stephenson, links mit montierter und rechts ohne Deckplatte. (© Reinhard Atzbach)

darstellbare Zahl ist demnach die 199. Eine Deckplatte mit zwei Kerben stellt sicher, dass immer nur die jeweils oberste Zahl einer jeden Scheibe sichtbar ist. Darüber hinaus sind Löcher in die Scheiben eingestanz, die unmittelbar bei den Zahlen positioniert sind (links nur bei der 0). Wie funktioniert nun dieser Rechner?

Bsp. 1.2 gibt Ihnen einen Einblick in den zugrunde liegenden Mechanismus. Einfach, ja fast schon trivial ist er; die Hauptschwierigkeit liegt in der Handhabung des Übertrags, also beim Zusammenspiel der Scheiben. Vielleicht sind Sie jetzt etwas enttäuscht: Das soll ein mechanischer Rechner sein? Im Wesentlichen ja, doch seien Sie beruhigt: Die Realisierung anderer Rechenoperationen ist ungleich komplizierter und nicht in ein paar Worten erklärt. Dennoch, auch moderne Rechner bauen auf simplen Grundprinzipien auf und solche verkörpern ja eine mechanische Uhr, ein Handzähler und Stephenson's Addiermaschine.

Beispiel 1.2: Funktionsweise des Stephenson-Addierers

Nehmen wir an, wir wollten die Summe aus 3, 5 und 2 berechnen. Als Erstes richten wir die Drehscheiben so aus, dass in beiden Kerben die 0 zu sehen ist – der Speicher ist somit zurückgesetzt und das Ergebnis einer eventuellen vorherigen Berechnung gelöscht. Anschließend bewegen wir die rechte Scheibe um drei Positionen, d. h., wir geben die 3 ein. Dazu können wir einen Stift nehmen, ihn in das Loch stecken, das mit der 3 auf der Deckplatte korrespondiert, und auf diese Weise die Scheibe im Uhrzeigersinn so weit drehen, bis sich der Stift direkt unterhalb der Kerbe befindet. Das Ganze wiederholen wir mit der 5: Stift in das Loch bei der 5 stecken, weiterdrehen bis zur Kerbe, und schon ist die erste Addition vollzogen. In der rechten Kerbe ist als Zwischenresultat die 8 zu sehen, da die Scheibe insgesamt um acht Positionen bewegt wurde. Für die letzte Addition drehen wir die Scheibe erneut, diesmal um zwei weitere Positionen. Es kommt wieder die 0 zum Vorschein, obwohl wir an sich bei der 10 angelangt wären. Der entstandene Übertrag müsste auf die linke Drehscheibe übertragen werden, und genau das passiert auch. Die an der rechten Drehscheibe angebrachte Zacke greift in die Zacken der linken Drehscheibe ein und schiebt diese um eine Position weiter, sobald der Übergang von der 9 zur 0 vollzogen wird.

Die in Abb. 1.8 gezeigte Pascaline, eine der ersten mechanischen Rechenmaschinen überhaupt und die erste, von der mehrere Exemplare angefertigt wurden, arbeitete ebenfalls nach diesem Schema. Wie Stephenson's Addierer besaß sie mehrere nebeneinander angebrachte Metallwählscheiben, die über einen ausgeklügelten Mechanismus ineinandergriffen und so Überträge weiterreichen konnten. Die Ergebnisse waren allerdings nicht direkt von den Scheiben ablesbar, sondern wurden darüber in separaten Kästchen angezeigt. Um diesen Bildschirm-Vorläufer anzusteuern, bedurfte es einer entsprechenden Übersetzung mittels Zahnrädchen. Die Pascaline war hauptsächlich auf das Summieren ausgerichtet, doch über einen Trick konnte man auch subtrahieren. Wie Sie eventuell schon

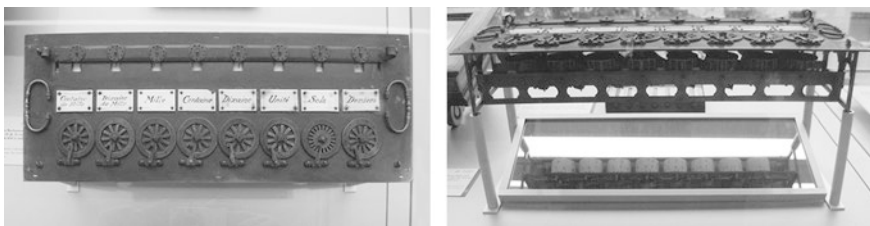


Abb. 1.8 Diese Fotografien einer Pascaline zeigen einmal die Oberseite (links) und einmal die Vorder- und Unterseite mit geöffnetem Bodendeckel (rechts). (© Arithmeum, Rheinische Friedrich-Wilhelms-Universität Bonn)

vermutet haben, war es Blaise Pascal, der französische Universalgelehrte, der diesen für damalige Verhältnisse wundersamen Kasten 1642 erfunden hat; 350 Jahre später wurde eine bekannte Programmiersprache nach ihm benannt.

Das war also die Einführung der Automatik. Der menschliche Benutzer musste zwar immer noch Kraft aufwenden, um die Zahnräder zu bewegen – in diesem Sinne funktionierten die Maschinen nicht vollautomatisch wie z. B. eine Schiebetür mit Bewegungssensor –, doch die einzelnen Denkschritte zwischendurch fielen weg und wurden durch mechanische Vorgänge ersetzt. Damit war das automatisierte Rechnen ins Leben gerufen und man kam dem modernen Computer einen gewaltigen Schritt näher. Doch die Programmierbarkeit fehlte noch, d. h. die Möglichkeit, die Berechnungsvorschrift zu variieren und flexibel einstellen zu können, welche Rechenschritte in welcher Reihenfolge automatisch ausgeführt werden sollen.

Die Geburt des modernen Computers

Wenn wir uns heutzutage eines Computers bedienen, ist es für uns selbstverständlich, ihn für verschiedenste Zwecke einzusetzen. Wir schreiben Texte, versenden Nachrichten, hören Musik, schauen Filme, bearbeiten Fotos usw. Wenn uns eine bestimmte Anwendung nicht gefällt, installieren wir das Produkt eines anderen Anbieters. Ist unsere Anwendung veraltet, so kann sie mittels eines Updates aktualisiert werden. Und das alles auf ein und demselben Gerät und ohne dieses verändern zu müssen. Weil unser Computer eben programmierbar ist und mit beliebigen Programmen gefüttert werden kann.

Die Grundlage für diese Flexibilität wurde im frühen 19. Jahrhundert gelegt, als Charles Babbage das Konzept für einen universellen, programmierbaren Rechner vorstellte und Ada Lovelace ein dazugehöriges Programm verfasste (Abb. 1.9). Babbage, ein britischer Mathematiker, begann erst im Alter von 30 Jahren, sich mit dem automatisierten Rechnen zu beschäftigen – dafür dann umso intensiver. Einen Großteil seines Lebens widmete er diesem Thema und tatsächlich gelang ihm 1834, als er sich die Analytical Engine ausdachte, etwas, das aus heutiger Sicht als großer Wurf und Pionierleistung angesehen werden muss – auch wenn diese Maschine nie Wirklichkeit wurde (Abb. 1.10). Man könnte sagen, Babbage war der Erste, der einen vollwertigen Computer beschrieb, und in der Tat spiegeln seine Pläne grundlegende Elemente eines modernen Computers wider: 1000 Zahlen hätte die Analytical Engine speichern können und das Rechenwerk – er nannte es „Mühle“ – wäre in der Lage gewesen, alle vier Grundrechenarten auszuführen; darüber hinaus wären weitere Aktionen

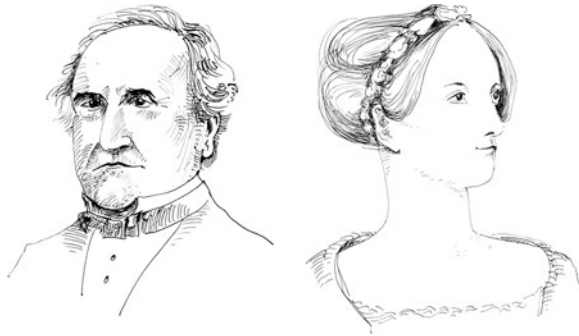


Abb. 1.9 Charles Babbage, der Erfinder der Analytical Engine, und Ada Lovelace, die das erste namhafte Programm für einen Computer entwickelte. (© Camille Büsser)

wie Vergleiche von Zahlen möglich gewesen. Gegenüber der Pascaline, die gerade mal eine Zahl speichern und Additionen sowie auf Umwegen auch Subtraktionen durchführen konnte, stellte das einen riesigen Fortschritt dar.

Wie die Idee der Programmierbarkeit effektiv in Babbages Maschine hätte umgesetzt werden sollen, ist hingegen etwas schwerer zu erklären; dafür ist seine Konstruktion zu komplex, schließlich hätte sie von einer Dampfmaschine bedient werden müssen – so viele Zahnräder hätten da ineinandergegriffen, dass ein Mensch den Kraftaufwand gar nicht hätte bewerkstelligen können. Eine grobe Vorstellung können Sie bekommen, wenn Sie an die Gangschaltung eines Fahrrads oder das Getriebe eines Autos denken: Hier ist die Übersetzung zwischen verschiedenen Zahnrädern flexibel einstellbar und wird beispielsweise über einen Schalthebel „programmiert“. Fakt ist auf jeden Fall, dass Ada Countess of Lovelace, eine 24 Jahre jüngere Zeitgenossin Babbages und Tochter des bekannten englischen Dichters Lord Byron, den Nutzen der Analytical Engine erkannte und 1842 einen schriftlichen Plan zur Berechnung von Bernoulli-Zahlen mithilfe dieser Maschine vorlegte. Aus diesem Grund wird sie als erste Programmiererin in der Geschichte angesehen.

Babbages und Lovelaces Ideen waren den technischen Möglichkeiten ihrer Zeit allerdings so weit voraus, dass es noch weitere 100 Jahre dauern sollte, bis in den 1940er-Jahren die ersten funktionstüchtigen Computer gebaut werden konnten; diese arbeiteten nicht mechanisch, sondern rechneten mittels Strom. Zuerst gelang das 1941 Konrad Zuse und Helmut Schreyer in Deutschland. Der erste rein elektronische Computer, der ENIAC (Abb. 1.11), wurde von J. Presper Eckert und John W. Mauchly

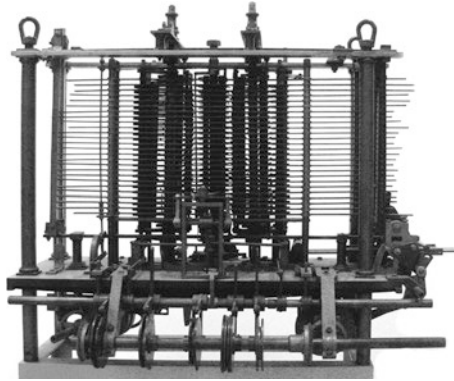


Abb. 1.10 Ein Versuchsmodell der Analytical Engine. (© Arithmeum, Rheinische Friedrich-Wilhelms-Universität Bonn)

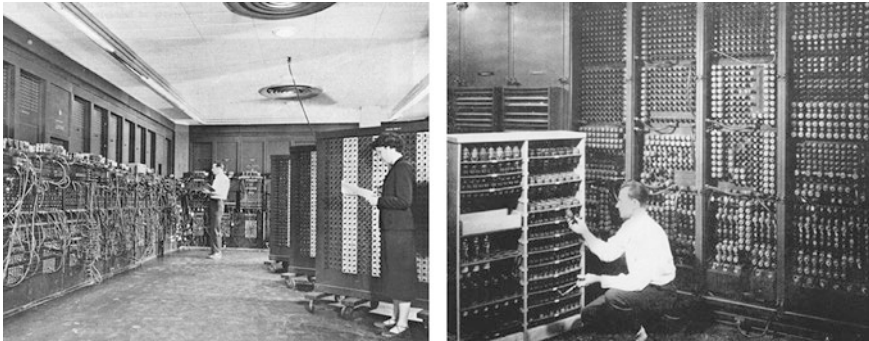


Abb. 1.11 Der ENIAC – die Abkürzung steht für *Electronic Numerical Integrator and Computer*. (© U. S. Army („U.S. Army Photo“))

in den USA entwickelt und schließlich 1946 der Öffentlichkeit vorgestellt. Diese Dinosaurier des Computerzeitalters waren riesig, schwer und gefräßig: Der ENIAC beispielsweise beanspruchte eine Grundfläche von 10×17 m, wog 27 t und hatte einen so hohen Stromverbrauch, dass Gerüchten zufolge die Lichter in Philadelphia flackerten, wenn der ENIAC eingeschaltet wurde. Der Durchbruch wurde eingeläutet mit der Erfindung des Transistors im Jahr 1947, einem kleinen, durch Strom steuerbaren Schalter, durch dessen Verwendung elektronische Rechner billiger, schneller, zuverlässiger und kleiner wurden. Es sollte noch eine Weile dauern, bis die Technologie ausgereift war, doch mit der Einführung des Mikroprozessors – des Kernstücks eines Computers – in den 1970er-Jahren nahm eine rasante Entwicklung ihren Lauf. Zunächst eroberte in den 1980er-Jahren der Personal Computer in verschiedensten Ausführungen Privathaushalte und

Büros, in den 1990ern folgte die weltweite Vernetzung durch das Internet und zu Beginn des 21. Jahrhunderts kamen kleine, mobile Computer-Endgeräte hinzu. Welchen gewaltigen Fortschritt all das bedeutet, sieht man heute.

Drei Wesensmerkmale eines Computers

Letztlich war es also immer wieder die Technologie – von der Zahnradmechanik bis zur Halbleitertechnik –, die der lang gereiften Computeridee zum Erfolg verhalf. Heutzutage ist für uns ein Computer ein elektronisches Gerät, das, wie wir gesehen haben, in verschiedensten Formen in Erscheinung tritt und dessen Anwendungsspektrum weit über das pure Rechnen hinausgeht. Doch das Konzept eines Computers ist nicht an eine bestimmte Technologie gebunden. Charles Babbages Pläne sahen eine mechanische Umsetzung vor, der erste funktionsfähige Computer – die Z3 von Konrad Zuse – arbeitete elektromechanisch, der ENIAC funktionierte rein elektronisch und nutzte vor allem Elektronenröhren, und heutige Computer basieren auf der Halbleitertechnologie. Selbst mit Wasser und DNA-Molekülen wurde schon gerechnet, und welche Rolle Quantencomputer – also Computer, die auf quantenmechanischen Prinzipien beruhen – spielen werden, wird die Zukunft zeigen.

Ob dieser Vielfalt an Computerausprägungen kann einem schwindlig werden. Umso mehr drängt sich die Ausgangsfrage wieder auf: Was ist eigentlich ein Computer, was macht sein Ureigenstes aus? Wie wir gesehen haben, lässt sie sich weder über das Erscheinungsbild noch über die Technologie beantworten. Doch drei Stichworte sind bereits gefallen, nämlich Speicher, Automatik und Programmierbarkeit. Rekapitulieren wir:

- Die Vorläufer der Rechenmaschinen sind geschickt konzipierte Gedächtnisstützen wie der Abakus, mit denen sich Berechnungen einfacher und zuverlässiger durchführen lassen; der Speicher, den diese Hilfsmittel verkörpern, wird durch Menschenhand bedient und verändert.
- Demgegenüber kann eine echte Rechenmaschine selbstständig Zwischenergebnisse produzieren und ablegen. Zum Speicher kommt ein z. B. mechanisches Rechenwerk hinzu, das die Speicherinhalte lesen und manipulieren kann. Das Rechenwerk automatisiert Berechnungsschritte, die sonst von Hand ausgeführt werden; die Schritte und ihre Abfolge sind dabei fest vorgegeben.

- Der moderne Computer stellt den dritten Meilenstein in dieser Entwicklung dar, indem er das Konzept der Programmierbarkeit realisiert. Bei ihm ist das Rechenwerk flexibel einstellbar, d. h., die auszuführenden Rechenoperationen und ihre Reihenfolge lassen sich – innerhalb eines vorgegebenen Rahmens – frei bestimmen. Damit ist ein Computer nicht auf spezielle Berechnungen festgelegt, sondern universell einsetzbar.

Somit haben wir die zentralen Prinzipien identifiziert, auch wenn eine fassbare Definition noch aussteht. Die wird im Folgenden nachgeliefert, allerdings nehmen wir dabei einen leicht veränderten Blickwinkel ein, indem wir unsere Ausgangsfrage umformulieren: Was bliebe übrig, wenn wir einen Computer auf sein Minimum reduzieren wollten, ohne das Charakteristische seines Wesens zu verlieren?

Des Computers Kern: die Turing-Maschine

Mit dieser Frage hat sich der Brite Alan Turing (Abb. 1.12) bereits beschäftigt, bevor die ersten funktionsfähigen Computer überhaupt gebaut waren – wenn auch eher indirekt. Er wollte nämlich wissen, was ein Mensch alles berechnen kann und was nicht. Zu diesem Zweck hat er sich ein einfaches Gerät ausgedacht, mit dem er den Vorgang des menschlichen Rechnens an sich nachbilden wollte. Von allen nicht essenziellen Details hat er dabei abstrahiert, weshalb wir auch so etwas wie eine Tastatur oder einen Bildschirm vergeblich suchen werden.



Abb. 1.12 Alan Turing, der mit seinen theoretischen Arbeiten die Informatik grundlegend geprägt hat. (© Camille Büsser)

Ein Maschinenkonstrukt, um das menschliche Rechnen nachzubilden

Die von ihm im Jahr 1936 vorgeschlagene Rechenmaschine gilt heute als Grundmodell eines Computers und wird zu seinen Ehren als Turing-Maschine bezeichnet. Auf den ersten Blick – werfen sie mal einen Blick auf Abb. 1.13 – sieht sie allerdings eher skurril aus und scheint wenig mit einem heutigen Computer gemein zu haben.

Der Grund ist, dass sie an sich gar nicht dafür gedacht war, in die Realität umgesetzt zu werden. Turing ging es um das Wesen eines Computers, um die grundlegenden Prinzipien des automatisierten Rechnens, aber es war nicht seine Absicht, eine praktisch einsetzbare Maschine zu entwerfen. In diesem Sinne ist sein Computerkonzept ein Gedankenexperiment, ein Gebilde der Vorstellung. Entscheidend für ihn war, dass seine Maschine, so wie er sie sich auf dem Papier ausgedacht hatte, prinzipiell gebaut werden konnte – insofern hatte er eher den in Abb. 1.14 skizzierten abstrakten Bauplan im Kopf als eine konkrete Umsetzung.

Turings Minimalcomputer besteht aus einem Band, einer Vorrichtung zum Lesen, Beschriften und Bewegen des Bandes – dem sogenannten Lese- und Schreibkopf – sowie einer Steuereinheit, quasi dem Gehirn der Maschine, das die Berechnungen steuert. Das Ganze erinnert stark und vielleicht nicht von ungefähr an ein Tonbandgerät – den analogen Vorläufer der digitalen Tonaufzeichnung –, das ebenfalls in den 1930er-Jahren entwickelt wurde. Turings Assoziation war jedoch die des rechnenden Menschen: Das Band versinnbildlicht das karierte Notizbuch, der Lese- und Schreibkopf

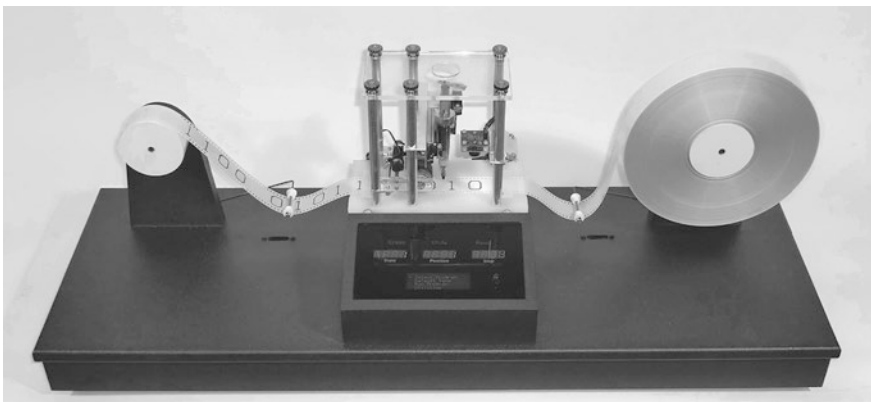


Abb. 1.13 Eine Turing-Maschine in Aktion; dieses Exemplar wurde von Mike Davey konstruiert und gebaut. (© Michael Davey)

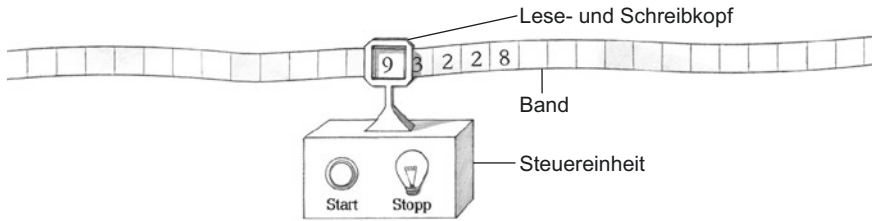


Abb. 1.14 Schematische Zeichnung einer Turing-Maschine. (© Susanne Staubli, Eckart Zitzler)

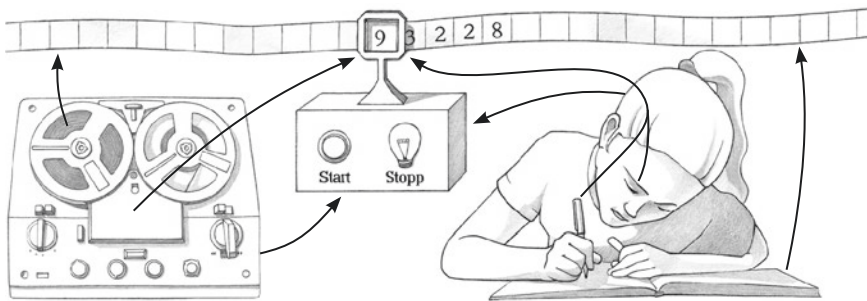


Abb. 1.15 Tonbandgerät und der rechnende Mensch im Vergleich zur Turing-Maschine. (© Susanne Staubli, Eckart Zitzler)

steht sowohl für die Hand mit dem Stift als auch für das Auge, und die Steuereinheit entspricht dem logisch denkenden Kopf, der die Rechnungen ausführt (Abb. 1.15).

Rechnen als Textbearbeitung

Turings altmodisch anmutende Maschine wirkt jedoch nicht nur in Bezug auf ihr Aussehen eigenartig, sondern auch hinsichtlich ihrer Arbeitsweise. Denn können Sie sich vorstellen, dass eine Maschine, die nichts anderes macht, als einen vorgegebenen Text durch schrittweises Ersetzen einzelner Zeichen in einen neuen Text umzuwandeln, tatsächlich rechnen kann? So absurd das auch erscheinen mag – wir werden nachher anhand zweier einfacher Rechenbeispiele den Beweis führen. Doch vorher schauen wir uns an, wie eine Turing-Maschine im Detail funktioniert.

Da gibt es zunächst einmal das Band. Aus welchem Material es besteht, ob aus Papier, Folie oder Magnetband, soll uns hier nicht weiter kümmern. Entscheidend ist, dass sich mit ihm Zeichen, seien es Ziffern,

Buchstaben oder sonstige Symbole, zwischenspeichern lassen und es damit als Gedächtnis der Maschine fungiert. Analog zu dem Karomuster auf dem Notizpapier ist das Band in einzelne Zellen aufgeteilt, von denen jede ein Zeichen aufnehmen kann. Der Einfachheit halber betrachten wir hier nur die Zahlen von 0 bis 9 und das Leerzeichen als infrage kommende Zeichen. Folglich repräsentiert das Band zu jedem Zeitpunkt eine riesige Zeichenkette (den oben erwähnten Text), die am Anfang aus lauter Leerzeichen und der Eingabe, z. B. einer Zahl in Form einer Ziffernfolge, besteht.

Mit dem Lese- und Schreibkopf kann der Speicher, also das Band, ausgelesen und manipuliert werden. Er kann das Zeichen an der Bandposition, wo er gerade steht, einlesen sowie ggf. ausradieren und durch ein anderes Zeichen ersetzen. Zudem ist der Lese- und Schreibkopf in der Lage, das Band jeweils um eine Zelle nach links oder rechts zu verschieben. Auch hier gibt es wieder verschiedene Möglichkeiten, so eine Lese-/Schreibvorrichtung zu realisieren: Bestände das Band aus Folie, so könnte beispielsweise ein wasserlöslicher Folienstift zum Beschriften, ein feuchtes Tuch zum Ausradieren und eine Kamera zum Einlesen dienen.

Schließlich ist da noch die Steuereinheit, im Bauplan als Kasten mit verborgenem Innenleben skizziert. Wie der Name sagt, steuert sie die Berechnung; dazu ist sie direkt mit dem Lese- und Schreibkopf verbunden. Sie macht nichts anderes, als den folgenden, aus drei Schritten bestehenden Vorgang immerzu zu wiederholen: erstens liest sie das aktuelle Zeichen unter dem Lese- und Schreibkopf; zweitens überschreibt sie es mit einem neuen Zeichen; drittens bewegt sie das Band um eine Zelle oder stoppt die Berechnung. Diese drei Schritte stellen zusammen die einzige Art von Rechenoperation – wir könnten auch sagen: Befehl – dar, die die Turing-Maschine kennt: das Zeichen unter dem Lese- und Schreibkopf durch ein anderes Zeichen ersetzen und anschließend das Band verschieben. Wie gesagt, Rechnen heißt in diesem Fall Textbearbeitung und tatsächlich lassen sich mit dieser Methode beliebig komplizierte Berechnungen durchführen.

Eine Rechenoperation wird über eine Tabelle beschrieben

Woher weiß die Steuereinheit nun, welches Zeichen geschrieben und ob das Band nach links oder nach rechts bewegt werden soll? Dazu muss die Maschine entsprechend programmiert werden, und zwar mittels einer Tabelle. Diese Tabelle legt für jedes gelesene Zeichen fest, welches Zeichen zu schreiben und in welche Richtung das Band zu verschieben ist. Abb. 1.16

Die dritte Rechenoperation ist identisch zu den vorherigen zwei und führt zum Zwischenergebnis 4000.

Das anschließende Lesen der 4 veranlasst die Steuereinheit, die 4 durch eine 5 zu ersetzen und die Berechnung zu stoppen – wie es durch die Tabelle vorgegeben ist. Das Ende der Berechnung könnte dabei durch das Aufleuchten eines Lämpchens signalisiert werden. Am Schluss steht die Zahl 5000 auf dem Band.

Natürlich stellt sich die Frage, wie die auf einer Tabelle festgehaltenen Informationen – also das Programm – in die Steuereinheit eingepeist werden können. Benutzen wir wie in Abb. 1.16 eine Lochkarte, um die Tabelle zu beschreiben, so kann die Steuereinheit mit einer entsprechenden Lesevorrichtung ausgestattet und das Programm über dieses Zusatzgerät geladen werden. So ein Lochkartenleser gehörte lange Zeit zur Standardausrüstung eines Rechenzentrums, denn Lochkarten waren die mobilen Speichermedien der Computer-Steinzeit (Abb. 1.18).

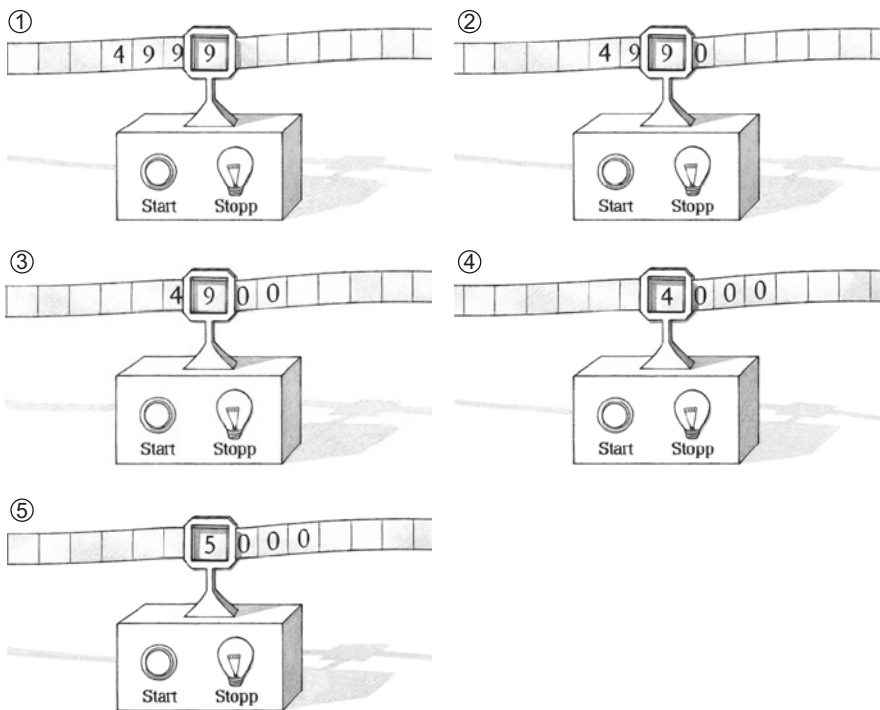


Abb. 1.17 Rechenbeispiel, wie eine Turing-Maschine zu einer vorgegebenen Zahl eins hinzuzählt. (© Susanne Staubli, Eckart Zitzler)

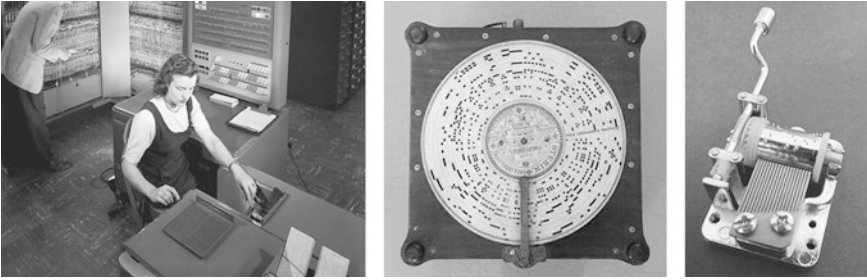


Abb. 1.18 Eine Frau bedient 1957 einen Lochkartenleser (links); ein mechanisches Musikinstrument (um 1900), das über eine mit Löchern versehene Pappscheibe „programmiert“ wird und somit verschiedene Melodien abspielen kann (Mitte); eine Spieluhr, bei der eine spezifische Melodie auf einer Walze festgeschrieben ist (rechts). (© NASA (links), Eckart Zitzler (Mitte, mit freundlicher Genehmigung des Museums ENTER Solothurn), Eckart Zitzler (rechts))

Das Prinzip, einen mit eingestanzten Löchern versehenen Kartonstreifen als Speichermedium zu verwenden, war bis in die 1980er-Jahre weit verbreitet, und selbst heute findet es im weiteren Sinne immer noch Verwendung: Spieluhren beispielsweise geben Melodien wieder, die auf Lochscheiben, Lochstreifen oder auf mit Stiften versehenen Metallwalzen festgehalten sind.

Mehrere Tabellen können mehrere Rechenoperationen ausführen

Der in Bsp. 1.3 beschriebene Rechenvorgang entspricht haargenau unserem Vorgehen, wenn wir schriftlich eine 1 zu einer Zahl hinzuaddieren. Wir gehen ziffernweise von rechts nach links, ersetzen die aktuelle Ziffer um die nächsthöhere und hören auf, wenn es keinen Übertrag mehr gibt. Die Tabelle verkörpert so gesehen das Regelwissen zur positionsweisen Addition von 1, welches wir als Kinder zu Beginn unserer Schulzeit lernen.

Doch sobald wir nur ein wenig kompliziertere Rechnungen, z. B. die Addition von 2, durchführen lassen wollen, reicht eine einzige Tabelle nicht mehr aus. Daher bietet die Turing-Maschine die Möglichkeit, mehrere Tabellen zu benutzen, wobei sie natürlich wissen muss, wann welche zu verwenden ist. Zu diesem Zweck werden die Tabellen kurzerhand nummeriert und jeweils um eine neue Spaltengruppe erweitert. Für jedes Zeichen, welches der Lese- und Schreibkopf möglicherweise einlesen kann, werden nicht nur das zu schreibende Zeichen und die Bandbewegung angegeben, sondern

auch die Nummer der Tabelle, die bei der nächsten Rechenoperation zu verwenden ist (siehe Abb. 1.19). Darüber hinaus muss festgelegt werden, mit welcher Tabelle die Programmausführung beginnen soll; per Konvention ist dies stets die Tabelle mit der Nummer 1.

Bsp. 1.4 zeigt auf, wie eine Programmausführung mit zwei Tabellen aussehen kann. Der Ablauf ähnelt demjenigen aus Bsp. 1.3, doch diesmal werden zwei hinzugezählt. Wollte man zwei beliebige Zahlen addieren, bräuchte man eine wesentlich größere Anzahl an Tabellen, ganz zu schweigen von komplexeren Berechnungen.

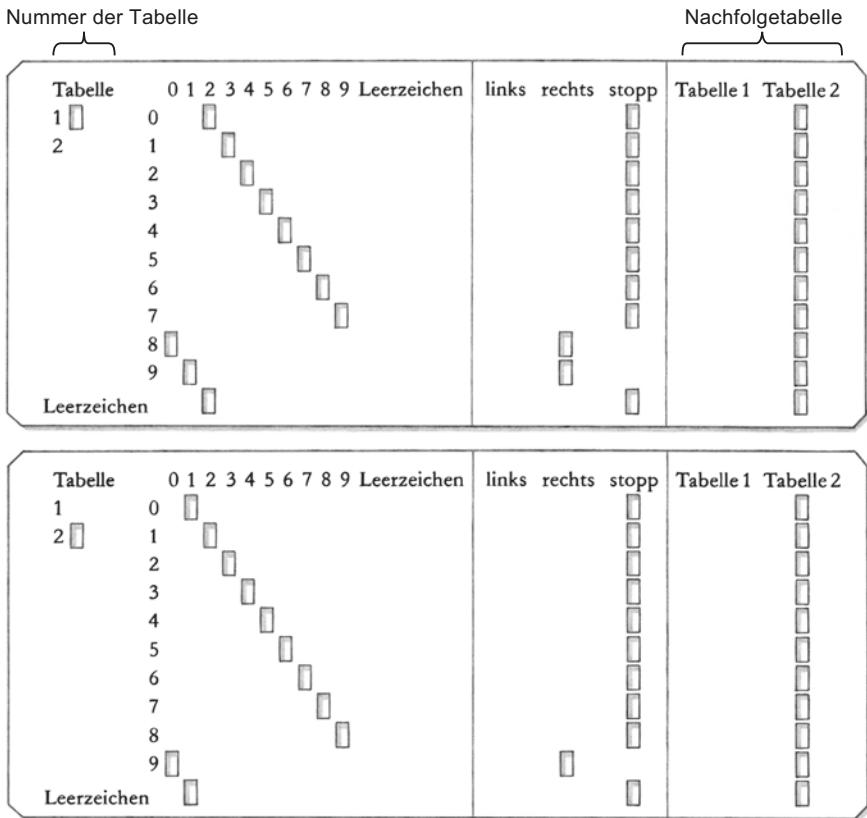


Abb. 1.19 Ein Programm mit zwei Tabellen zur Addition von 2. (© Susanne Staubli, Eckart Zitzler)

Beispiel 1.4: Zwei hinzuzählen mit einer Turing-Maschine

Programmieren wir die Turing-Maschine nun mit den Tabellen aus Abb. 1.19 und starten die Berechnung in derselben Ausgangssituation wie vorher (siehe Abb. 1.20).

Im ersten Schritt wird eine 9 gelesen, und gemäß der Konvention legt Tab. 1 die nächsten Aktionen fest. Der Tabelleneintrag zur Ziffer 9 besagt, dass die 9 durch eine 1 zu ersetzen, anschließend das Band um eine Zelle nach rechts zu verschieben und bei der nächsten Rechenoperation Tab. 2 zu verwenden ist. Wenn diese Aktion durchgeführt wurde, steht die Zahl 4991 auf dem Band.

In der nachfolgenden Rechenoperation wird wiederum eine 9 eingelesen, aber diesmal wird Tab. 2 konsultiert, um die Aktionen zu bestimmen. Diese lauten: die 9 durch eine 0 ersetzen, das Band nach rechts verschieben und auch bei der nächsten Operation Tab. 2 benutzen. Der Zwischenstand danach ist 4901.

Die nächsten zwei Rechenoperationen sind identisch zur dritten und vierten Operation im vorherigen Rechenbeispiel. Die letzte 9 wird durch eine 0 ersetzt und die 4 durch eine 5, beide Male gemäß Tab. 2. Wenn die Maschine schließlich stoppt, steht 5001 auf dem Band. Das Programm mit den zwei Tabellen hat eine Addition mit 2 bewirkt, wiederum ganz in Anlehnung an das Vorgehen beim handschriftlichen Addieren auf Papier.

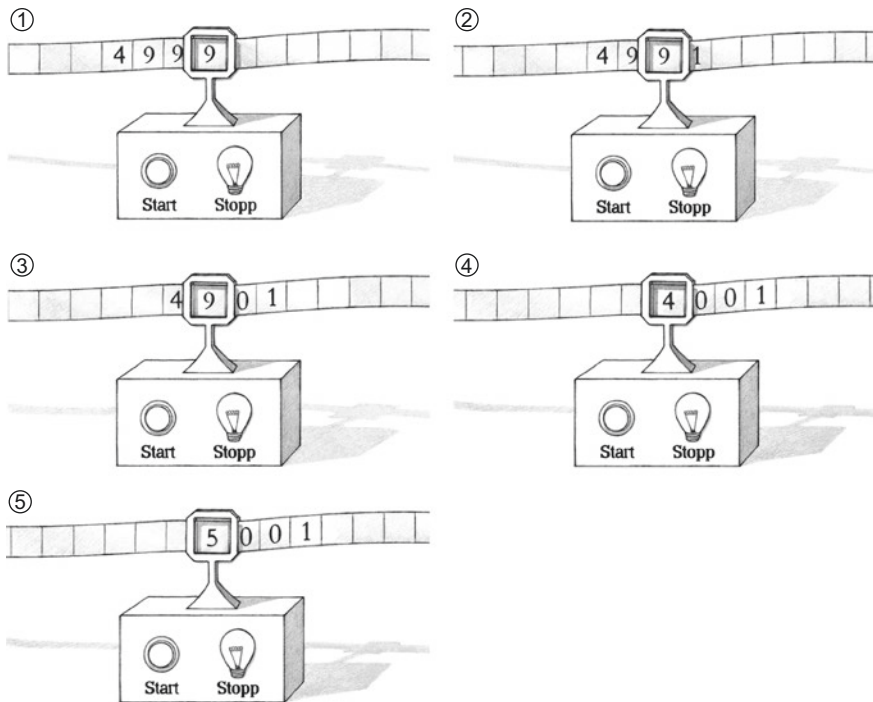


Abb. 1.20 Erweitertes Rechenbeispiel: Wie eine Turing-Maschine zu einer vorgegebenen Zahl zwei hinzuzählt. (© Susanne Staubli, Eckart Zitzler)

Bei einer Turing-Maschine ist ein Programm also über einen Satz an Tabellen beschrieben. Im Allgemeinen legt ein Programm eine Menge von konkreten Rechenoperationen oder Befehlen fest (die Tabellen) und darüber hinaus die Reihenfolge, in welcher diese auszuführen sind (bestimmt durch die Angabe der Nachfolgetabellen). Welche Befehle effektiv zur Ausführung kommen, hängt von den zu verarbeitenden Daten ab; im vorherigen Beispiel bestimmt die Anzahl der Ziffern auf dem Band, wie oft die durch Tab. 2 beschriebene Rechenoperation ausgeführt wird. Somit führt eine spezifische Programmausführung stets zu einer spezifischen Sequenz von Rechenoperationen, d. h., andere Eingabedaten können in einer anderen Befehlsabfolge resultieren.

Sinn und Unsinn von Turings Minimalcomputer

So – damit ist eigentlich alles Wesentliche zur Arbeitsweise einer Turing-Maschine gesagt. Gratulation! Sie kennen jetzt den einfachsten Computer der Welt und wissen, wie er funktioniert. Er kann Daten speichern, diese automatisch verarbeiten und ist darüber hinaus programmierbar, d. h., die Art und Weise, wie die Daten verarbeitet werden, ist flexibel einstellbar. Somit verfügt er über alle charakteristischen Merkmale, die wir einer vollwertigen Rechenmaschine zuschreiben. Obwohl er nur über einen einzigen, simplen Rechenmechanismus verfügt – ein Zeichen durch ein anderes ersetzen, das Band verschieben und ggf. die Tabelle wechseln –, steht er bzgl. seines Rechenvermögens einem beliebigen heutigen Computer in nichts nach (die Rechengeschwindigkeit ist eine andere Geschichte, aber davon ist hier nicht die Rede). Dabei umfasst Rechnen nicht nur rein mathematische Aufgaben, z. B. die Berechnung von Integralen oder die Lösung von Gleichungssystemen, sondern auch ganz praktische Probleme wie die Steuerung eines Mars-Roboters oder die Vorhersage von Wirbelstürmen (Abb. 1.21). Alles, was sich grundsätzlich im intuitiven Sinne berechnen lässt, kann auch von einer Turing-Maschine berechnet werden. Diese Aussage, die auch als Churchsche These bekannt ist, lässt sich nicht beweisen, doch die Informatikergilde ist sich einig darin, dass diese Vermutung zutrifft.

Wenn wir nun zurück zu unserer Ausgangsfrage kommen, nämlich „Was ist ein Computer?“, so könnte man salopp sagen: Ein Computer ist ein Gerät, das alles kann, was eine Turing-Maschine kann. Wer es gerne genauer hat, der kann auf folgende Definition eines Fachbuchs zur Didaktik der Informatik zurückgreifen (Schubert und Schwill 2011, S. 4):

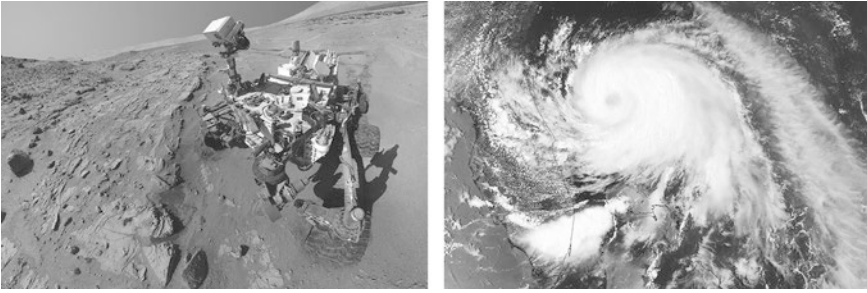


Abb. 1.21 Robotersteuerung und Wettersimulation – zwei exemplarische Anwendungen, bei denen die Berechnungen von einer Turing-Maschine durchgeführt werden könnten; die Fotos zeigen ein Selbstporträt des Mars-Roboters Curiosity (links) und die Satellitenaufnahme eines Hurrikans (rechts). (© NASA/JPL-Caltech/MSSS (links), NASA Goddard MODIS Rapid Response Team (rechts))

Ein **Computer** [...] ist ein **universell** einsetzbares Gerät zur automatischen Verarbeitung von Daten (einschl. deren Speicherung).

Auch in dieser Beschreibung finden wir die drei Stichworte Speicher, Automatik und Programmierbarkeit („universell einsetzbar“) wieder.

Schön und gut, aber was ist letztlich der Nutzen der Turing-Maschine? Die zwei Rechenbeispiele haben bereits aufgezeigt, wie aufwändig die Programmierung einer Turing-Maschine selbst für elementarste Aufgaben ist. Keine Informatikerin, kein Informatiker würde sich das antun wollen, es sei denn, im Studium zu Übungszwecken. Dann ist da noch die Frage, wie eine Turing-Maschine eigentlich bedient werden soll; denn von Ein- und Ausgabevorrichtungen war bislang keine Rede. Grundsätzlich könnte man schon Zusatzgeräte bauen, mit denen sich Zeichen auf das Band schreiben und Teile des Bandes auf einem Bildschirm anzeigen ließen. Aber so richtig praktisch würde das Ganze dann immer noch nicht, man denke nur an Foto- oder Videobearbeitung. Machen wir uns also nichts vor: Für den praktischen Einsatz ist eine Turing-Maschine kaum geeignet, dafür ist sie viel zu langsam und zu schwerfällig in der Bedienung – auch wenn es, wie wir gesehen haben, Hobbybastler gibt, die sie für Demonstrationszwecke tatsächlich bauen.

Der Nutzen ist eher theoretischer Natur. Weil sie so schön einfach ist und von allen unnötigen Details abstrahiert, eignet sie sich bestens, um das Wesen eines Computers zu illustrieren und zu studieren – die Turing-Maschine als des Pudels Kern, als Essenz eines Computers. Wie ich bereits angedeutet habe, basieren auch modernste Rechenmaschinen in ihrem Innersten auf äußerst einfachen Rechenoperationen, ähnlich

dem Verarbeitungsmechanismus einer Turing-Maschine; die Prinzipien sind also dieselben. Zudem ist Turings fiktive Maschine interessant für Informatikerinnen und Informatiker, die sich mit theoretischen Fragestellungen beschäftigen, und zwar als repräsentatives ComputermodeLL: Wenn sie zeigen wollen, ob sich ein Problem ganz allgemein mithilfe eines Computers berechnen lässt oder nicht, dann zeigen sie, dass dies mit einer Turing-Maschine geht bzw. nicht geht – und schon haben sie die Aussage auch für jeden beliebigen anderen Computer bewiesen.

Insgesamt hat Alan Turing mit seinem Gedankenexperiment und anderen Arbeiten die gesamte Computerwissenschaft so nachhaltig beeinflusst und geprägt, dass zu seinen Ehren der Turing-Award – quasi der Nobelpreis für Informatikerinnen und Informatiker – ins Leben gerufen wurde. Dass er sich in einer seiner Veröffentlichungen auch mit der Biologie beschäftigte und hier wichtige Impulse gab, ist hingegen weniger bekannt. Auch wir werden uns nun der Biologie zuwenden und am Beispiel einer Bakterie der Frage nachgehen, wie in der Natur „gerechnet“ wird.

Natürliche Informationsverarbeitung

Für die Natur sind die Grundprinzipien der automatisierten Informationsverarbeitung, wie sie Turing mit seinem Minimalcomputer skizziert hat, ein alter Hut. Jeder Organismus nimmt permanent Informationen aus seiner Um- und Innenwelt auf, wertet sie aus und merkt sich diese gegebenenfalls, um anschließend situativ darauf zu reagieren. Selbst das Konzept der Programmierbarkeit lässt sich wiederfinden, wenn wir die Variabilität des Erbguts mitberücksichtigen. Das in dieser Hinsicht ausgeklügeltste Lebewesen ist derzeit sicherlich der Mensch, dessen Gehirn enorme Informationsmengen bewältigt und speichert; aber auch Kleinstlebewesen, die nur aus einer einzigen Zelle bestehen, weisen ganz erstaunliche Steuerungsmechanismen auf. Somit ist die Thematik der selbstständig ablaufenden Informationsverarbeitung keinesfalls auf die Informatik beschränkt, stellt sie doch einen Kernaspekt des Lebens und der Evolution dar. Der Unterschied zwischen Biologie und Informatik liegt vielmehr in der Art der Systeme begründet, die untersucht werden: natürliche Organismen auf der einen Seite und vom Menschen für seine Zwecke kreierte Computer auf der anderen Seite.

Insofern scheint es naheliegend und kann aufschlussreich sein, die Informationsverarbeitung bei beiden Arten von Systemen zu vergleichen. Im Folgenden werden wir exemplarisch die Kolibakterie (Abb. 1.22), eines

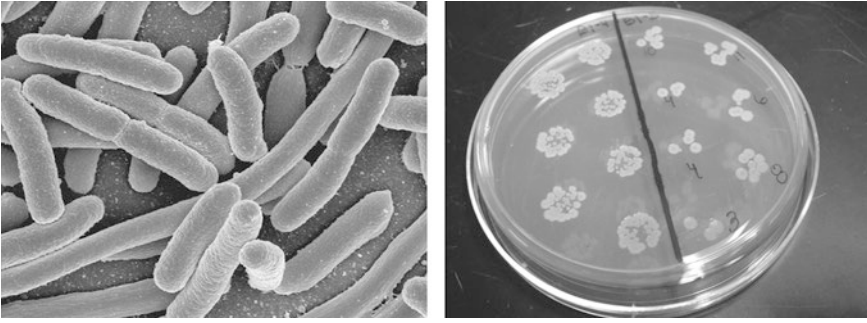


Abb. 1.22 Kolibakterien unter dem Elektronenmikroskop in 10.000-facher Vergrößerung als einzelne Zellen (links) und in einer Petrischale in Form von punktförmigen Kolonien, wie sie sich dem bloßem Auge darstellen (rechts). (© National Institute of Allergy and Infectious Diseases (links), NASA/Ames (rechts))

der am besten untersuchten Lebewesen überhaupt, der Turing-Maschine gegenüberstellen und – bei allen offensichtlichen Unterschieden – bemerkenswerte Gemeinsamkeiten entdecken. Das Interesse an diesem Bakterium kommt übrigens nicht von ungefähr. Obwohl es normalerweise als wertvoller Vitaminproduzent in unserem Darm lebt, gibt es einzelne Stämme, die für den menschlichen Organismus äußerst gefährlich sein können. Dieser Einzeller hat es also in sich!

Das Band des Lebens: die DNA

Die frappanteste Übereinstimmung ist das Vorhandensein eines Bands, das eine riesige Zeichenkette speichert und eine Schlüsselrolle bei der Informationsverarbeitung einnimmt. Das Band der Kolibakterie ist die berühmte DNA, allgemein bekannt als Trägerin des Erbguts. Man kann sie sich als enorm lange, verwirnte Kordel vorstellen, welche in einer Kapsel – dem stäbchenförmigen Bakterienkörper – aufbewahrt wird. Gemäß Abb. 1.23 scheint dieses Band nicht so akkurat aufgewickelt wie dasjenige der Turing-Maschine in Abb. 1.13; fast könnte man meinen, es sei schnell und unachtsam in die Kapsel hineingestopft worden. Doch das Gegenteil ist der Fall: All die Lebensvorgänge in der Zelle, in deren Zentrum die DNA steht, laufen mit höchster Präzision ab.

Erstaunlich ist auch der Aufbau dieses biologischen Langzeitspeichers, und es war eine Sensation, als 1953 zum ersten Mal seine chemische Struktur, die Doppelhelix, beschrieben wurde. An dieser Stelle soll uns jedoch einzig interessieren, dass es sich bei der DNA um ein Kettenmolekül

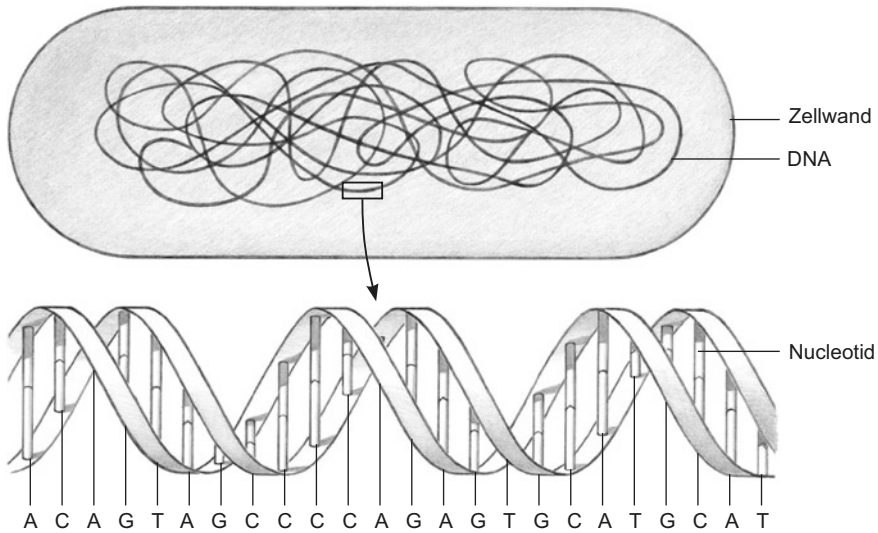


Abb. 1.23 Oben ist eine Kolibakterie im Querschnitt schematisiert; in der Mitte ist die DNA als langfädiges, verknäultes Molekül dargestellt. Ein Ausschnitt der DNA ist unten in einer detaillierten Zeichnung zu sehen, zusammen mit der codierten Zeichenfolge. (© Susanne Staubli, Eckart Zitzler)

handelt, das sich aus vier Arten von Kettengliedern zusammensetzt: den Nucleotide genannten Molekülen Adenin, Cytosin, Guanin und Thymin. Die Nucleotide sind wie auf einer Perlenschnur hintereinander aufgereiht, und da ihre Namen häufig durch die entsprechenden Anfangsbuchstaben abgekürzt werden, lässt sich die DNA als eine Zeichenfolge begreifen: das Buch des Lebens, verfasst in einer Alphabetschrift mit den vier Buchstaben A, C, G und T (Abb. 1.23). In der Kolibakterie umfasst der codierte Text mehr als 4 Mio. Zeichen, entsprechend lang ist das biologische Band. Die DNA ist ausgestreckt mehr als 200-mal länger als eine einzelne Kolibakterie und muss mehrfach gefaltet und dicht verpackt werden, um in das Innere der Zelle hineinzupassen. Die so erzielte Speicherdichte ist enorm. Um den in einem Gramm DNA codierten Text zu speichern, bräuchte man eine Milliarde mal eine Milliarde Lochkarten mit einem Gesamtgewicht von über 6 Mio. t.

Schließlich gibt es auch ein Pendant zum Lese- und Schreibkopf der Turing-Maschine: ein Molekül namens RNA-Polymerase, über das der Bandinhalt ausgelesen werden kann (Schreiben ist eine andere Geschichte, dazu später mehr). Die RNA-Polymerase lagert sich an der DNA an, tastet sie Nucleotid für Nucleotid bzw. Zeichen für Zeichen ab und überträgt

die gelesene Sequenz auf ein anderes Molekül. Es wird also eine Abschrift erstellt, allerdings nicht von der gesamten DNA, sondern nur von kleineren Abschnitten, Gene genannt. Anfang und Ende dieser Abschnitte sind jeweils durch spezielle Nucleotidabfolgen markiert, sodass die RNA-Polymerase weiß, von wo bis wo sich das abzulesende Textsegment erstreckt.

Welchem Zweck die resultierenden Textschnipsel dienen? Im Wesentlichen sind es Konstruktionspläne, anhand derer die Kolibakterie in einem komplizierten Zusammenspiel von über 50 molekularen Maschinen Proteine herstellt, d. h. Werkzeuge und Baumaterial für den Lebensunterhalt (Bsp. 1.5). Proteine sind essenziell für das Funktionieren des Organismus, denn diese Moleküle sind an nahezu allen Vorgängen in der Zelle beteiligt und regeln diese – kein Wunder, machen sie gesamt- haft doch mehr als die Hälfte des Trockengewichts einer Kolibakterie aus. Entsprechend groß sind ihre Formenvielfalt – vom Stab über das Netz bis zur Hohlkugel – und ihr Anwendungsspektrum. Als Enzyme können Proteine Stoffe verbinden, zerlegen und umwandeln, also chemische Reaktionen in Gang setzen; als Transportmittel bewegen sie; als Trägerelemente stabilisieren sie; und als Teamplayer können sie molekulare Miniaturmaschinen bilden, von der Pumpe zum Motor bis zur komplexen Baumaschine – es scheint nichts zu geben, was es nicht gibt. Es sind folglich die Proteine, über die die Bakterienzelle die Lebensvorgänge in ihr steuert.

Beispiel 1.5: Wie ein Protein hergestellt wird

Abb. 1.24 skizziert den Ablauf, wie aus dem auf der DNA gespeicherten Bauplan ein Protein hergestellt wird. Die RNA-Polymerase übt die Funktion eines Lesekopfes aus, indem sie an der DNA entlangfährt und die gespeicherten Informationen auf ein anderes Molekül – die Abschrift – überträgt. Aus der Abschrift wird anschließend in hier nicht weiter erläuterten Verarbeitungsschritten das Protein zusammengesetzt, dessen Aussehen je nach Bauplan stark variieren kann.

Damit haben wir bereits einen Teil des Informationsverarbeitungsapparats einer Kolibakterie kennengelernt, nämlich die DNA, die die einzelnen Verarbeitungsschritte und ergo den gesamten Verarbeitungsprozess beschreibt, sowie die Proteine, die die Verarbeitungsschritte effektiv umsetzen. Mit den Begrifflichkeiten der Biologie könnte man es auch so ausdrücken, dass die DNA das Verhaltensrepertoire definiert und die Proteine konkrete Reaktionen realisieren. Ein wichtiger Aspekt fehlt dabei allerdings noch: die zeitliche Steuerung. Ein Verarbeitungsvorgang im Sinne einer

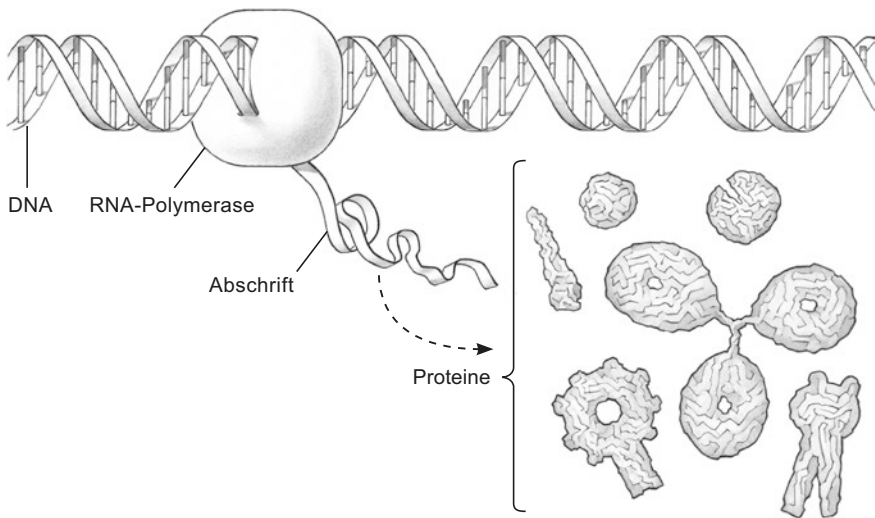


Abb. 1.24 Von der DNA zum Protein, in Anlehnung an Zeichnungen von David S. Goodsell (2010). (© Susanne Staubli, Eckart Zitzler)

komplexen Reaktion ist zum einen situationsabhängig und zum anderen zusammengesetzt aus einzelnen Teilschritten, die in einer bestimmten Reihenfolge ausgeführt werden, also aufeinander abgestimmt sein müssen. Daher dürfen nicht alle Proteine in der Zelle gleichzeitig operieren – nicht auszumalen, wenn alle möglichen Reaktionen simultan ablaufen würden. Entscheidend ist, dass sie gezielt und nach Bedarf eingesetzt werden. Wie entscheidet die Kolibakterie also, wann welches Protein zum Einsatz kommt, und wie wird diese Entscheidung in die Tat umgesetzt?

Proteine beeinflussen den DNA-Ablesevorgang

Es ist kaum verwunderlich, dass wiederum Proteine bei der Regulation der Proteinproduktion eine wesentliche Rolle spielen, indem sie sich an spezifische Stellen der DNA andocken und damit verhindern, erschweren, fördern oder ermöglichen, dass bestimmte DNA-Abschnitte abgelesen werden können. Wir werden den Mechanismus an einem konkreten Vorgang studieren: wie eine Kolibakterie Nahrung in Form von Milchzucker aufnimmt und verwertet. Dazu ist es nützlich zu wissen, dass die Kolibakterie zum festen Bestandteil unserer Darmflora gehört und dort wichtige Funktionen ausfüllt, sei es, dass sie Nährstoffe spaltet, Vitamine bildet oder für die Abwehr

von Krankheitserregern sorgt. Sie selbst lebt von dem, was wir uns mit dem Essen zuführen – bevorzugt von Traubenzucker, aber wenn der nicht verfügbar ist, kann es eben auch Milchzucker sein (Abb. 1.25). Letzteren kann die Kolibakterie allerdings nicht direkt verwerten, sie muss ihn zunächst einmal ins Zellinnere befördern und dort in seine Bestandteile zerlegen. Für beide Aufgaben gibt es spezialisierte Proteine: ein Transport- und ein Schneideprotein.

Die zwei Proteine werden allerdings nur produziert, wenn sie auch wirklich gebraucht werden. Die Kolibakterie muss ihre Werkzeuge ökonomisch und gezielt einsetzen, daher werden auch nicht mehr benötigte Proteine durch andere Proteine abgebaut – die Bausteine der Proteine, die Aminosäuren, bleiben so im Spiel. Die Produktion muss sich also an- und ausschalten lassen, und das wird mittels eines extra dafür konzipierten Schalterproteins erreicht. Dieses Schalterprotein ist so beschaffen, dass es sich an einer ganz bestimmten Stelle an der DNA anlagern kann – genau dort, wo die Konstruktionspläne für die besagten zwei Proteine abgelegt sind. Im Normalzustand ist der Schalter auf „aus“ gelegt, d. h., das Schalterprotein ist an der DNA angedockt (Abb. 1.26 links). Die Folge ist, dass die RNA-Polymerase, der Lesekopf, nicht oder nur erschwert an der DNA entlangfahren und die Abschrift erstellen kann. Die Produktion von Transport- und Schneideprotein ist folglich stark eingeschränkt. Erst wenn Milchzuckermoleküle mit dem Schalterprotein in Berührung kommen, verändert letzteres seine Form, löst sich von der DNA und gibt so die Produktion von Transport- und Schneideprotein frei; auf diese Weise wird der Schalter umgelegt (Abb. 1.26 rechts).

Dieses Beispiel illustriert, wie eine Kolibakterie Informationen aus ihrer Umwelt verarbeitet und daraus ein Verhalten „berechnet“, wenn man so sagen will. Vereinfacht ausgedrückt: Ein Signal („Milchzucker vorhanden“)



Abb. 1.25 Kolibakterien ernähren sich u. a. von verschiedenen Zuckern, wie sie in Trauben oder Milchprodukten enthalten sind. (© Eckart Zitzler)

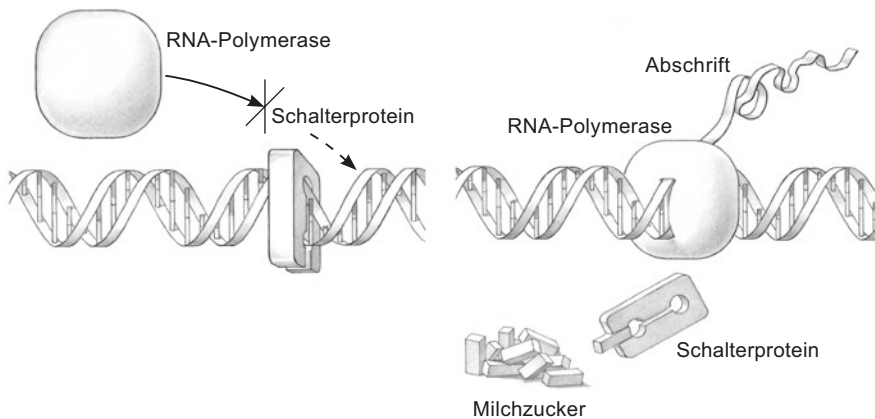


Abb. 1.26 Zusammenspiel von Schalterprotein und Milchzuckermolekülen. (© Susanne Staubli, Eckart Zitzler)

trifft ein, ein Protein reagiert darauf, womit die Herstellung anderer Proteine angestoßen und eine Antwort („Milchzucker verwerten“) provoziert wird. Im Detail ist das alles viel komplizierter, z. B. kann ein Schalterprotein auch ein anderes Schalterprotein aktivieren, womit sich vielfach verästelte Kaskaden solcher Schaltvorgänge sowie Rückkopplungen ergeben können. Für den Moment hingegen wollen wir es bei dieser oberflächlichen Betrachtung belassen.

Sind Kolibakterien Turing-Maschinen?

Jetzt, nachdem wir die groben Züge ihres Verarbeitungsmechanismus kennengelernt haben, können wir die Kolibakterie der Turing-Maschine gegenüberstellen und aus der Perspektive der Informatik beleuchten. Um den Vergleich zu vereinfachen, betrachten wir ein leicht modifiziertes Computermode. Bei dieser Variante werden die Tabellen nicht über einen Lochkartenleser, sondern über ein Zusatzband eingespeist. Die Beschreibung der Rechenoperationen ist also in einem geeigneten Format auf diesem Speichermedium abzulegen, bevor die Turing-Maschine gestartet wird. Während der Programmausführung schaut die Steuereinheit dann jeweils auf dem Zusatzband nach, um den nächsten Berechnungsschritt zu bestimmen. Anm. 1.2 verdeutlicht die Idee.

übernimmt damit die gleiche Rolle, die die Steuereinheit bei der Turing-Maschine innehat. Die Konstruktionspläne für die Proteine, die als Gene auf der DNA festgeschrieben sind, korrespondieren mit den Tabellen, die die Rechenoperationen der Turing-Maschine beschreiben; vereinfacht ausgedrückt: **Gene = Tabellen**. Folglich enthält die DNA – analog dem Zusatzband – das Programm und definiert das Spektrum der zur Verfügung stehenden Befehle, ihr kommt die Funktion eines Programmspeichers zu; sprich: **DNA = Zusatzband**. Dann sind da noch die Stoffe und Signale, die die Kolibakterie pausenlos aufnimmt und verarbeitet; ihnen stehen die Zahlen und ganz allgemein die Daten gegenüber, die die Turing-Maschine bearbeitet, also: **Stoffe/Signale = Daten**. Dabei erfüllt der Bakterienkörper die Aufgaben des Bands, weil er als Reservoir für die Daten, also als Datenspeicher fungiert. Es ließen sich über die in Abb. 1.28 gezeigten Parallelen hinaus noch viele weitere aufführen, z. B. zwischen den Schalterproteinen (die durch das Andocken an die DNA beeinflussen, welche Gene überhaupt abgelesen werden) und der gerade aktiven Tabelle bei der Turing-Maschine; doch die erwähnten Punkten mögen hier genügen.

Natürlich ist die Darstellung simplifiziert. Trotzdem haben die Entsprechungen etwas für sich, denn tatsächlich lassen sich bei der Kolibakterie die drei beschriebenen Charakteristika eines Computers wiederfinden: Speicherfähigkeit, selbsttätige Ablaufsteuerung und – im übertragenen Sinn – Programmierbarkeit. Letztere Eigenschaft ist erfüllt, weil die DNA veränderbar ist und dadurch die Art und Weise der Informationsverarbeitung modifiziert werden kann. Das Schreiben erfolgt

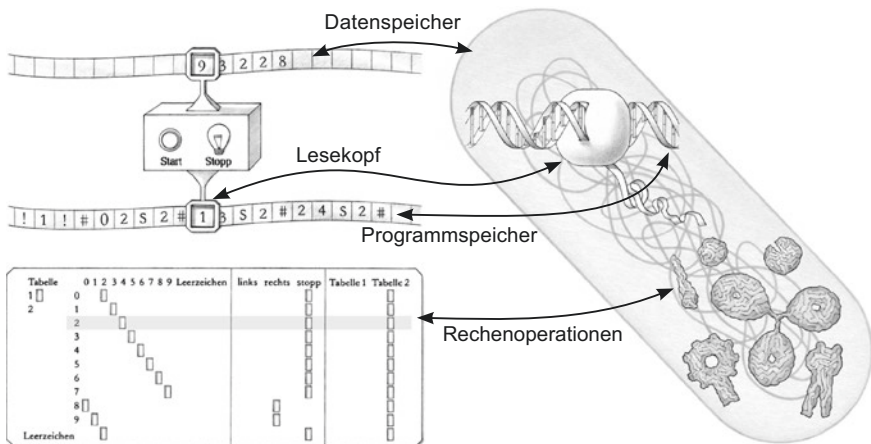


Abb. 1.28 Analogien in der Informationsverarbeitung zwischen Turing-Maschine und Kolibakterie. (© Susanne Staubli, Eckart Zitzler)

allerdings nicht über einen Lese- und Schreibkopf, sondern über andere Mechanismen wie Mutationen. So gesehen stellt sich durchaus die Frage, ob unser Darmbewohner im Grunde genommen nicht eine Art erweiterte Turing-Maschine repräsentiert. So interessant solche Analogien auch sind, sie haben immer ihre Grenzen. Die Tücke steckt bekanntlich im Detail.

Wundersam vertrackt wird es beispielsweise beim Begriff des Programms. Bei einer Turing-Maschine verstehen wir darunter eine Folge von einzelnen, nach dem gleichen Schema aufgebauten Rechenschritten, die bei gleicher Eingabe stets dasselbe Ergebnis produziert. Bei einer Kolibakterie ist es viel schwieriger darzulegen, was eigentlich das Programm ist: Am ehesten lässt es sich als ein mit der Umwelt wechselwirkendes Geflecht vieler verschiedener Verarbeitungsaktivitäten beschreiben, welches erstaunlich präzise, aber niemals exakt gleich reagiert. Auch bei der Konstruktion wartet die Kolibakterie mit so einigen Besonderheiten auf. Beispielsweise ist das Pendant zur Steuereinheit, die Gesamtheit an Proteinen, selbst Teil der Programmausführung, da ihre Baupläne auf der DNA gespeichert und abgelesen werden. Eine Maschine, die sich selbst erschafft? Das erinnert an das Motiv der zwei sich gegenseitig zeichnenden Hände (Abb. 1.29), ursprünglich 1948 vom Holländer M. C. Escher erdacht und seitdem vielfach variiert.

Damit ist klar: Eine Kolibakterie lässt sich nicht einfach als ein Computer in anderer Gestalt betrachten, zu unterschiedlich sind die beiden Systeme in ihrer Wesensart. Zumal sich der Aufbau einer Bakterie nicht an den Belangen des Menschen, z. B. Übersichtlichkeit, Handhabbarkeit und Beherrschbarkeit, orientiert, sondern sich an der Anpassungs- und

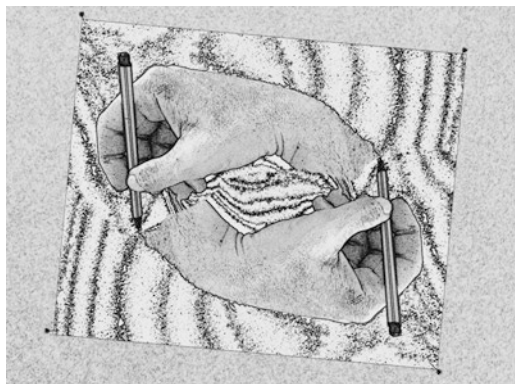


Abb. 1.29 Zwei sich zeichnende Hände als Metapher für das intrazelluläre Wechselspiel zwischen DNA und Protein, hier als Fotocollage in Szene gesetzt. (© Eckart Zitzler)

Überlebensfähigkeit des Organismus ausrichtet – das macht die Biologie ja so faszinierend. Vielleicht ist das auch gar nicht das Überraschende und Entscheidende an dieser Gegenüberstellung. Erstaunlich ist vielmehr, wie viele Querbezüge und Ähnlichkeiten wir feststellen können, **obwohl** Turing-Maschine und Kolibakterie auf den ersten Blick rein gar nichts miteinander zu tun haben. Denken wir nur an das Band als Speichermedium. Wohlgermerkt hat Alan Turing seine Ideen gut 15 Jahre vor der Entdeckung der DNA-Struktur entwickelt. Die Übereinstimmungen deuten darauf hin, dass gewisse Grundprinzipien der automatischen Informationsverarbeitung generell sind, obschon wir gesehen haben, dass bei Verallgemeinerungen Vorsicht angebracht ist. Gerade deshalb ist es stimulierend und aufschlussreich, die Dinge einmal aus der Perspektive der anderen Disziplin zu betrachten; es fördert das Verstehen und Durchdringen auf beiden Seiten. So ein Perspektivenwechsel kann schließlich dazu führen, dass eigenständige Forschungsgebiete entstehen, die beide Welten miteinander verbinden.

Rechnen mit DNA

DNA Computing ist so ein Forschungsgebiet. Wie die englische Bezeichnung nahelegt, dreht sich alles darum, auf der Basis von DNA Berechnungen durchzuführen, quasi eine Art DNA-basierten molekularen Computer zu bauen. Dabei hat man sich weniger einen Computer aus Fleisch und Knochen oder eine riesige rechnende Zelle vorzustellen, sondern das Ziel ist vielmehr, Kernprozesse der Informationsverarbeitung – wie in der Kolibakterie – mittels chemischer Vorgänge zu realisieren und dabei die DNA – anders als in der Kolibakterie – als Datenspeicher zu benutzen.

Eine verwegene Idee, doch wenn wir uns die obige Diskussion vor Augen führen, nicht abwegig. Die Natur stellt ja die Werkzeuge, mit der DNA gezielt manipuliert werden kann, größtenteils selbst zur Verfügung: Proteine. Mit gewissen lassen sich DNA-Moleküle kopieren und vervielfältigen, andere können DNA-Moleküle an bestimmten Stellen zerteilen, und wiederum andere ermöglichen es, DNA-Moleküle spezifisch aneinanderzuketten und zu einem Strang zu verbinden. Daneben gibt es die Biotechnologie. Deren Methoden erlauben es beispielsweise, DNA-Moleküle gemäß einer vorgegebenen A-C-G-T-Sequenz synthetisch herzustellen, den in einem DNA-Molekül gespeicherten Text auszulesen oder DNA-Moleküle nach Größe zu sortieren und entsprechend zu selektieren. Setzt man diese Hilfsmittel geschickt ein, dann lassen sich einfache Operationen auf molekularen Zeichenketten implementieren – und schließlich Berechnungen durchführen, denn auch die Turing-Maschine rechnet ja

einzig mittels elementarer Zeichenoperationen und besitzt trotzdem die gleiche Mächtigkeit wie ein moderner Computer.

Der Beweis, dass diese Idee Hand und Fuß hat, wurde 1994 von Leonard Adleman erbracht, einem amerikanischen Informatiker (Abb. 1.30). Indem Adleman ein Verfahren vorstellte, wie sich ein bekanntes informatisches Problem, das Hamilton-Pfad-Problem, im Prinzip mit den oben aufgeführten Hilfsmitteln – DNA, Enzymen und biotechnologischen Methoden – lösen lässt, wurde DNA Computing Realität.

Das Hamilton-Pfad-Problem – wir betrachten hier eine spezielle Variante – ist schnell erklärt. Auf einem Stadtplanausschnitt ist eine Route von einem Start- zu einem Zielpunkt zu ermitteln, die alle auf dem Ausschnitt ersichtlichen Kreuzungen genau einmal quert. Diese Aufgabenstellung kann sich z. B. ergeben, wenn an den Kreuzungen Abfallbehälter stationiert sind und die Müllabfuhr diese in einer einzigen Tour leeren möchte. Einfach zu erklären heißt jedoch nicht einfach zu lösen – die Routenwahl kann zu einer echten Herausforderung werden, insbesondere wenn viele Einbahnstraßen die Fahrtrichtungen einschränken. Wie ist nun Adleman dieses Problem angegangen? Mit roher Gewalt, könnte man sagen, denn die verwendete Strategie ist in Informatikerkreisen unter dem englischen Ausdruck *brute force* bekannt.

Zunächst einmal hat er sich überlegt, wie eine Route auf der Basis des A-C-G-T-Alphabets beschrieben werden kann. Dazu hat er die Kreuzungen mit eindeutigen Namen versehen, z. B. AC oder AG. Die Bezeichnungen für die Fahrbahnen hat er anschließend aus diesen abgeleitet, und zwar durch Aneinanderreihung der Namen der zwei verbundenen Kreuzungen. Die Fahrbahn, die von Kreuzung AC zu Kreuzung AG führt, heißt dann ACAG, wie in Abb. 1.31 illustriert. In diesem Fall ist eine Route nichts



Abb. 1.30 Leonard Adleman, der Begründer des DNA Computing. (© Camille Büsser)

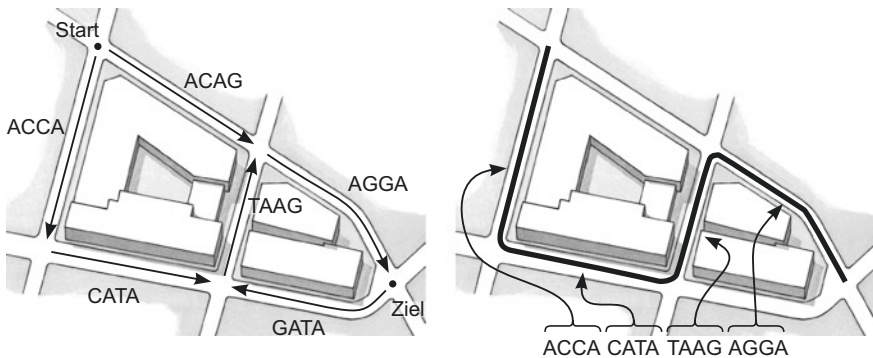


Abb. 1.31 Ein Stadtplanausschnitt mit fünf Kreuzungen und sechs Einbahnstraßen: Auf der linken Seite lässt sich erkennen, wie die Kreuzungen und die erlaubten Fahrtrichtungen benannt werden können; die rechte Seite zeigt das Beispiel einer korrekten Route und ihrer Beschreibung als A-C-G-T-Zeichenkette vom Start- zum Zielpunkt. (© Susanne Staubli, Eckart Zitzler)

anderes als eine Folge von Fahrbahnbezeichnungen, d. h. A-C-G-T-Zeichenketten, und lässt sich entsprechend mittels eines DNA-Moleküls repräsentieren. Doch wie kommt man zu diesem DNA-Molekül und vorzugsweise zu einem, das die gesuchte Route verkörpert?

Das Prinzip an sich ist simpel. Wir stellen für jede Fahrbahn ein entsprechendes DNA-Molekül als Namensträger her, vervielfältigen die resultierenden DNA-Schnipsel in enormen Mengen, lassen nun beliebige Kombinationen aus diesen DNA-Schnipseln entstehen und sortieren anschließend alle DNA-Stränge, die keine gültige Route repräsentieren, aus. Bsp. 1.6 erläutert die einzelnen Arbeitsschritte.

Beispiel 1.6: Routen finden mittels DNA Computing

In Abb. 1.32 sehen Sie den „Berechnungsprozess“ – es ist ja eigentlich ein chemischer Verarbeitungsprozess – grafisch dargestellt. Die Ausgangsbasis bilden die Namen der vorhandenen Fahrbahnen zwischen den Kreuzungen, die synthetisch in Form von DNA-Schnipseln hergestellt werden und die elementaren Lösungsbausteine repräsentieren. In einem ersten Verarbeitungsschritt werden diese Bausteine in großen Mengen vervielfältigt. Im zweiten Schritt werden die DNA-Moleküle zu längeren DNA-Strängen zusammengeklebt – zufällig, je nachdem, welches Molekül gerade auf welches trifft, aber immer so, dass die zusammengefügte Teilrouten an einer Kreuzung ineinander übergehen, sich also zu einem Weg ergänzen. Der dritte Verarbeitungsvorgang umfasst mehrere Filterungsstufen, die zum Ziel haben, ungültige Routen auszusortieren. Da muss z. B. getestet werden, ob alle Kreuzungsnamen **mindestens** einmal vorkommen, und gleichzeitig muss sichergestellt sein, dass jeder Kreuzungsname

höchstens einmal vorkommt. Das Filtern hat den Effekt, dass alle DNA-Moleküle, die zum Schluss im Reagenzglas übrig bleiben, gültige Routen verkörpern. Entsprechend genügt es im vierten Schritt, einen der verbleibenden DNA-Stränge in die dazugehörige Buchstabensequenz zurückzuübersetzen; Nucleotid für Nucleotid wird das Molekül ausgelesen und in die codierte A-C-G-T-Zeichenkette übersetzt, welche die gesuchte Lösung beschreibt.

Das Ganze setzt natürlich voraus, dass beim Kombinieren der Schnipsel tatsächlich eine korrekte Route generiert wird; hundertprozentige Sicherheit gibt es hierbei nicht, nur eine sehr hohe Wahrscheinlichkeit. Dieser Ansatz, effektiv alle möglichen – korrekten und unsinnigen – Routen zu generieren und dann die meisten davon wegzwerfen, hat tatsächlich etwas Brachiales an sich, doch die Winzigkeit der Moleküle und die massive Parallelität bei den chemischen Prozessen machen dies möglich, wenn die Anzahl der Kreuzungen nicht zu groß ist. Das also kann Rechnen im Reagenzglas heißen.

Zugegeben: Das Ganze ist ziemlich aufwändig und erinnert noch nicht wirklich an einen universell einsetzbaren, bedienfreundlichen Computer – doch wie gesagt, es war die Geburt eines Forschungsgebiets. Natürlich stellt sich die Frage, was man sich vom Rechnen mit DNA-Molekülen verspricht. Die reine Neugierde spielt in der Wissenschaft immer eine Rolle,

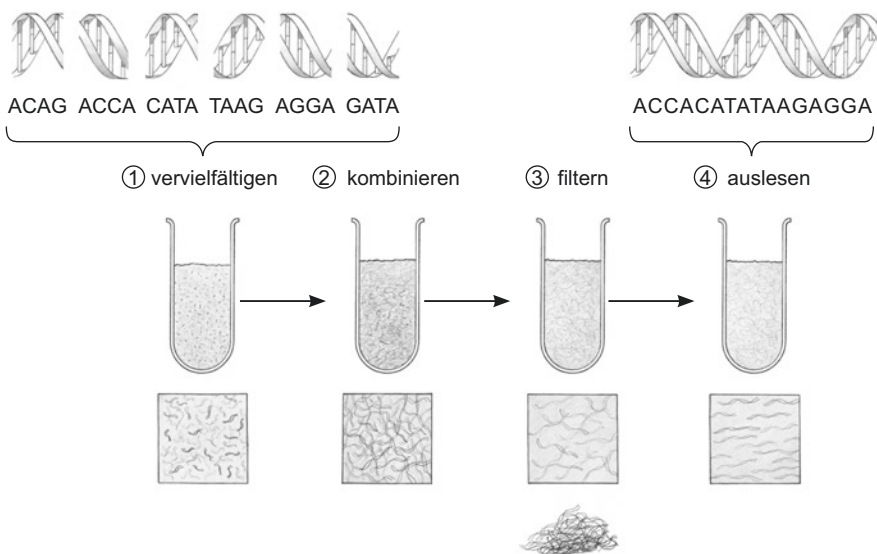


Abb. 1.32 Wie sich mit DNA rechnen lässt, illustriert am Beispiel des Hamilton-Pfad-Problems. (© Susanne Staubli, Eckart Zitzler)

aber letztlich steckt wesentlich mehr dahinter. Der große Vorteil eines wirklichen DNA-Computers gegenüber seinem elektronischen Bruder wäre seine Fähigkeit, eine immense Anzahl von Rechenoperationen auf kleinstem Raum und bei sehr geringem Energieaufwand parallel ausführen zu können. Tatsächlich haben sich Forscher Gedanken darüber gemacht, wie sich eine Turing-Maschine und abgespeckte Varianten davon auf der Basis von DNA, Proteinen und weiteren Molekülen bauen ließen – zunächst in der Theorie. Adlemans Brute-force-Ansatz erwies sich jedoch als nicht praktikabel: Zur Lösung realer Probleme bräuchte man eine unvorstellbare Menge an Molekülen. Trotzdem gehen die Arbeiten weiter. Knapp 20 Jahre nach Adlemans Versuchen gelang es Forschenden, DNA als Datenspeicher zu verwenden und eine Art biologischen Transistor zu bauen. Noch stellt DNA Computing hauptsächlich eine Ideen-Spielwiese dar, denn konkrete praktische Anwendungen haben sich bislang nicht daraus ergeben. Doch der Ideen gibt es viele. Wir werden sehen, wohin die Forschungsarbeiten effektiv führen und ob wir jemals eine organische Miniaturrechenmaschine in unseren Händen halten werden.

Fazit

Der Ausgangspunkt für dieses Kapitel war die simple Frage „Was ist ein Computer?“, und von da aus ging es im Schnelldurchlauf von der Geschichte des Computers über das grundlegende Berechnungsmodell der Theoretischen Informatik zu den Grundlagen der Molekularbiologie, inklusive Synthese dieser Themen. Wir haben gesehen: Ein Computer ist weit mehr als ein Rechner, er stellt ganz allgemein ein künstliches System dar, mit dessen Hilfe sich die Verarbeitung von Informationen automatisieren lässt. Charakteristisch ist, dass er Daten speichern sowie diese gemäß einer vorgegebenen Abfolge von Operationen manipulieren kann und dass er es ermöglicht, die Verarbeitungsvorschrift selbst flexibel einzustellen. Im weitesten Sinne weisen auch Lebewesen diese Eigenschaften auf, mit dem Unterschied, dass es die Natur ist, die diese Systeme erschafft. Daran lässt sich erkennen, dass grundlegende Prinzipien der automatischen Informationsverarbeitung universell sind und bereits Milliarden von Jahren vor dem Zeitpunkt erfunden wurden, als der erste Computer die Weltbühne betrat. Insofern haben viele informatische Fragestellungen einen Bezug zur Biologie und umgekehrt; eigenständige Forschungsgebiete, die beide Welten miteinander verbinden, unterstreichen dies.

Zum Nach- und Weiterlesen

- Goodsell, D. S. (2010). *Wie Zellen funktionieren: Wirtschaft und Produktion in der molekularen Welt* (2. Aufl.). Heidelberg: Spektrum Akademischer Verlag.
- Hofstadter, D. R. (1992). *Gödel, Escher, Bach – ein Endloses Geflochtenes Band*. München: Deutscher Taschenbuch.
- Lippe, W.-M. (2013a). *Die Geschichte der Rechenautomaten: Von der Himmelscheibe von Nebra bis zu den ersten Rechenmaschinen*. Berlin: Springer Vieweg.
- Lippe, W.-M. (2013b). *Die Geschichte der Rechenautomaten: Von mechanischen Chiffriergeräten bis zu den ersten programmierbaren Rechnern*. Berlin: Springer Vieweg.
- Padua, S. (2015). *The thrilling adventures of lovelace and babbage: The (Mostly) true story of the first computer*. New York: Pantheon Books.
- Schubert, S., & Schwill, A. (2011). *Didaktik der Informatik* (2. Aufl.). Heidelberg: Spektrum Akademischer.

Bildquellen

- Abb. 1.3: NASA, <http://spaceflight.nasa.gov/gallery/images/shuttle/sts-101/html/jsc2000e10522.html> (links, abgerufen am 11.04.2017); NASA, http://www.nasa.gov/mission_pages/station/multimedia/S.135e011867.html (rechts, abgerufen am 11.04.2017)
- Abb. 1.7: Reinhard Atzbach, <http://www.rechenwerkzeug.de/stephenson.htm> (abgerufen am 11.04.2017)
- Abb. 1.11: U. S. Army, <http://ftp.arl.mil/ftp/historic-computers/> (abgerufen am 22.08.2014)
- Abb. 1.13: Michael Davey, <http://www.aturingmachine.com/> (abgerufen am 11.04.2017)
- Abb. 1.18: NASA, <https://www.flickr.com/photos/nasacommons/9467782802/in/set-72157634982467137> (links, abgerufen am 11.04.2017)
- Abb. 1.21: NASA/JPL-Caltech/MSSS, <http://photojournal.jpl.nasa.gov/catalog/PIA18390> (links, abgerufen am 11.04.2017); NASA Goddard MODIS Rapid Response Team, http://www.nasa.gov/sites/default/files/arthur.a2014185.1520.2km_2.jpg (rechts, abgerufen am 11.04.2017)
- Abb. 1.22: National Institute of Allergy and Infectious Diseases, http://www.niaid.nih.gov/topics/biodefenserelated/biodefense/publicmedia/Pages/image_library.aspx (links, abgerufen am 22.08.2014); NASA/Ames, http://www.nasa.gov/centers/ames/multimedia/images/2006/genebox_prt.htm (rechts, abgerufen am 11.04.2017)

2

Schalten und Walten

Überblick

Wie lässt sich eigentlich mit Strom rechnen? Um diese Frage geht es in diesem Kapitel. Dazu schauen wir uns an, wie Zahlen durch Stromkreise repräsentiert werden können. Wir lernen den stromgesteuerten Schalter kennen – das Kernelement für die elektronische Informationsverarbeitung – und werden sehen, dass sich durch die geschickte Verschaltung von Stromkreisen Rechenoperationen, aber auch anderweitige Verarbeitungsoperationen implementieren lassen. Zudem setzen wir uns mit dem Prinzip der Rückkopplung auseinander, auf dessen Basis sich Speicherelemente bauen lassen. Damit es praktisch bleibt, konstruieren wir eine einfache elektrische Rechenmaschine und setzen die gewonnenen Erkenntnisse dabei um. Am Ende wissen Sie, wie sich Automatik und Speicherfähigkeit mittels elektrischer Bauelemente realisieren lassen. Das Faszinierende dabei: In lebenden Zellen und im Gehirn finden wir die gleichen Prinzipien wieder, die Natur hat sie lange vor der Informatik entdeckt.

Alan Turing hat es uns mit seiner Maschine vor Augen geführt: Automatisierte Informationsverarbeitung lässt sich begreifen als ein Vorgang, bei dem Zeichen jedweder Art in andere Zeichen überführt werden. Nach diesem Prinzip arbeiten auch moderne Computer: Sie führen Berechnungen durch, indem sie die Ziffern einer Zahlenfolge schrittweise durch andere Ziffern ersetzen. Turing war insofern äußerst weitsichtig, allerdings beantwortete seine gedankliche Konstruktion nicht die Frage der praktischen Umsetzung. Erst als man herausfand, wie sich elektrische Spannung nutzen lässt, um Zeichen zu repräsentieren, und wie elektrisch betriebene Schalter eingesetzt werden können, um einzelne Zeichen zu ersetzen, wurde der erste

Computer Realität. Und genau darum geht es in diesem Kapitel: um das Rechnen mit Strom.

Damit wir uns nicht in den Untiefen der Digitaltechnik – sie bildet die Grundlage der heutigen Computertechnologie – verlieren, halten wir uns an etwas Konkretes. Wir konstruieren ein elektrisches Pendant zum mechanischen Stephenson-Addierer aus Kap. 1: einen einfachen stromgesteuerten Taschenrechner. Zunächst konzentrieren wir uns darauf, zwei Zahlen zu addieren; später werden wir dann Zwischenergebnisse festhalten und Zahlenkolonnen aufsummieren. Zugegeben, das haut einen noch nicht vom Hocker. Doch in dieser Anwendung sind alle Aspekte enthalten, um die unterste Ebene der Computer-Hardware zu verstehen.

Noch ein Hinweis, bevor Sie sich ins Getümmel mit den Nullen und Einsen stürzen. Die Auseinandersetzung mit den elektronischen Grundlagen des Computers erfordert etwas Puste. Wir zoomen hinein bis in die Sphäre des Transistors – auch wenn wir seinen Vorgänger, das Relais, als Ausgangspunkt nehmen. Da wird es naturgemäß technisch, und es kann schon vorkommen, dass Sie den Computer vor lauter Schaltern nicht mehr sehen. Denken Sie an Charles Babbage: Er hat ein Leben lang an dem mechanischen Fundament seiner Analytical Engine getüftelt. So viel Zeit sollte das Lesen dieses Kapitels aber nicht in Anspruch nehmen.

Am Anfang ist die Zahl

Legen wir also los und schauen zunächst einmal, wie sich Zahlen in einem elektrischen System darstellen lassen – das ist ja die Voraussetzung dafür, dass wir überhaupt rechnen können. Bei den mechanischen Rechenmaschinen waren es Zahnräder, über deren Stellung Zahlen ausgedrückt werden konnten, bei elektrischen Rechenmaschinen sind es hingegen Stromkreise, die als elementare Zeichenträger fungieren.

Der Stromkreis als elementarer Zeichenträger

Ein einfacher Stromkreis besteht aus einer Spannungsquelle, einem Schalter, einem Verbraucher und Leitungen, die die Elemente miteinander verbinden, wie in Abb. 2.1 zu sehen ist. Das kennen wir bestens aus dem Alltag, denken wir nur an eine Taschenlampe oder eine Wohnzimmerleuchte: Die Batterie bzw. die Steckdose stellt den Strom zur Verfügung, mit dem Schalter können wir den Stromkreis schließen sowie unterbrechen und die integrierte

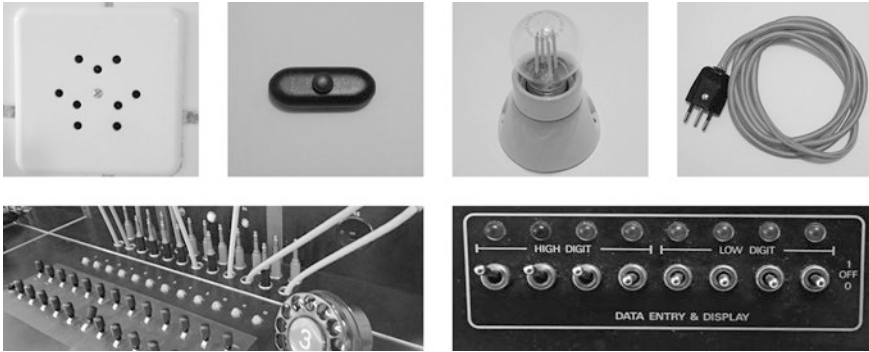


Abb. 2.1 Die Elemente eines einfachen Stromkreises (oben) und ihre frühere Verwendung in Telefonvermittlungsanlagen und Computerkonsolen (unten). (© Eckart Zitzler (oben), Eckart Zitzler (unten, mit freundlicher Genehmigung des Museums ENTER Solothurn))

Lichtquelle zeigt uns als Verbraucher unmittelbar an, ob Strom fließt oder nicht.

Aufgrund der zwei möglichen Schalterstellungen können wir zwei Zustände unterscheiden, in denen sich ein Stromkreis befinden kann: „an“ oder „aus“ bzw. „offen“ oder „geschlossen“. Natürlich können wir auch weitere Charakterisierungen in Betracht ziehen, beispielsweise wie viel Strom fließt – damit hätten wir ein Kontinuum an möglichen Zuständen. Beim Computer hat es sich jedoch bewährt, intern nur mit den erwähnten zwei Zuständen, also digital anstatt analog zu arbeiten (Anm. 2.1). Das Prinzip der Digitalisierung macht die Computer-Hardware robuster und vereinfacht ihre Konstruktion.

Anmerkung 2.1: Digital und analog

Digital heißt wortwörtlich, dass etwas mit den Fingern abzählbar ist – die Wortwurzel bildet das lateinische Wort *digitus* = Finger. Man meint damit, dass nur eine diskrete Menge von Zuständen unterschieden wird, also „an“ oder „aus“ beim Stromkreis oder „warm“ und „kalt“ bei der Temperatur. Dabei lassen sich durchaus mehr als zwei Zustände betrachten, z. B. zusätzlich „heiß“ und „eisig“, die Beschränkung auf zwei Werte ist lediglich eine Möglichkeit der Digitalisierung, die sich in der Computertechnologie durchgesetzt hat.

Das Pendant zu digital ist analog, womit ein Kontinuum an Zuständen bezeichnet wird. Die elektrische Spannung oder die elektrische Stromstärke kann beliebige Zwischenzustände annehmen, ebenso wie die Temperatur. Man könnte analog auch mit stufenlos übersetzen und digital synonym mit gestuft.

Die zwei Zustände eines Stromkreises – Strom fließt oder Strom fließt nicht – lassen sich mit beliebigen Bedeutungen versehen, so wie uns dies im Alltag allenthalben begegnet: Eine Lampe an einem Backofen signalisiert, ob die gewünschte Heiztemperatur erreicht ist oder nicht; eine Lampe an einem Bahnübergang signalisiert, ob demnächst ein Zug queren wird oder nicht. Wollen wir mehr als zwei Fälle unterscheiden, können wir kurzerhand mehrere Stromkreise kombinieren; ein Paradebeispiel hierfür ist die Verkehrsampel.

Ganz allgemein können wir von einem Aktivitätsmuster sprechen, das ein oder mehrere Stromkreise erzeugen. Werden die Zustände der Stromkreise mittels Lampen sichtbar gemacht, so ergibt sich ein Lichtmuster. Dieses Muster steht für eine bestimmte Konfiguration der Stromkreise, die zur Codierung von Informationen verwendet werden kann. Welche Bedeutung welches Muster hat, ist eine Übereinkunft, die im Voraus getroffen werden muss; ansonsten lassen sich die Zeichen nicht deuten. In Abb. 2.2 sehen Sie z. B., wie sich die auf einer Lochkarte eingepprägten Informationen über ein entsprechendes Raster von Lampen darstellen lassen. Das Wissen über die

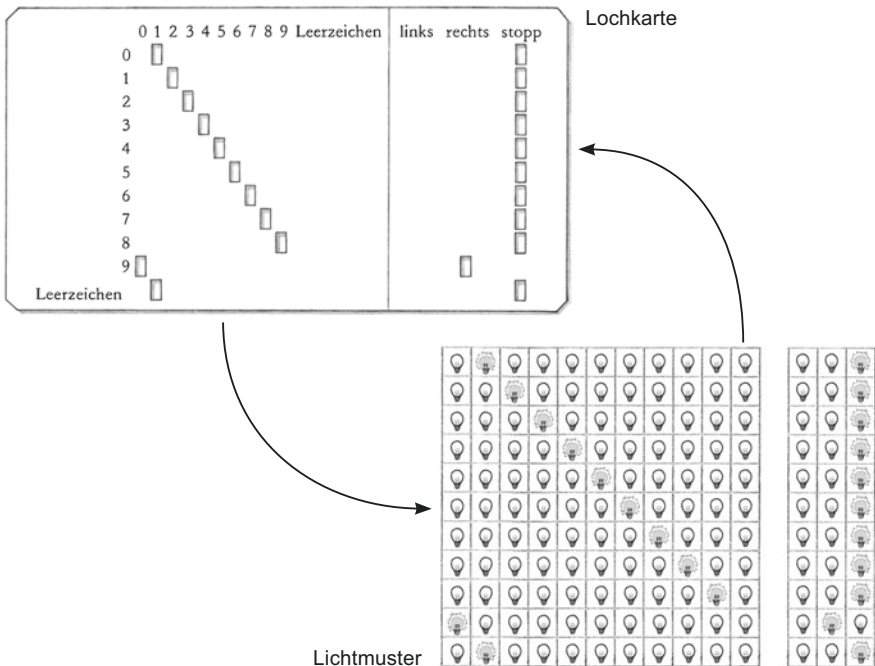


Abb. 2.2 Die gleiche Information, einmal in Form einer Lochkarte und einmal als Lichtmuster codiert. (© Susanne Staubli, Eckart Zitzler)

Anordnung der Stromkreise ist entscheidend, um das Lichtmuster interpretieren zu können.

Computer nutzen also Stromkreise, um Muster – auch Zeichenketten sind Muster – darzustellen. Doch wir sprachen ja von den Zahlen. Wie lassen sich mittels Lichtmustern Zahlen darstellen? Eine naheliegende Variante ist, dass die Anzahl der leuchtenden Lampen analog zu einer Strichliste die entsprechende Zahl repräsentiert. Diese Darstellungsweise erfordert enorm viele Stromkreise, wenn wir mit großen Zahlen zu tun haben; darüber hinaus eignet sie sich nicht besonders zum Rechnen. Viel kompakter und praktischer ist da das Zahlensystem, das wir selbst tagtäglich verwenden. Dieses lässt sich auf Lichtmuster übertragen. Doch um das zu erklären, muss ich etwas ausholen.

Von Lichtmustern zu Zahlen

Erinnern Sie sich noch daran, wie Sie zählen gelernt haben, als Anzahlen Namen bekommen haben? Ihre Mutter zeigte Ihnen einen Haufen von drei Äpfeln und nannte das „Drei“; ein anderes Mal waren es fünf Äpfel und sie nannte das „Fünf“. Später wurden aus den Lautfolgen für Drei und Fünf visuelle Symbole, die Sie zu schreiben lernten: „3“ und „5“. Irgendwann wussten Sie: Kleinen Anzahlen von bis zu neun Dingen weisen wir die Symbole „0“, „1“, „2“, „3“, „4“, „5“, „6“, „7“, „8“ und „9“ zu. Es hätten stattdessen auch die Symbole „A“, „B“, „C“ etc. sein können oder anderweitige Fantasiezeichen. Es sind nun einmal die uns bekannten Ziffern geworden, und daran haben wir uns gewöhnt.

Als Ihnen diese Zahlenwelt zu klein wurde, machten Sie Bekanntschaft mit dem Dezimalsystem, einem sogenannten Stellenwertsystem, bei dem wir auf der Basis unserer zehn Ziffern jede beliebige (An)Zahl ausdrücken können. Sind beim Hochzählen alle Ziffern ausgeschöpft, fügen wir einfach links eine weitere Ziffer an. Sie bezeichnet die Anzahl, wie häufig wir bei der rechten Stelle schon durchgezählt haben, die Anzahl der Umrundungen sozusagen. Nach 9 kommt also 10, logisch. Letztlich wird eine Zahl also durch eine Folge von Ziffern dargestellt, sie setzt sich aus mehreren Stellen zusammen. Die Position der Ziffer innerhalb dieser Folge bestimmt die Wertigkeit, d. h. wie viel die Ziffer zur Gesamtzahl beiträgt: Die Zahl ganz rechts zählt einfach, die nächste Stelle links zählt zehnfach, also die Zehner, die dritte Stelle steht für die Hunderter usw.

Das Prinzip hinter dem vertrauten Dezimalsystem lässt sich nun verallgemeinern, sodass wir auch mit weniger oder mehr als zehn Grundsymbolen Zahlen darstellen können. Abb. 2.3 führt Beispiele auf, bei denen zwei, vier

	0	0			A
	1	1			B
	2	10			C
	3	11			D
	4	100			E
	5	101			F
	6	110			G
	7	111			H
	8	1000			I
	9	1001			J
	10	1010			K
	11	1011			L

Abb. 2.3 Das Dezimalsystem verallgemeinert: verschiedene Stellenwertsysteme.
(© Susanne Staubli, Eckart Zitzler)

und zwölf Grundsymbole verschiedener Art vorliegen; die mittlere Spalte mit den Lampen können Sie für den Moment außer Acht lassen. In allen Fällen ist das Zählschema dasselbe. Wir benutzen zunächst die vorhandenen Symbole, um die ersten Anzahlen darzustellen. Die Symbole sind entsprechend geordnet, besitzen also eine Reihenfolge, und das erste Symbol verkörpert die Null. Sobald wir größere Anzahlen beschreiben wollen, hängen wir wie gewohnt links eine Stelle an und setzen die Grundsymbole in der definierten Reihenfolge ein. Wie das konkret funktioniert, erläutert Bsp. 2.1 exemplarisch für das Dualsystem – dem Zahlensystem, das nur zwei Ziffern kennt.

Beispiel 2.1: Das Dualsystem

Stehen uns nur zwei Symbole zur Verfügung, z. B. die Ziffern „0“ und „1“, dann geht Zählen so: 0, 1, 10, 11, 100, 101 usw. Diese Zählweise ist in Abb. 2.3 in der vierten Spalte von rechts zu sehen. Mit jeder weiteren Stelle verdoppelt sich der Wertigkeit – genauso wie sie sich in unserem Dezimalsystem jeweils verzehnfacht. Das heißt: Die Ziffernfolgen 10, 100, 1000, 10000 stehen bei der Verwendung von zwei Ziffern für die Anzahlen zwei, vier, acht, sechzehn,

während bei der Verwendung von zehn Ziffern die Anzahlen zehn, hundert, tausend und zehntausend ausgedrückt werden.

Ein Stellensystem, das nur zwei Symbole bzw. Ziffern aufweist, nennt man Dualsystem, und die so dargestellten Zahlen bezeichnet man als Dualzahlen oder auch als binäre Zahlen.

Es spielt letztlich keine Rolle, ob wir zehn Ziffern wie in unserem Dezimalsystem oder zwei wie im Dualsystem verwenden, es sind zwei gleichwertige Darstellungsarten von Anzahlen. Wie Sie in Abb. 2.3 sehen, bietet sich für unsere elektrische Rechenmaschine das Dualsystem an: Binäre Zahlen lassen sich eins zu eins durch Lichtmuster repräsentieren. Jede Lampe bzw. jeder Stromkreis stellt eine Ziffer dar und ist gleichzeitig mit einer festgelegten Stelle assoziiert: Brennt eine Lampe nicht, so steht dort eine 0, brennt die Lampe, dann steht dort eine 1. Der einzige Unterschied zur üblichen Zahlendarstellung besteht darin, dass die Anzahl der Stellen fixiert ist und die führenden Nullen sozusagen mit aufgeführt werden. Verwenden wir vier Stellen wie in Abb. 2.3, dann codiert das Lichtmuster für die Dualzahl 1 genau genommen die Ziffernfolge 0001, die Dualzahl 111 wird in Form von Stromkreisen als 0111 dargestellt.

Die Zahlenwelt des Computers besteht so gesehen aus Nullen und Einsen. Erstaunlich ist, dass diese Welt viel älter ist als der erste Computer, es gab sie schon lange vor Babbages Überlegungen. Das erste Zahlensystem mit nur zwei Zeichen wurde bereits gut 200 Jahre vor unserer Zeitrechnung ersonnen, auch wenn es da noch keine Null gab. Der deutsche Philosoph und Mathematiker Gottfried Wilhelm Leibniz war es, der schließlich Anfang des 18. Jahrhunderts das Dualsystem veröffentlichte und dokumentierte (Abb. 2.4). Damit legte er den Grundstein für das Rechnen mit Strom.



Abb. 2.4 Gottfried Wilhelm Leibniz. (© Camille Büsser)

Komm, wir bauen einen Taschenrechner!

Jetzt können wir die besagte elektrische Rechenmaschine konzipieren, die zunächst eine einzige Rechenoperation beherrschen soll: das Zusammenzählen zweier Zahlen. Dazu müssen wir uns erst einmal vor Augen führen, wie Dualzahlen addiert werden. Im Grunde genommen ist das Vorgehen identisch zu dem, wie wir handschriftlich zwei Dezimalzahlen addieren, nämlich stellenweise von rechts nach links. Der einzige Unterschied ist, dass wir beim Zusammenzählen zweier Ziffern neue Regeln lernen müssen, denn $1 + 1$ ergibt nicht mehr 2, sondern 10. Wem das noch etwas spanisch vorkommt, der möge einen Blick auf Bsp. 2.2 werfen; dort werden die Schritte im Detail erklärt.

Beispiel 2.2: Wie man zwei Dualzahlen addiert

In Abb. 2.5 sehen Sie im Vergleich, wie die gleichen Zahlen (aber mit unterschiedlicher Darstellung!) im Dezimalsystem und im Dualsystem handschriftlich miteinander addiert werden. Konzentrieren wir uns auf die Dualzahlen auf der rechten Seite: Genauso wie beim Dezimalsystem schreiben wir die zwei Summanden untereinander, und zwar rechtsbündig. Dann gehen wir wie gewohnt ziffernweise von rechts nach links. Wir addieren die zwei Ziffern an der aktuellen Stelle, wobei es zwei Möglichkeiten gibt: Die Summe ist mit einer Ziffer darstellbar oder es tritt ein Übertrag auf. Wenn ein Übertrag auftritt, vermerken wir diesen an der nächsten, weiter links stehenden Stelle. Schließlich schreiben wir die letzte Ziffer der Summe nach unten und gehen dann zur nächsten Stelle.

Beim Addieren der zwei oder drei Ziffern an einer Stelle müssen wir jedoch die Zählweise des Dualsystems berücksichtigen, denn $1 + 1 = 10$ (auf 1 folgt 10) und $1 + 1 + 1 = 11$. Was bleibt, ist: $0 + 0 = 0$, $1 + 0 = 1$ und $0 + 1 = 0$.

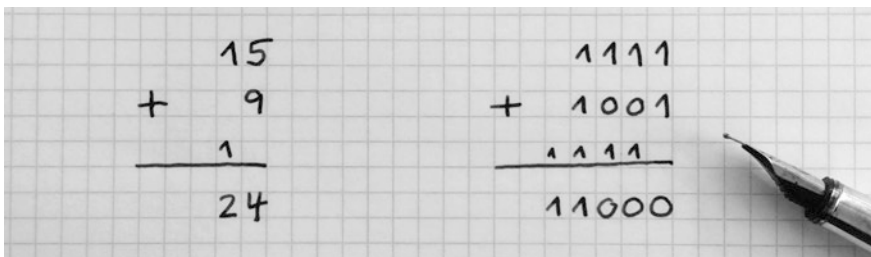


Abb. 2.5 Handschriftliches Addieren zweier Dezimalzahlen (links) und zweier Dualzahlen (rechts). (© Eckart Zitzler)

Die elektrische Rechenmaschine, die wir im Folgenden konstruieren werden, können Sie sich als einen Taschenrechner aus der Computersteinzeit vorstellen, der das handschriftliche Addieren nachbildet. Dieser Taschenrechner ist äußerst spartanisch ausgestattet und weist drei Lampenreihen auf, die die zwei Summanden und das Ergebnis wiedergeben (siehe Abb. 2.6). Wie auf dem Papier stehen die beiden Summanden untereinander, beide haben maximal fünf Stellen, jede Stelle wird durch eine Lampe repräsentiert. Die einzelnen Ziffern beider Summanden können wir mit Schaltern einstellen, sodass die entsprechende Lampe leuchtet (stellvertretend für eine 1) oder nicht (stellvertretend für eine 0). In der untersten Zeile steht schließlich das Ergebnis, ebenfalls durch eine Reihe von fünf Lampen dargestellt. Das Ergebnis wird immer unmittelbar angezeigt, sobald wir Änderungen an den Eingabe-Schaltern vornehmen. Zudem werden auch die Überträge angezeigt, sofern solche auftreten: Die kleinen Lampen über dem Strich stehen für Überträge, und auch hier bedeutet eine leuchtende Lampe, dass es einen Übertrag gibt.

So simpel der Taschenrechner auch sein mag: Einiges von dem, was einen ausgewachsenen Computer ausmacht, können wir bereits wiederfinden. Zum Beispiel gibt es eine Eingabe, die über Schalter gesteuert wird, und es gibt eine Ausgabe in Form von Lampen. Bei den ersten Computern

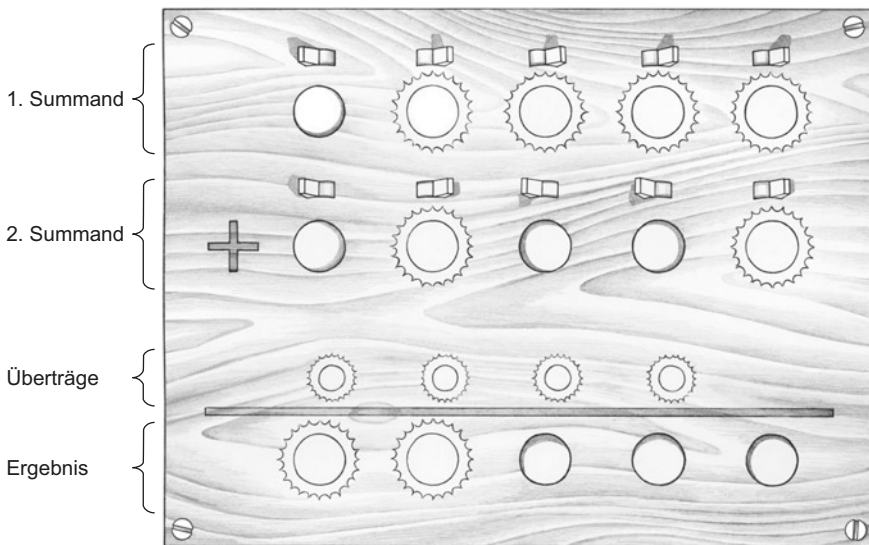


Abb. 2.6 Ein fiktiver minimalistischer Taschenrechner zur Addition zweier Dualzahlen, deren Ziffern durch Lampen dargestellt werden. (© Susanne Staubli, Eckart Zitzler)

war diese Art der Ein- und Ausgabe üblich, und auch einige Jahrzehnte später blieb dieses Prinzip erhalten. Eine Tastatur ist ja nichts anderes als eine Anordnung von Schaltern, während ein Bildschirm als eine gitterartige Anordnung von einer Vielzahl winziger Lichtquellen verstanden werden kann.

Bevor wir uns dem Innenleben des Taschenrechners zuwenden, unternehmen wir noch einen Abstecher in die Biologie. Denn auch dort spielen Muster eine wichtige Rolle.

Geniale Muster

Wie wir gesehen haben, sind in einem Computer Informationen, insbesondere Zahlen, als Aktivitätsmuster von Stromkreisen codiert – nicht nur die Ein- und Ausgabe, sondern jegliche Art von Information, die während der Verarbeitung entsteht. Letztlich können wir uns einen Computer als ein riesiges Netz von Stromkreisen vorstellen, in dem codierte Informationen hin- und herfließen. Würden wir die Gesamtaktivität aller Stromkreise über Lampen sichtbar machen, dann ließe sich der Berechnungsvorgang als eine wilde Folge von Lichtmustern wahrnehmen, etwa so, als würden wir eine Großstadt wie Tokio nachts aus der Vogelperspektive betrachten. Jedes Lichtmuster spiegelte den aktuellen Zustand des Computers wider: welche Daten er verarbeitet, welchen Rechenschritt er ausführt usw. Auch in lebenden Zellen, seien es nun unsere Körperzellen oder einzellige Organismen wie die Kolibakterie, finden wir solche Aktivitätsmuster wieder, und auch sie repräsentieren Informationen, nämlich über die Außen- und die Innenwelt dieses biologischen Systems.

Nehmen wir beispielsweise die Proteine, von denen bereits in Kap. 1 die Rede war. Mit Hinblick auf die Informationsverarbeitung sind sie für die Zelle das, was die Stromkreise für den Computer bedeuten. Proteine sind nicht nur Akteure, die die biologischen „Rechenoperationen“ verkörpern, sondern auch Zeichenträger: Sie können für Signale stehen, die direkt oder indirekt verschiedene Aktionen auslösen können. Daher setzt die Zelle ihre Proteine effizient und gezielt ein, um die zum Leben notwendigen Vorgänge abzuwickeln; jedes Protein wird nur in dem Maße produziert, wie es gerade benötigt wird. Das Vorhanden- oder Nichtvorhandensein eines Proteins entspricht so gesehen der Lampe beim Stromkreis, die dunkel ist bzw. leuchtet. Ganz so schwarz-weiß ist es im Leben allerdings nicht. Die Lampe muss verschiedene Helligkeitsstufen annehmen können, denn der Herstellungsprozess eines Proteins kann ja graduell reguliert werden, er ist nicht einfach nur „an“ oder „aus“.

Zu verschiedenen Zeitpunkten können folglich ganz unterschiedliche Proteine in einer Zelle aktiv sein. Die Aktivitäten aller Proteine in der Zelle bilden ebenfalls ein Muster, welches mit entsprechenden experimentellen Methoden sichtbar gemacht werden kann (Bsp. 2.3). Das Resultat kommt dem Bild eines Lichtmusters recht nahe, wie Sie in Abb. 2.7 auf der rechten Seite sehen können; genauso wie beim Computer repräsentiert es diverse Informationen über das aktuelle Innenleben der Zelle. Auch wenn es sich dabei immer nur um einen Ausschnitt handelt: Für Biologen ermöglichen solche Schnappschüsse wertvolle Einsichten, sie liefern Anhaltspunkte dafür, was genau im zellulären Raum abläuft. Die Kernfrage ist, wie die zugrunde liegenden biologischen Prozesse funktionieren, die von einem Aktivitätsmuster zu einem anderen Aktivitätsmuster führen.

Beispiel 2.3: Biologische Chips

Die Aktivität eines Proteins lässt sich über verschiedene Methoden ermitteln, bei der hier vorgestellten Technik geschieht das indirekt. Man bestimmt, wie häufig der entsprechende Bauplan auf der DNA abgelesen wird, und zwar über die Anzahl der Abschriften, die zu einem bestimmten Zeitpunkt in der Zelle vorhanden sind. Dazu filtert man zunächst alle Moleküle heraus, die Abschriften von Proteinbauplänen darstellen. Diese Moleküle werden anschließend auf ein fingernagelgroßes Plastik- oder Glasplättchen aufgetragen, das sogenannte Microarray. Das Microarray enthält eine Vielzahl von rasterartig angeordneten Bindestellen, von denen jede spezifisch für ein bestimmtes Ablese-Molekül ist. Die Ablese-Moleküle haften an den für sie vorgesehenen Bindestellen fest.

Die Menge an Ablese-Molekülen, die sich an den einzelnen Bindestellen angelagert hat, kann mittels Laserlicht abgelesen werden. Das Ergebnis liegt schließlich in Form eines Punkterasters vor, jeder Punkt steht für eine Bindestelle. Je heller ein Punkt, desto mehr Ablese-Moleküle haben sich an der entsprechenden Bindestelle angelagert, desto höher war die Produktion des entsprechenden Proteins. Der Sachverhalt ist in Abb. 2.7 visualisiert. Microarrays dieser Art werden auch Gen-Chips genannt; sie enthalten in der Regel viele Tausend Bindestellen.

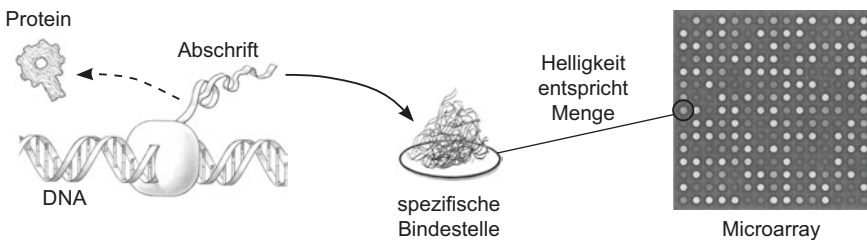


Abb. 2.7 Wie häufig der Bauplan eines Proteins abgelesen wird, lässt sich mittels Microarrays sichtbar machen. (© Susanne Staubli, Eckart Zitzler)

Der gleichen Frage werden wir uns jetzt aus der Informatikperspektive stellen: Wie werden aus Eingabemustern, die durch eine Kombination aus Stromkreisen repräsentiert werden, spezifische Ausgabemuster, die durch einen Satz anderer Stromkreise dargestellt werden? Oder anders herum: Wie müssen die Eingabestromkreise mit den Ausgabestromkreisen gekoppelt werden, damit die gewünschte Berechnung stattfindet?

Kleines Ding, große Wirkung

Zurück zu unserem Taschenrechner. Richten wir unsere Aufmerksamkeit für den Moment auf die rechte Ziffernspalte, als auf das Zusammenzählen von zwei einzelnen Dualziffern. In diesem Fall wird das Eingabemuster aus den zwei Lampen der Summanden erzeugt, das Ausgabemuster bilden die Lampen für die Ergebnisziffer und den ersten Übertrag. Für jedes der vier möglichen Eingabemuster möchten wir dasjenige Ausgabemuster erhalten, welches sich aus der Additionsvorschrift ergibt. Die Aufgabe des Taschenrechners besteht also darin, eine bestimmte Zuordnung von Eingabemustern zu Ausgabemustern umzusetzen.

In Abb. 2.8 ist die Situation grafisch dargestellt. Rechts sehen Sie die zu realisierende Musterzuordnung, wobei die linke Ausgabelampe für den Übertrag und die rechte für die Ergebnisziffer steht. In der Mitte ist das

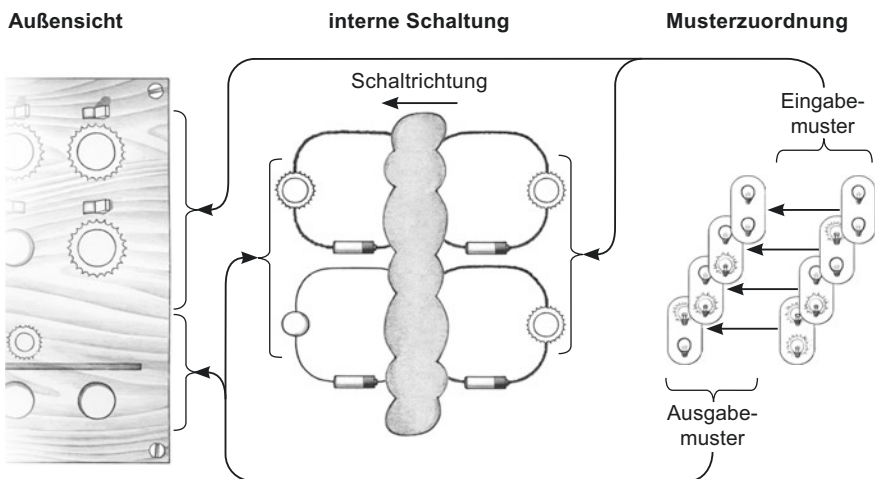


Abb. 2.8 Wie der Taschenrechner im Innern prinzipiell funktioniert. (© Susanne Staubli, Eckart Zitzler)

Innere des Taschenrechners schematisiert: Zwei Eingabestromkreise werden irgendwie – für das Irgendwie steht die Wolke – mit zwei Ausgabestromkreisen verknüpft. Man kann auch sagen, sie werden verschaltet, und das Gesamte ergibt dann eine Schaltung. Bitte beachten Sie, dass die Eingabestromkreise ohne Schalter dargestellt sind und sich stets auf der rechten Seite der Schaltung befinden; die noch kommenden Illustrationen werden dadurch übersichtlicher und einfacher.

Im Folgenden geht es um die Wolke in Abb. 2.8: darum, was sich dahinter versteckt, was darin passiert. Erst einmal werde ich Ihnen die grundlegenden Ideen erläutern. In einem späteren Schritt werden Sie dann die allgemeine Vorgehensweise kennenlernen, nach der solche Typen von Schaltungen – man bezeichnet sie als digitale Schaltungen – entwickelt werden.

Elektrischer Staffellauf

Wollen wir den Zustand eines Stromkreises an die Zustände anderer Stromkreise koppeln, brauchen wir ein Verbindungselement, über das Zustandsinformationen übertragen werden können. Das Bauteil, nach dem wir suchen, ist ein Schalter, der nicht von Hand, sondern über Strom gesteuert wird. So eine Art Schalter wurde bereits 1835 erfunden und war ein Schlüsselement in der Telegrafie, um weite Distanzen zu überbrücken. Die Rede ist vom Relais.

Ein Relais besteht einerseits aus einem Elektromagneten – ein mit Draht umwickelter Eisenkern – und andererseits aus einem magnetischen Schalter (Abb. 2.9). Ersterer ist in den Stromkreis eingebunden, der steuert, Letzterer in den Stromkreis, der gesteuert wird. Sobald der Elektromagnet von Strom durchflossen wird, baut sich ein Magnetfeld auf, über das der magnetische Schalter bewegt wird: Entweder schließt er den Kontakt – dann sprechen wir von einem schließenden Relais – oder er unterbricht den Kontakt – dann reden wir von einem öffnenden Relais. So weit, so gut. Das Relais ist allerdings ein altertümliches Beispiel für einen Schalter, der selbst durch Strom betätigt wird; in der Computertechnologie ist es schon längst durch den Transistor abgelöst worden, mehr dazu in Anm. 2.2. Wir bleiben im Folgenden beim Relais, weil es einfacher zu verstehen ist als der Transistor.

Symbol



Bauweise

schließendes Relais

öffnendes Relais

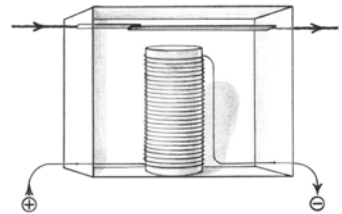
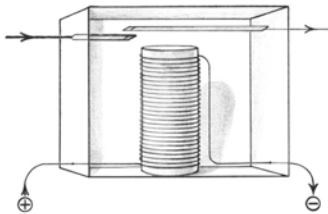
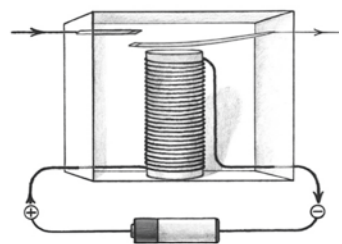
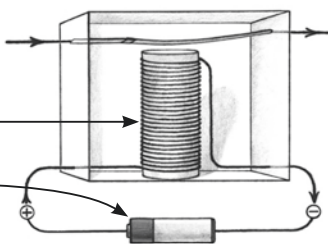
Relais
deaktiviertRelais
schaltet

Abb. 2.9 Zwei Arten von Relais. (© Susanne Staubli, Eckart Zitzler)

Anmerkung 2.2: Vom Relais zum Transistor

Das Wort Relais heißt eigentlich Vermittler oder im Französischen auch Staffel. Eine recht treffende Bedeutung, denn tatsächlich lassen sich mit Relais Ketten von Stromkreisen konstruieren: Ein erster Stromkreis schaltet mit einem Relais einen zweiten, dieser wiederum steuert über ein weiteres Relais einen dritten Stromkreis usw. Tatsächlich hat man das in der Telegrafie genutzt, um elektrische Signale zu verstärken. Da das elektrische Signal jeweils nur für gewisse Distanzen stark genug war, schaltete man ein Relais – einen Mittler – dazwischen, und das aktivierte einen weiteren Stromkreis mit einer eigenen Spannungsquelle, der das Signal weiterreichte. Später waren Relais dann in Telefonanlagen im Einsatz, um Verbindungen zwischen entfernten Telefonapparaten herzustellen (Abb. 2.10).

Der erste Computer – die Z3 von Konrad Zuse – basierte auf dieser Technologie, doch schon bald wurden Relais durch Elektronenröhren und später Transistoren verdrängt. Letztere arbeiten zum einen schneller und zuverlässiger; zum anderen können Milliarden von Transistoren auf einem daumennagelgroßen Chip untergebracht werden, der Platzbedarf eines Relais ist demgegenüber um Größenordnungen höher. Transistoren – ein paar Beispiele können Sie in Abb. 2.10 sehen – beruhen auf der Halbleitertechnologie, die der Computerentwicklung einen enormen Schub gegeben hat.

Mit dem Relais können wir zwei Stromkreise verbinden und den einen unter die Kontrolle des anderen stellen. In Abb. 2.11 sind die zwei

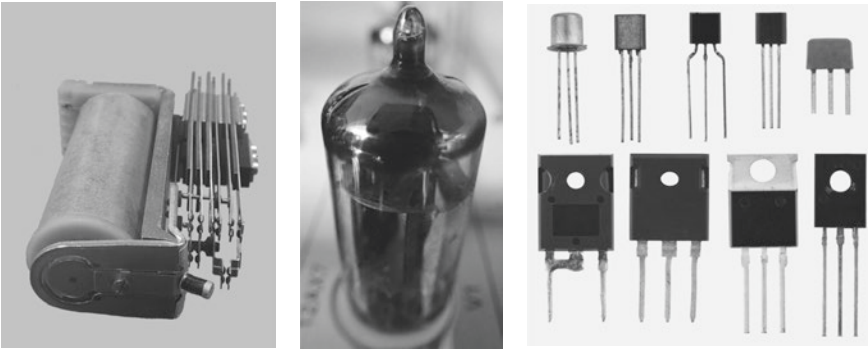


Abb. 2.10 Ein Relais (links), eine Elektronenröhre (Mitte) und verschiedene Transistoren als Einzelbauteile (rechts). (© Eckart Zitzler (links und Mitte, mit freundlicher Genehmigung des Museums ENTER Solothurn), pioneer111/[stock.adobe.com](https://www.adobe.com) (rechts))

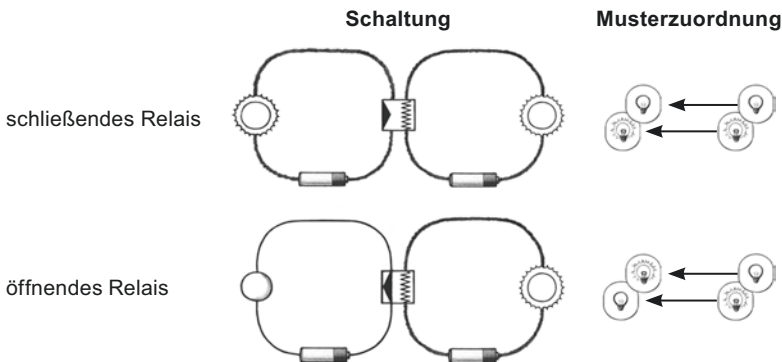


Abb. 2.11 Zwei Möglichkeiten, zwei Stromkreise miteinander zu verschalten; die Eingabeschaltkreise sind ohne Schalter gezeigt und befinden sich jeweils auf der rechten Seite. (© Susanne Staubli, Eckart Zitzler)

Möglichkeiten abgebildet, die sich aus der Verwendung der zwei Relaisarten ergeben. Entweder weist der Ausgabestromkreis ein übereinstimmendes oder ein entgegengesetztes Verhalten zum Eingabestromkreis auf. Anders ausgedrückt: Bei einem schließenden Relais wird aus 0 wieder 0 und aus 1 wieder 1, bei einem öffnenden Relais wird aus einer 0 eine 1 und umgekehrt. So einfach das ist, es ist bereits die Grundlage der elementaren Textersetzung, dem Prinzip von Turings fiktiver Rechenmaschine. Wir können gezielt eine Ziffer durch eine beliebige andere – nun ja, im Dualsystem stehen ja nur zwei zur Verfügung – ersetzen.

Die Kopplungsmöglichkeiten sind jedoch nicht auf zwei Stromkreise beschränkt. Indem wir mehrere Relais verbauen, können auch mehrere Stromkreise auf einen Stromkreis Einfluss nehmen. Dabei gibt es verschiedene Methoden der Verknüpfung – genauso wie es beim Stricken, Häkeln oder Teppichknüpfen verschiedene Techniken gibt.

Eine kleine Knüpfkunde

Die drei Kopplungsprinzipien, die ich Ihnen nachfolgend zeigen werde, sind das Handwerkszeug für das Knüpfen von Stromkreisnetzen. Ich werde die Idee jeweils an Situationen mit zwei Eingabestromkreisen und einem Ausgabestromkreis illustrieren.

Beim ersten Kopplungsprinzip, der Reihen-Verschaltung, werden die Relais, über die die Eingabestromkreise auf den Ausgabestromkreis Einfluss nehmen, hintereinander angeordnet; nur wenn die magnetischen Kontakte aller Relais geschlossen sind, kann im Ausgabestromkreis Strom fließen. Je nachdem, welchen Relaisstyp wir pro Eingabestromkreis wählen, gibt es verschiedene Verschaltungsvarianten (siehe Bsp. 2.4). Mit dieser Art der Verschaltung lässt sich bewirken, dass die Ausgangslampe nur bei einem ganz bestimmten Eingangsmuster leuchtet.

Beispiel 2.4: Die Reihen-Verschaltung

Abb. 2.12 demonstriert, wie die Relais bei einer Reihen-Verschaltung innerhalb des Ausgabestromkreises angeordnet werden. Verwenden wir ausschließlich schließende Relais, so leuchtet die Ausgangslampe genau dann, wenn auch beide Eingangslampen leuchten. Nehmen wir stattdessen öffnende Relais, so leuchtet die Ausgangslampe einzig, wenn beide Eingangslampen nicht leuchten. Wir können auch beide Relais Typen miteinander kombinieren (nicht in der Abbildung zu sehen): Die Ausgangslampe leuchtet in dem Fall, wenn die eine Eingangslampe leuchtet und die andere nicht.

Beim zweiten Kopplungsprinzip werden die Relais der Eingabestromkreise nicht hintereinander, sondern nebeneinander angeordnet (Bsp. 2.5). Das nennt man eine Parallel-Verschaltung. In diesem Fall kann jeder Eingabestromkreis den Ausgabestromkreis unabhängig vom anderen Eingabestromkreis aktivieren: Nur eines der beiden Relais muss geschlossen sein, damit die Ausgangslampe leuchtet. Auf diese Art und Weise lassen sich Alternativen festlegen.

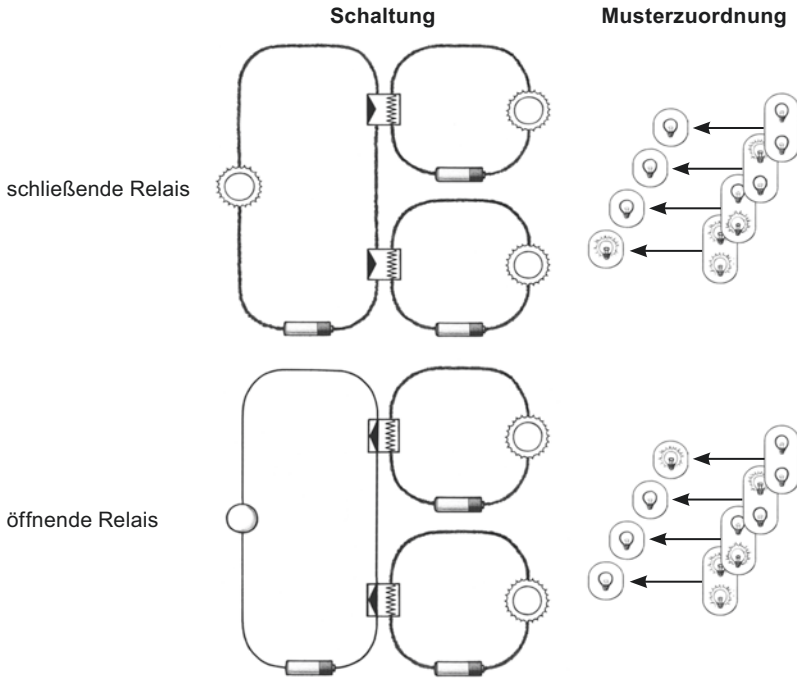


Abb. 2.12 Einfache Reihen-Verschaltungsvarianten von zwei Stromkreisen, die einen dritten Stromkreis ansteuern. (© Susanne Staubli, Eckart Zitzler)

Beispiel 2.5: Die Parallel-Verschaltung

In einer Parallel-Verschaltung sind die Relais über eigenständige Schleifen in den Ausgabestromkreis integriert; jedes einzelne Relais kann den Ausgabestromkreis schließen. In Abb. 2.13 sehen Sie zwei mögliche Varianten dieser Verschaltungsart, einmal mit schließenden und einmal mit öffnenden Relais. Im ersten Fall leuchtet die Lampe, wenn eine der beiden Eingangslampen leuchtet, im anderen Fall leuchtet der Ausgang, wenn eine der beiden Eingangslampen nicht leuchtet. Auch hier können wieder schließende und öffnende Relais kombiniert werden, sodass die Ausgangslampe genau dann **nicht** leuchtet, wenn die eine Eingangslampe leuchtet, aber die andere nicht.

Das dritte Kopplungsprinzip besteht darin, die vorherigen zwei miteinander zu kombinieren. Zunächst bilden wir Gruppen von Relais, die in Reihen hintereinander angeordnet werden. Diese Reihen werden schließlich mittels einer Parallel-Verschaltung in den Ausgabestromkreis eingebaut. Der Ausgabestromkreis ist geschlossen, wenn mindestens eine der

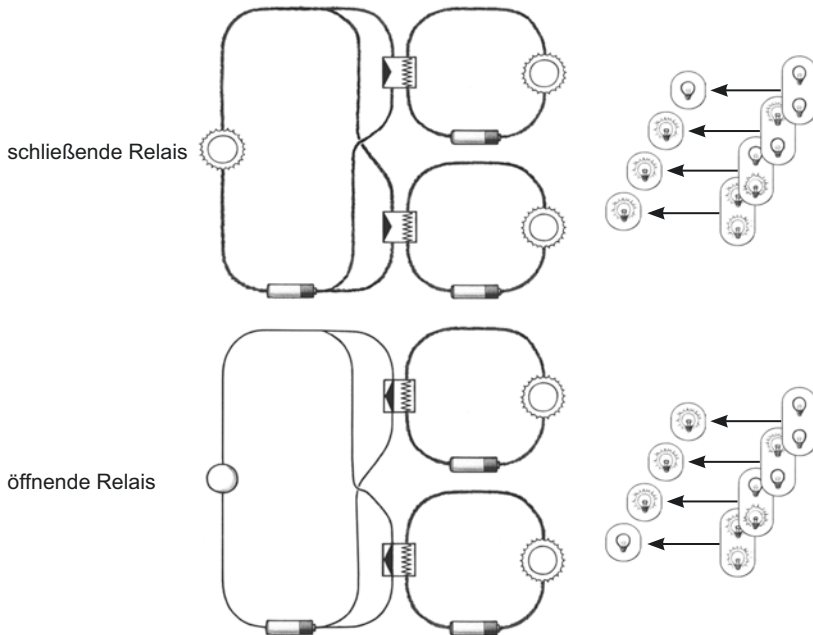


Abb. 2.13 Einfache Parallel-Verschaltungsvarianten von zwei Stromkreisen, die einen dritten Stromkreis ansteuern. (© Susanne Staubli, Eckart Zitzler)

Reihen-Verschaltungen schließt. Diese Arbeitsweise ist nicht mehr so einfach ersichtlich, Bsp. 2.6 hilft beim Verstehen. Über die Kombinations-Verschaltung können wir also einen Satz an Eingabemustern definieren, bei denen die Ausgangslampe leuchten soll.

Beispiel 2.6: Reihen- und Parallel-Verschaltung kombiniert

Wenn Sie einen Blick auf Abb. 2.14 werfen, sehen Sie zwei Relais-Paare, bei denen jeweils ein schließendes mit einem öffnenden Relais in Reihe geschaltet ist. Die beiden Reihen wiederum sind parallel zueinander verschaltet. Jeder Eingabestromkreis steuert je ein schließendes und ein öffnendes Relais an und beeinflusst so beide Reihen-Verschaltungen.

Was ist nun das Schaltverhalten? Das eine Relais-Paar schließt genau dann, wenn der eine Eingabestromkreis aktiv ist und der andere nicht. Beim anderen Relais-Paar ist es genau umgekehrt: Es schließt, wenn der erste Eingabestromkreis inaktiv und der zweite aktiv ist. Folglich leuchtet die Lampe im Ausgabestromkreis, wenn genau eine der Eingabelampen leuchtet.

Etwas spröde, das Ganze? Seien Sie beruhigt: Es fehlt nur noch ein Schritt und wir haben das Rüstzeug, um mit Strom zu rechnen.

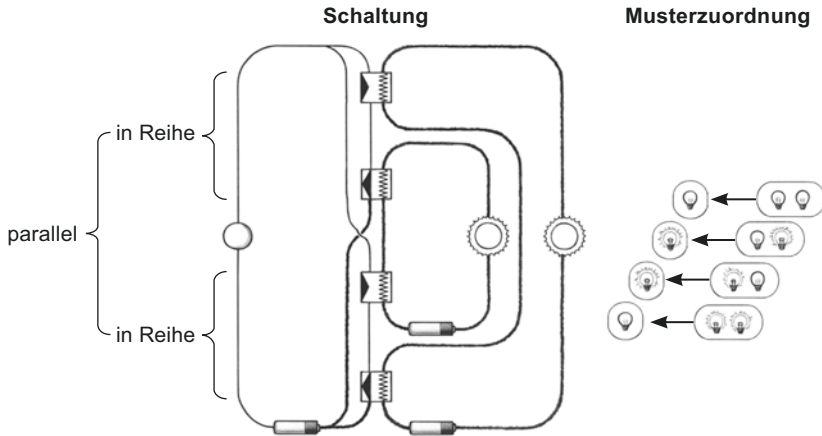


Abb. 2.14 Kombination von Reihen- und Parallel-Verschaltungsvarianten von zwei Stromkreisen, die einen dritten Stromkreis ansteuern. (© Susanne Staubli, Eckart Zitzler)

Viele mit vielen koppeln

Die Verschaltungstechniken, die wir soeben kennengelernt haben, lassen sich auch einsetzen, wenn mehr als zwei Eingabestromkreise denselben Ausgabestromkreis steuern sollen. Damit können wir beliebige Kombinationen von Eingabeziffern auf eine einzige Ausgabeziffer abbilden. Doch was ist, wenn mehrere Ausgabestromkreise bedient werden müssen?

Die Lösung ist einfach: Wir betrachten jeden Ausgabestromkreis separat. Zuerst stellen wir für jede Ausgabelampe die gewünschte Musterzuordnung auf und blenden dabei alle restlichen Ausgangslampen aus. Auf dieser Basis entwerfen wir für jeden Ausgabestromkreis eine Verschaltung mit den Eingabestromkreisen. Zum Schluss fügen wir die einzelnen Schaltungen zu einer Gesamtschaltung zusammen, in der die Eingabestromkreise mit mehreren Ausgabestromkreisen interagieren, die Ausgabestromkreise sich untereinander aber nicht beeinflussen. Das Vorgehen ist in Abb. 2.15 visualisiert für eine Situation mit zwei Eingabe- und zwei Ausgabestromkreisen.

Ausgezeichnet, jetzt sind wir so weit und können die Wolke in Abb. 2.8 füllen. Vergewenärtigen wir uns dafür noch einmal, was genau beim Addieren zweier Dualziffern zu tun ist. Die Lampe der Ergebnisziffer soll leuchten, wenn genau eine der Eingangslampen leuchtet; die Übertragslampe soll leuchten, wenn beide Eingangslampen leuchten. Das könnte Ihnen bekannt vorkommen. Diese zwei Musterzuordnungen sind

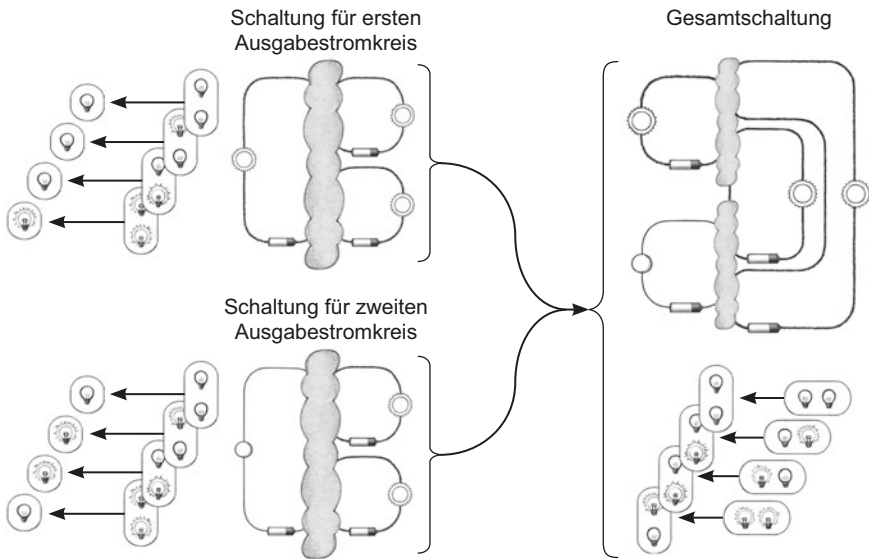


Abb. 2.15 Vorgehen, um eine Verschaltung mit mehreren Ausgabestromkreisen zu konstruieren. (© Susanne Staubli, Eckart Zitzler)

uns bereits in Abb. 2.14 und 2.12 oben über den Weg gelaufen. Die entsprechenden zwei Verschaltungen realisieren gemeinsam das Aufsummieren der niederwertigsten Summandenziffern, wie in Abb. 2.16 ersichtlich. Was noch bleibt, ist das Zusammenführen beider Verschaltungen in eine Gesamtschaltung – darauf werden wir später noch eingehen.

Sie sehen: Mittels einfacher Verschaltungen lässt sich rechnen. Wir müssen nur auf Turings Ideen zurückgreifen und Rechnen als einen abstrakten Vorgang begreifen, bei dem Eingabemuster auf Ausgabemuster abgebildet werden – das ist das Fundament der elektrischen Informationsverarbeitung. Wir werden uns gleich anschauen, wie sich beliebige Rechnungen mittels solcher elementarer Verschaltungen bewerkstelligen lassen, doch vorher entführe ich Sie wieder kurz in die Biologie.

Wie Proteine über Gene miteinander verknüpft sind

Auch Proteine sind miteinander gekoppelt und können sich gegenseitig beeinflussen. erinnern Sie sich noch an das Schalterprotein in Abb. 1.26, über das die Kolibakterie ihre Ernährungsweise steuern kann? Es ist ein Beispiel dafür, wie ein Protein das Ablesen eines Protein-Bauplans von der DNA und somit die Produktion des entsprechenden Proteins regulieren

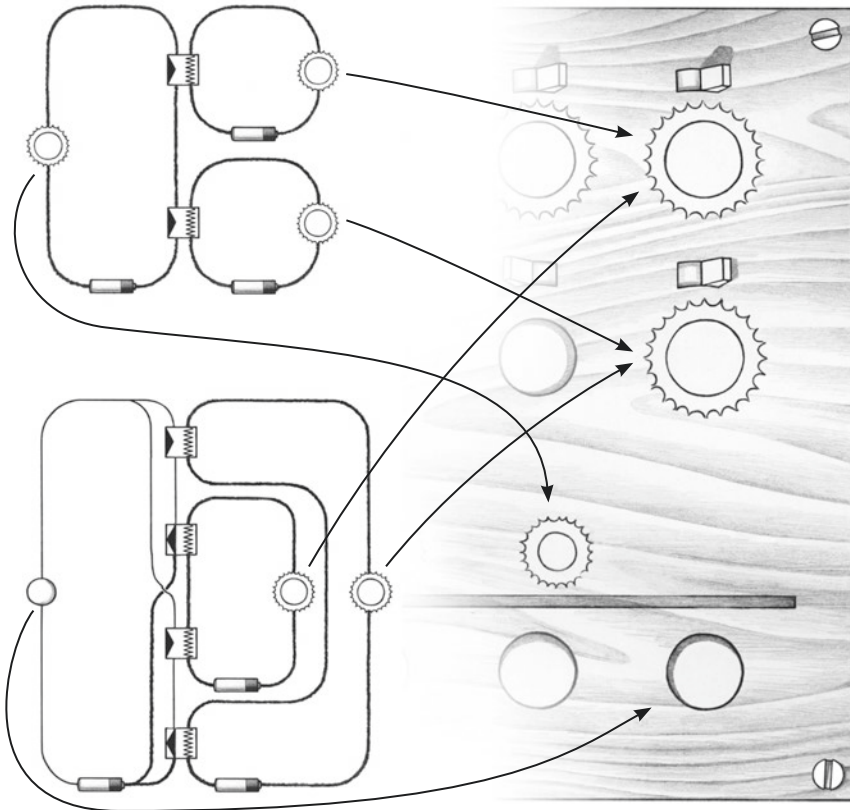


Abb. 2.16 So lassen sich bei unserem Mini-Taschenrechner die ersten, ganz rechts stehenden zwei Ziffern der beiden Summanden mittels zweier Verschaltungen addieren. (© Susanne Staubli, Eckart Zitzler)

kann – an sich das gleiche Prinzip wie bei den elektrischen Schaltungen: „Protein schaltet Protein“ entspricht „Strom schaltet Strom“.

Die Rolle des Verbindungsglieds übernehmen hierbei Gene. Ein Gen verknüpft Schalterproteine – Proteine, die sich am jeweiligen DNA-Abschnitt anlagern können – mit demjenigen Protein, dessen Bauplan das Gen codiert. Dabei stellt sich die Frage, wie solch eine Verknüpfung konkret aussieht. Woher wissen die Schalterproteine, wo sich das jeweilige Gen auf dem extrem langen DNA-Molekül befindet und an welche Stelle sie sich anheften sollen? Das Gleiche können wir uns auch bei der RNA-Polymerase fragen, der molekularen Maschine, die für das Erstellen der Abschrift zuständig ist. Schließlich besteht die DNA nicht nur aus Genen. In beiden Fällen lautet die Antwort: Es gibt spezielle Markierungen auf der DNA. Wie solche

das Schalterprotein als auch für das angesteuerte Gen. Mit anderen Worten: Binderegion und Bauplan definieren eine Leitung von Schalterprotein zu gesteuertem Protein. Vor einem Gen kann es sogar mehrere Binderegionen haben, d. h., mehrere Proteine können die Aktivität eines Gens beeinflussen. Leitungen sind also – einmal abgesehen davon, dass sie nur imaginär existieren – spezifisch in der Zelle, sie lassen sich nicht einfach umstöpseln.

Letztlich entstehen so ebenfalls Verschaltungen in der Zelle, Verschaltungen von Proteinen und Genen. In der Sprache der elektrischen Schaltungen repräsentieren Gene Leitungen und Kontakte, während Proteine Stromkreise und Elektromagneten verkörpern.

Was Lattentore mit Logik zu tun haben

Sie haben jetzt eine Vorstellung davon, wie sich mittels Relais Stromkreise verschalten und so bestimmte Musterzuordnungen realisieren lassen. Jetzt können wir uns daranmachen, das Innenleben des Taschenrechners zu erarbeiten. Dabei werde ich Ihnen zeigen, wie Profis Schaltungen entwerfen. Sie fassen Teilverschaltungen in Bausteinen zusammen, auf deren Basis sie dann größere Bausteine und komplexere Schaltungen realisieren. Schauen wir uns also an, was es mit den Bausteinen auf sich hat.

Von Stromkreisen zu Bausteinen

Die Vorstellung der sich gegenseitig beeinflussenden Stromkreise hilft dabei, die Arbeitsweise eines Computers auf der untersten Hardware-Ebene zu verstehen. Für den Entwurf von Schaltungen ist sie hingegen weniger geeignet. Sobald wir mit vielen miteinander gekoppelten Stromkreisen hantieren müssen, wird es unübersichtlich. Und es kommt noch hinzu, dass wir ja nicht für jeden einzelnen Stromkreis eine eigene Batterie einsetzen wollen. In der Praxis betrachtet man daher statt der separaten, in sich geschlossenen Stromkreise ein verzweigtes Stromnetz mit einer einzigen Spannungsquelle. Darüber hinaus werden zusammengehörige Relais in einem Kasten untergebracht, der als Baustein fungiert und die konkrete Verschaltung verbirgt. Abb. 2.18 illustriert die Idee und demonstriert, wie wir von gekoppelten Stromkreisen zu einem Stromnetz mit Bausteinen kommen.

Die Idee des Bausteins entspricht eigentlich dem Bild der Wolke in Abb. 2.8: Rechts befinden sich die Eingabeleitungen von den steuernden Stromkreisen, links die Ausgabeleitungen zu den gesteuerten Stromkreisen sowie unten und oben die Verbindungen zur Spannungsquelle. Sie können sich das so vorstellen,

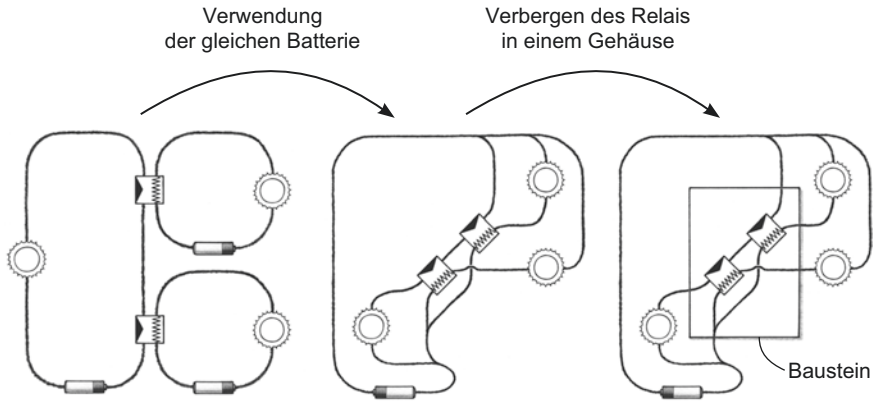
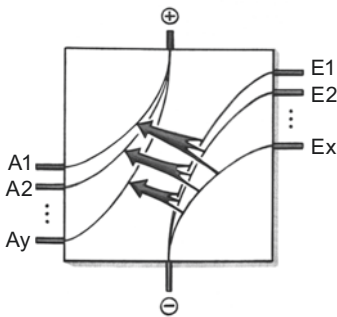


Abb. 2.18 Von einer Schaltung mit separaten Stromkreisen zu einem Stromnetz mit einem Baustein. (© Susanne Staubli, Eckart Zitzler)

Prinzip Baustein



Plus-Baustein

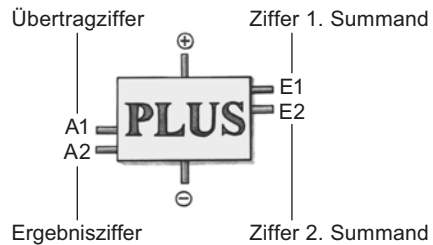


Abb. 2.19 Die Funktionsweise eines Bausteins (links) und ein einfacher Baustein zur Addition zweier Dualziffern (rechts). (© Susanne Staubli, Eckart Zitzler)

als würde der Strom rechts hineinfließen und links heraus. Genau genommen ist es jedoch so, dass der Strom, der über die Eingabeleitungen in den Baustein einfließt, über den unteren Batterieanschluss abfließt; er reguliert über die Relais, welche Ausgabeleitungen mit dem oberen Batterieanschluss verbunden und so mit Strom versorgt werden, wie in Abb. 2.19 links skizziert ist.

Entsprechend können wir nun einen Plus-Baustein konzipieren, der die Verschaltungen in Abb. 2.16 zusammenfasst und in einer Box verbirgt. Er nimmt die ersten beiden, am weitesten rechts stehenden Ziffern der zwei Summanden als Eingabe und gibt die erste Ziffer des Ergebnisses sowie einen möglichen Übertrag aus (Abb. 2.19 rechts). Wie das Innere im Detail aussieht, erläutert Bsp. 2.8.

Beispiel 2.8: Ein Plus-Baustein

In Abb. 2.20 sehen Sie oben die zwei Verschaltungen, die wir bereits entwickelt hatten. Führen wir diese in einem Baustein zusammen, so entsteht ein Relais-Leitungs-Wirrwarr, das nicht mehr so leicht zu durchschauen ist. Sie können die Bausteinkonstruktion nachvollziehen, wenn Sie die Beschriftungen der Leitungen und Relais oben und unten vergleichen. Wichtig ist vor allem, dass Ihnen die Funktionsweise des Bausteins klar ist: Wenn an den Eingängen das Muster 00 (auf beiden Eingangsleitungen liegt keine Spannung an) eingespeist wird, ist das Muster an den Ausgängen ebenfalls 00; 01 und 10 an den Eingängen führen zu 01 an den Ausgängen (die Übertragslampe ist ausgeschaltet); das Eingangsmuster 11 hingegen wird auf das Ausgangsmuster 10 (die Übertragslampe leuchtet) abgebildet.

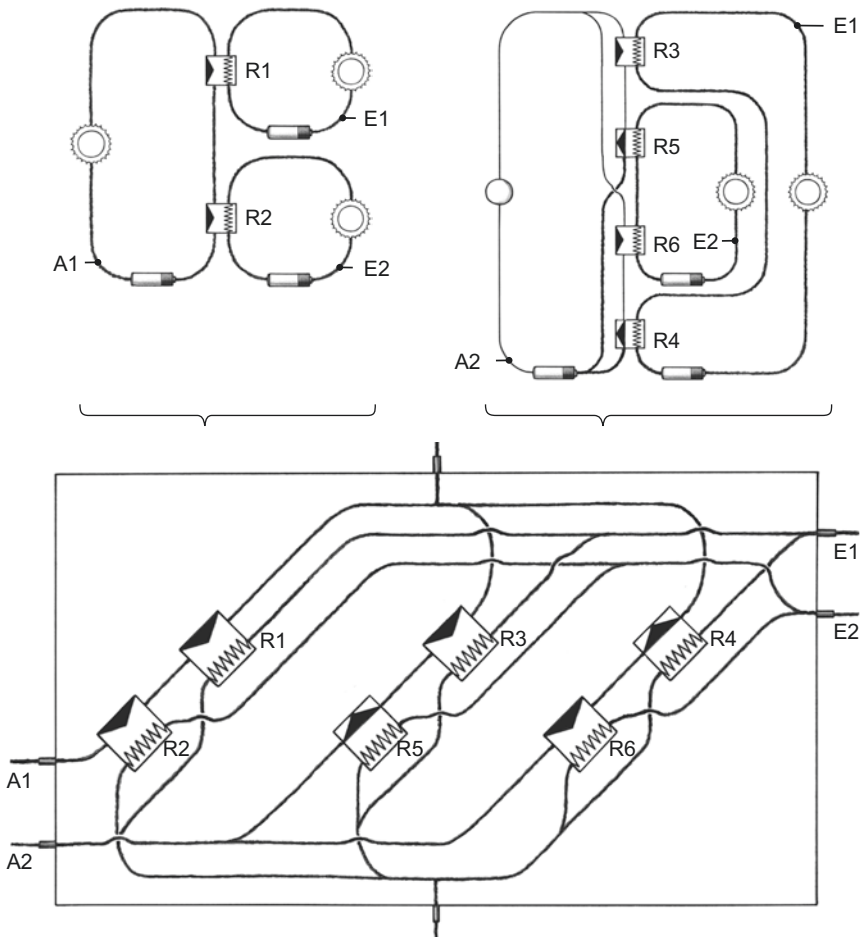


Abb. 2.20 Ein Baustein zur Addition zweier einstelliger Dualzahlen. (© Susanne Staubli, Eckart Zitzler)

Setzen wir diesen Plus-Baustein doch einmal ein und verbauen ihn in unserem Taschenrechner (siehe Abb. 2.21). Sie können die Schalter und Lampen für die ersten zwei Ziffern der Summanden erkennen, die mit den Eingängen des Plus-Bausteins verbunden sind. Die Ausgangsleitungen führen dann direkt zu den Lampen für den ersten Übertrag an der zweiten Stelle und die erste Ziffer des Ergebnisses. Natürlich ist die innere Verschaltung noch nicht komplett, aber der Taschenrechner kann zumindest schon die Ziffern an der ersten, niederwertigsten Stelle addieren, quasi zwei einstellige Dualzahlen zusammenzählen.

Nach demselben Schema können wir eigentlich auch die restlichen Stellen im Taschenrechner verschalten, doch stopp: Wir können dazu nicht einfach den vorhandenen Plus-Baustein verwenden, denn ab der zweiten Stelle kommt eine weitere Eingabe hinzu, nämlich der Übertrag, der von rechts eingespeist wird. Dazu benötigen wir einen erweiterten Plus-Baustein, der drei Ziffern addieren kann. Den werden wir im Folgenden entwickeln, und dabei stelle ich Ihnen gleich ein generelles Verfahren vor, wie beliebige Schaltungen entworfen werden können.

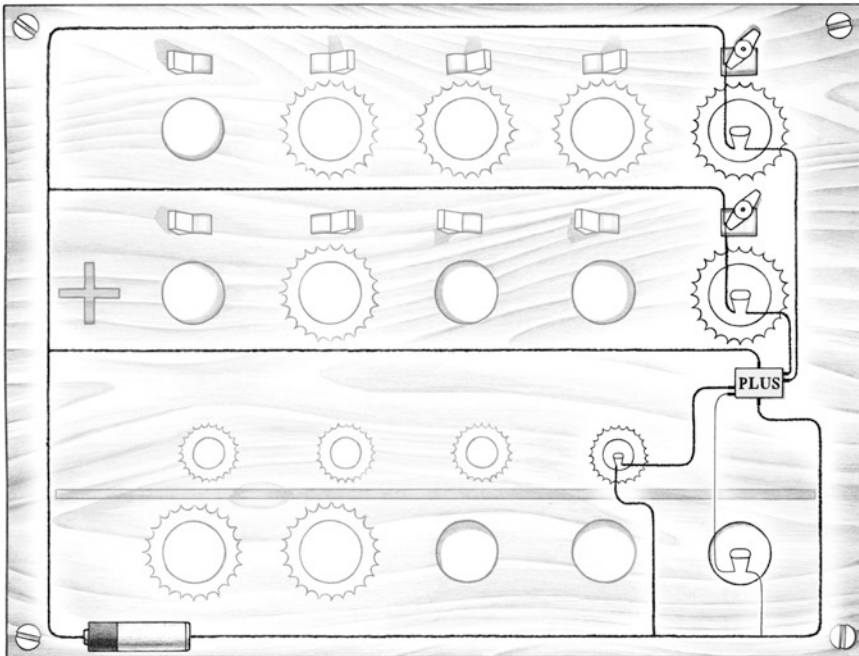


Abb. 2.21 Ein Teil der inneren Verschaltung des Taschenrechners. (© Susanne Staubli, Eckart Zitzler)

Bausteine bauen

Am Anfang steht immer die konkrete Musterzuordnung, die wir mittels einer Schaltung realisieren möchten: Welche Eingabemuster sollen auf welche Ausgabemuster abgebildet werden? Natürlich benutzt man in der Fachwelt nicht Lampen, um eine Musterzuordnung zu beschreiben, sondern Tabellen mit Nullen und Einsen – Wahrheitstabellen genannt. In Abb. 2.22 sehen Sie, wie die Wahrheitstabelle für den erweiterten Plus-Baustein mit drei Eingängen aussieht.

Aus einer Wahrheitstabelle kann nun Schritt für Schritt eine Schaltung abgeleitet werden. Das Fundament dafür bilden drei Bausteine, deren Verschaltungen Sie bereits kennengelernt haben und die in Abb. 2.23 dargestellt sind. Der erste negiert den Eingangswert, der zweite realisiert eine einfache Reihen-Verschaltung mit schließenden Relais und der dritte repräsentiert eine Parallel-Verschaltung mit ebenfalls schließenden Relais. Man nennt diese Grundbausteine auch Gatter, weil sie ähnliche Funktionen wie Lattentore haben, nämlich die Nutzung der Wege lenken. Die drei mit NICHT, UND sowie ODER bezeichneten Grundbausteine genügen, um Schaltungen für beliebige Musterzuordnungen zu entwerfen. Dabei gehen wir in drei Schritten vor.

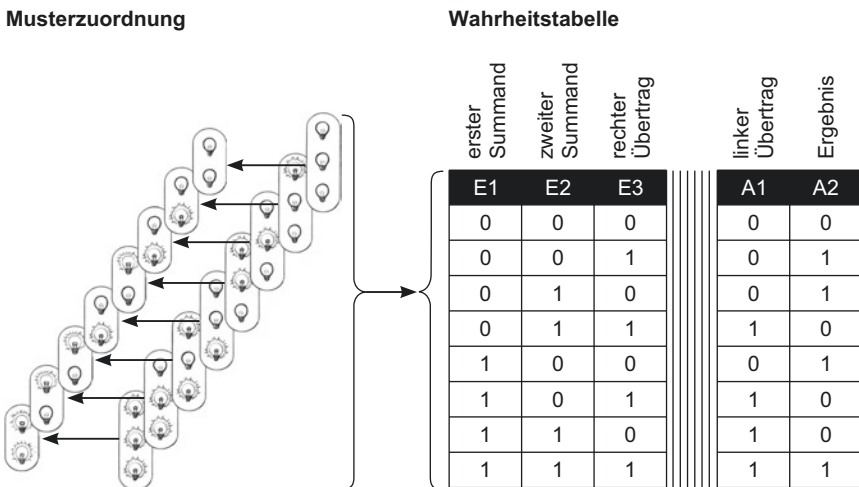


Abb. 2.22 Eine Musterzuordnung wird in Form einer Wahrheitstabelle beschrieben. (© Susanne Staubli, Eckart Zitzler)

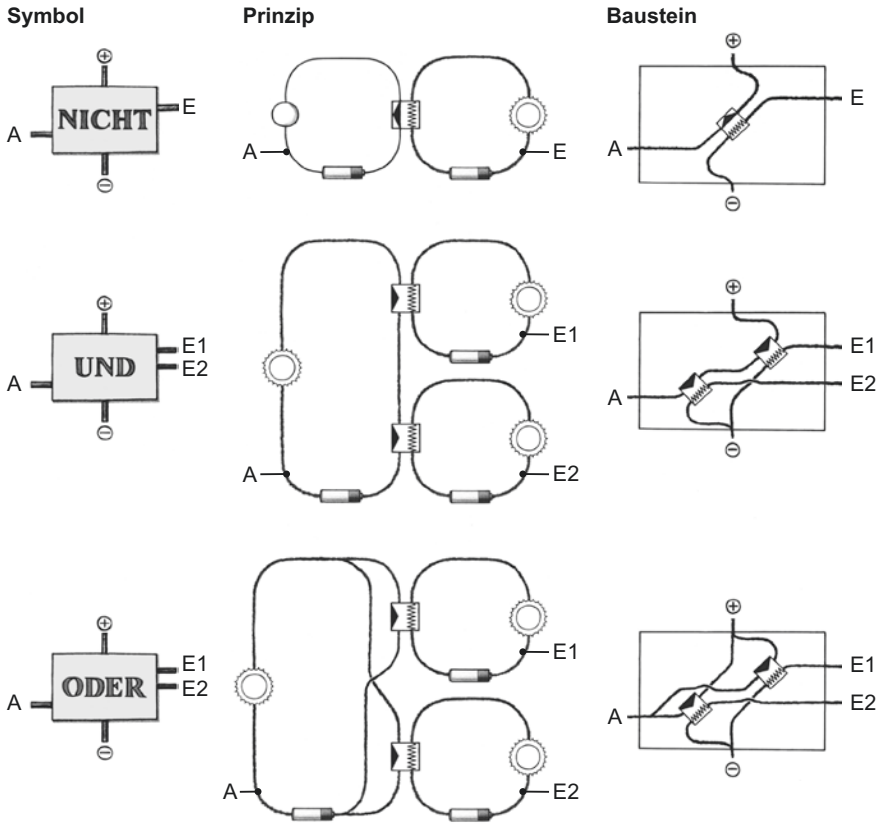


Abb. 2.23 Drei Grundbausteine und ihre Funktionsweise. (© Susanne Staubli, Eckart Zitzler)

- Schritt 1:** Zunächst gestalten wir Teilschaltungen, die einzelne Eingabemuster erkennen können – diejenigen Eingabemuster, bei denen eine Ausgabeleitung auf 1 zu setzen ist. Die Konstruktion für ein spezifisches Eingabemuster besteht aus zwei Stufen und ist in Abb. 2.24 exemplarisch für zwei ausgewählte Eingaben illustriert. In der ersten Stufe werden alle Eingangsleitungen so mittels NICHT-Gattern angepasst, dass alle Leitungen eine 1 tragen, wenn immer das Eingangsmuster anliegt. In der zweiten Stufe werden diese Leitungen mit einem UND-Gatter in Reihe verschaltet. Die Konsequenz: Der Ausgang des UND-Gatters ist genau dann 1, wenn das Eingangsmuster anliegt.

Wahrheitstabelle

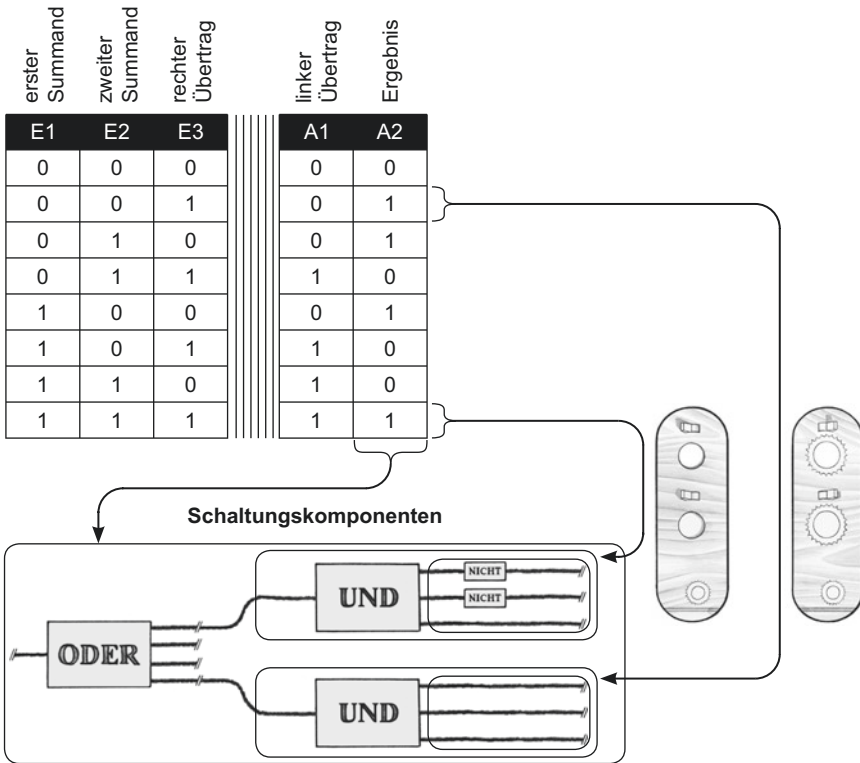


Abb. 2.24 Wie aus einer Wahrheitstabelle eine Schaltung konstruiert wird. (© Susanne Staubli, Eckart Zitzler)

- **Schritt 2:** Anschließend wird für jede Ausgabeleitung eine Verschaltung konstruiert, indem pro Eingabemuster, bei dem der Ausgang 1 ist, die entsprechenden UND-Gatter mittels eines ODER-Gatters parallel verschaltet werden. Der Ausgang des ODER-Gatters ist genau dann 1, wenn eines der Eingabemuster anliegt. Abb. 2.24 demonstriert das Prinzip für die zwei ausgewählten Eingabemuster.
- **Schritt 3:** Am Ende werden die für die einzelnen Ausgangsleitungen resultierenden Schaltungen in einem Baustein zusammengefasst – so, wie wir das bereits im Zusammenhang mit Abb. 2.15 diskutiert hatten.

Bsp. 2.9 erläutert die dreistufige Entwurfsmethodik im Detail und zeigt auf, wie sich so der erweiterte Plus-Baustein konstruieren lässt. Wen die Einzelheiten weniger interessieren, der merke sich einfach: Ausgehend von einer Wahrheitstabelle werden erst Teilschaltungen für diejenigen Zeilen entworfen, bei denen ein Ausgang auf 1 steht; anschließend werden pro Spalte all die Teilschaltungen zusammengeführt, in denen der Ausgang eine 1 aufweist; zum Schluss wird das Ganze zusammengeführt.

Beispiel 2.9: Ein erweiterter Plus-Baustein

In Abb. 2.25 sehen Sie, wie der erweiterte Plus-Baustein mittels der drei Gattertypen NICHT, UND sowie ODER realisiert werden kann. Sie können erkennen, dass es pro Ausgangsleitung eine Verschaltung gibt, deren Ergebnis von einem ODER-Baustein geliefert wird. Die UND-Gatter repräsentieren die Eingabemuster, bei denen die jeweilige Ausgabeleitung auf 1 gesetzt werden muss – wie in der Wahrheitstabelle in Abb. 2.25 zu sehen ist, gibt es genau acht solcher Eingabemuster.

So, nachdem wir jetzt den erweiterten Baustein konstruiert haben, können wir unseren Taschenrechner vollständig zusammensetzen. Wir bauen noch den erweiterten Plus-Baustein an den übrigen Stellen ein und die vollständige Schaltung für den Taschenrechner präsentiert sich dann so wie in Abb. 2.26 dargestellt.

Bravo! Sie haben Ihre erste digitale Schaltung konstruiert und wissen jetzt, wie sich Berechnungen auf Dualzahlen realisieren lassen: nämlich durch das Aufstellen der Wahrheitstabelle und das Ableiten einer dreistufigen Verschaltung mittels NICHT-, UND- sowie ODER-Gattern aus dieser Tabelle. Die Entwurfsmethodik ist generell einsetzbar, hat allerdings ihre Grenzen, wie wir gleich sehen werden.

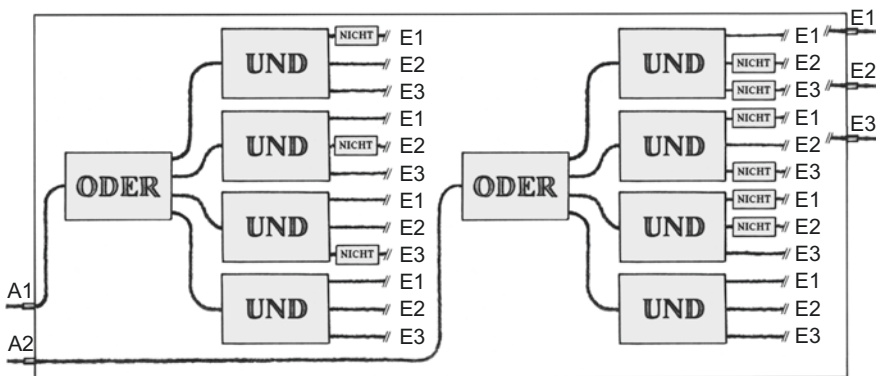


Abb. 2.25 So lässt sich der erweiterte Plus-Baustein unter Verwendung von NICHT-, UND- sowie ODER-Gattern bauen. (© Susanne Staubli, Eckart Zitzler)

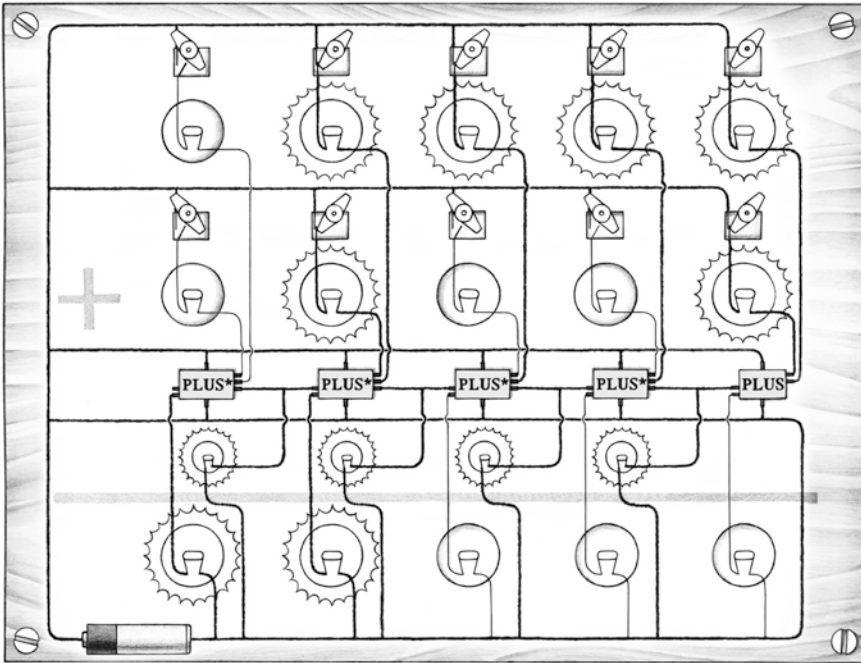


Abb. 2.26 Die vollständige Schaltung für den Taschenrechner mit einem einfachen Plus-Baustein und mehreren erweiterten Plus-Bausteinen, die jeweils mit einem Sternchen markiert sind. (© Susanne Staubli, Eckart Zitzler)

Vom Baustein zur Schaltung

Nehmen wir einmal an, wir wollten unseren Taschenrechner intern mit einem einzigen Baustein implementieren, der dann zehn Eingänge und neun Ausgänge hätte. In diesem Fall umfasste die vollständige Wahrheitstabelle über 1000 Zeilen und entsprechend viele Gatter würden benötigt. Folglich gibt es da ein Problem: Die Wahrheitstabellen und damit auch die Verschaltungen können sehr groß werden. Aus diesem Grund gibt es Methoden, mit denen sich die Anzahl der Gatter minimieren lässt; sie haben ihren Ursprung im Gebiet der Logik.

Die entscheidenden Grundlagen hierfür erarbeitete George Boole (Abb. 2.27). Er war zunächst als Lehrer und später als Mathematikprofessor tätig und gilt als einer der Begründer der modernen mathematischen Logik, insbesondere der Aussagenlogik. Mitte des 19. Jahrhunderts verfasste er eine Schrift, die sich als enorm nützlich für den Bau digitaler Schaltungen erweisen sollte. Ihm ging es jedoch um etwas ganz anderes, nämlich



Abb. 2.27 George Boole, der das Fundament für die nach ihm benannte Boolesche Algebra schuf. (© Camille Büsser)

um die Formalisierung des menschlichen Denkens und des logischen Schlussfolgerns. Ihn beschäftigte die Frage, wie sich Aussagen über die Welt verknüpfen lassen und wie sich der Wahrheitsgehalt zusammengesetzter Aussagen aus dem Wahrheitsgehalt der Einzelaussagen ableiten lässt. Zum Beispiel: Wenn Aussage A gilt und gleichzeitig Aussage B, dann ist auch die Aussage $A \text{ UND } B$ wahr. Die logischen Verknüpfungsoperatoren, die er u. a. verwendete, waren NICHT, UND sowie ODER. Kommt Ihnen das bekannt vor?

Tatsächlich, digitale Schaltungen lassen sich in Form von logischen Aussagen beschreiben (Bsp. 2.10). Der Vorteil: Man kann solche Aussagen umformen, wie wir das aus der klassischen Mathematik kennen. So ist beispielsweise die Aussage „ $(A \text{ UND } B) \text{ ODER } (B \text{ UND } C)$ “ gleichwertig mit der Aussage „ $B \text{ UND } (A \text{ ODER } C)$ “ – das hat Auswirkungen auf die repräsentierten Schaltungen, denn im ersten Fall werden drei Gatter, im zweiten Fall jedoch nur zwei Gatter benötigt.

Beispiel 2.10: Eine Verschaltung als Aussage

Schaltungen lassen sich in Form von logischen Ausdrücken darstellen, indem jede Eingangs- und Ausgangsleitung in eine Aussage überführt und jeder Grundbaustein in eine logische Verknüpfung übersetzt wird. Wenn wir die drei Grundbausteine aus Abb. 2.23 betrachten, so lässt sich deren Verhalten in Form der Aussagen „ $A = \text{NICHT } E$ “, „ $A = E1 \text{ UND } E2$ “ bzw. „ $A = E1 \text{ ODER } E2$ “ beschreiben. Komplexe Schaltungen wie beispielsweise in Abb. 2.24 lassen sich als zusammengesetzte Aussagen repräsentieren: „ $A = ((\text{NICHT } E1) \text{ UND } (\text{NICHT } E2) \text{ UND } E3) \text{ ODER } (E1 \text{ UND } E2 \text{ UND } E3)$ “. Dabei gehen wir von folgender Interpretation aus: Eine Aussage ist wahr genau dann, wenn die entsprechende Leitung auf 1 steht.

Die sogenannte Boolesche Algebra bietet die formale Grundlage, Schaltungen systematisch zu vereinfachen und zu minimieren, sodass möglichst wenige Gatter und Schaltungsstufen benötigt werden. Das geht, weil die Booleschen Operatoren NICHT, UND sowie ODER eben den gleichnamigen Grundbausteinen entsprechen, die wir weiter vorne entwickelt haben. Übrigens beruht das Prinzip, nach denen wir eine Wahrheitstabelle in eine Schaltung übersetzt haben, ebenfalls auf einem Konzept der Booleschen Algebra: der disjunktiven Normalform – dies für den Fall, dass Sie einmal tiefer in die Informatik einsteigen möchten.

Wann immer möglich, versucht man also eine Schaltung nach dem bereits beschriebenen Verfahren zu konstruieren und dann mittels der Booleschen Algebra zu optimieren. Übersteigt die Wahrheitstabelle aber die Grenze des Machbaren, so lassen sich komplexere Schaltungen aus mehreren einzelnen Bausteinen zusammensetzen, so wie wir das bei unserem Taschenrechner durchexerziert haben. Indem wir eine Kaskade von Bausteinen entwickeln, können Berechnungsabläufe durch Schaltungen realisiert werden. Diese Konstruktionsidee, die ja auf den Gattern beruht, versucht man auch in der Biologie einzusetzen.

Gibt es Bausteine in der Zelle?

Im weitesten Sinne finden wir Gatter auch in der Zelle bei der Kopplung von Proteinen mittels der Gene wieder. Eine NICHT-Verknüpfung bei Proteinen heißt ja nichts anderes, als dass ein Protein den Ablesevorgang und somit die Produktion eines anderen Proteins verhindert; letzteres kann nur hergestellt werden, d. h. aktiv sein, wenn ersteres nicht präsent ist. Das erwähnte Schalterprotein in der Kolibakterie realisiert so eine NICHT-Verschaltung. Auch die Logik hinter dem UND-Gatter trifft man an: Zwei Proteine müssen vorhanden sein, damit ein bestimmtes Gen abgelesen und somit das entsprechende Protein produziert wird. Und auch das ODER-Gatter gibt es: Zwei Proteine können alternativ den Ablesevorgang eines bestimmten Gens in Gang setzen.

Wissenschaftler arbeiten nun daran, Protein-Protein-Verschaltungen über neuartige Gene selbst zu konzipieren, um somit Bakterien neue Eigenschaften zu geben. Beispielsweise konnten sie Bakterien dazu zu bringen, sich wie ein fotografischer Film zu verhalten und Bildinformationen festzuhalten – indem sie neue Verschaltungen in die Zellen eingebracht haben (siehe Abb. 2.28). Das Gebiet, welches sich mit solchen Fragestellungen beschäftigt, nennt sich Synthetische Biologie. Dabei versuchen Forschende



Abb. 2.28 Bakterien, die mittels neuer Verschaltungen projizierte Bilder speichern können. (© Jeffrey J. Tabor, Rice University)

ähnlich den Gattern standardisierte „Bausteine“, bestehend aus DNA-Sequenzen und entsprechenden Proteinen, zu bauen, aus denen sich Verschaltungen konstruieren lassen.

Ganz so einfach, wie sich das anhört, ist es natürlich nicht. Wie wir bereits gesehen, ist es in der Zelle nicht so einfach, Leitungen von einem Gatterausgang zu einem Gattereingang zu ziehen. Die Schaltungselemente müssen spezifisch miteinander verknüpft werden, und das macht das Ganze kompliziert. Zellen sind eben nicht dafür entwickelt worden, dass sie für den Menschen handhabbar sind – im Gegenteil zu den digitalen Schaltungen.

Feedback erwünscht!

Die Rechenmaschine, die wir bis hierher entwickelt haben, kann addieren, doch merken kann sie sich nichts. Sobald wir einen Summanden verändern, ändert sich das Ergebnis und die vorherige Summe ist verloren. Um mehrere Zahlen oder sogar Zahlenkolonnen aufzusummieren – denken wir nur an eine Supermarktkasse –, müssten wir uns jedes Zwischenergebnis merken und erneut eingeben. Beheben wir also die Schwachstelle und erweitern den Taschenrechner um die Fähigkeit, Zahlen speichern zu können. Oder anders gesagt: Schauen wir uns an, wie sich die zweite Kerneigenschaft eines Computers, die Speicherfähigkeit, mittels elektrischer Bauteile umsetzen lässt.

Der Taschenrechner wird erweitert

In Abb. 2.29 sehen Sie eine erweiterte Version des Taschenrechners, die das Ergebnis der vorherigen Addition festhält und in der nächsten Addition

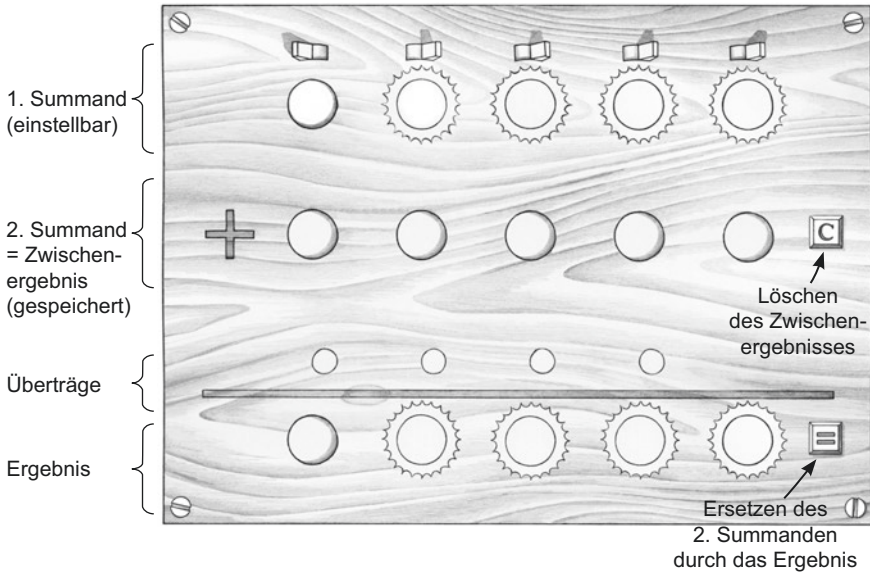


Abb. 2.29 Der erweiterte Taschenrechner, der Zwischenergebnisse festhalten kann. (© Susanne Staubli, Eckart Zitzler)

weiterverwendet. Der erste Summand ist wie gehabt über die Schalter einstellbar. Beim zweiten Summanden sind die Schalter hingegen verschwunden; er lässt sich nicht mehr direkt verändern, sondern repräsentiert die Summe der vorangegangenen Addition, verkörpert also ein Zwischenergebnis. Dafür gibt es nun zwei Taster, mit „=“ und „C“ angesprochen. Wie funktioniert nun dieser modifizierte Taschenrechner?

Zu Beginn, nach dem Einschalten, ist undefiniert, was in der zweiten Lampenreihe angezeigt wird. Mithilfe des „C“-Tasters können Sie den zweiten Summanden auf die Dualzahl 00000 setzen, sodass alle Lampen in der zweiten Reihe ausgeschaltet sind. Stellen Sie nun in der obersten Reihe den ersten Summanden ein, so sehen Sie die eingegebene Zahl auch unten als Ergebnis – klar, der zweite Summand ist ja die Null. Jetzt kommt der „=“-Taster ins Spiel. Wenn Sie ihn drücken, wird der zweite Summand im Zwischenspeicher durch das Ergebnis ersetzt; unmittelbar darauf ändert sich auch wieder das Ergebnis, schließlich hat sich der zweite Summand verändert. Jetzt können Sie die nächste Zahl in der oberen Reihe eingeben und erhalten somit die Summe aus dieser und der vorherigen Zahl. Bsp. 2.11 verdeutlicht die Funktionsweise.

Beispiel 2.11: Wie der erweiterte Taschenrechner funktioniert

Stellen Sie sich vor, Sie schalten den Taschenrechner an, d. h., Sie setzen die Batterie ein, drücken anschließend die „C“-Taste und stellen den ersten Summanden auf die Dualzahl 01111 ein. Dann haben wir die in Abb. 2.29 dargestellte Situation: Die Summe entspricht genau dem ersten Summanden. Drücken Sie jetzt die „=“-Taste, wird in der Zeile des zweiten Summanden die Dualzahl 01111 angezeigt und nahezu gleichzeitig unten die neue Summe 11110. Drücken Sie erneut die „=“-Taste, so wird nun 01111 zu 11110 hinzugezählt; die Summe ist eigentlich 101101, doch weil der Taschenrechner auf fünf Stellen beschränkt ist, wird unten nur die Dualzahl 01101 angezeigt.

Gut, jetzt zur Umsetzung. Um unseren erweiterten Taschenrechner zu bauen, benötigen wir offenbar einen Baustein, mit dem sich eine Ziffer speichern lässt – und zwar unabhängig davon, was sich an den Eingabeleitungen tut. Es braucht eine Schaltung, die ihren Zustand konstant hält und diesen nur ausdrücklich, auf Befehl sozusagen, ändert. Nur wie lässt sich eine solche konstruieren?

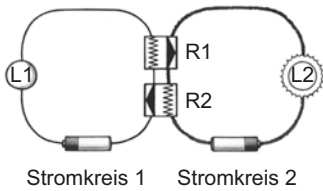
Was Feedback bewirken kann

Der Kniff ist: Wir koppeln Stromkreise so, dass sie sich gegenseitig beeinflussen, wir bauen also eine Schleife in die Wirkungskette ein. Bislang war es ja so, dass ein Stromkreis einen anderen beeinflusste, letzterer aber keinen Einfluss auf den vorigen hatte. Verwenden wir hingegen die Schleifenkonstruktion, dann verläuft die Steuerung nicht immer in die gleiche Richtung (von rechts nach links in unseren Abbildungen), sondern auch zurück, es ergibt sich also ein Wirkungskreis. Das nennt man Rückkopplung.

Die einfachste Variante der Rückkopplung ist in Abb. 2.30 links gezeigt: zwei Stromkreise – der Übersichtlichkeit halber sind sie ohne Schalter dargestellt –, die sich gegenseitig ausschalten. Die Konstruktion mit den öffnenden Relais bewirkt, dass immer nur einer der beiden Stromkreise aktiviert sein kann. Logisch, denn ein deaktivierter Stromkreis aktiviert den anderen Stromkreis und umgekehrt.

Die Frage ist nun, wie wir den Zustand dieses Systems der zwei verschlungenen Stromkreise verändern und gezielt einstellen können. Wie können wir bestimmen, welche der zwei Lampen leuchtet? Offensichtlich kann das nur von außen geschehen, und zwar indem wir zu jedem Stromkreis einen weiteren hinzufügen, der diesen – unabhängig von dessen Gegenspieler – aktivieren kann. Diese Erweiterung sehen Sie in Abb. 2.30 rechts. Die Stromkreise

Prinzip



erweitertes Prinzip

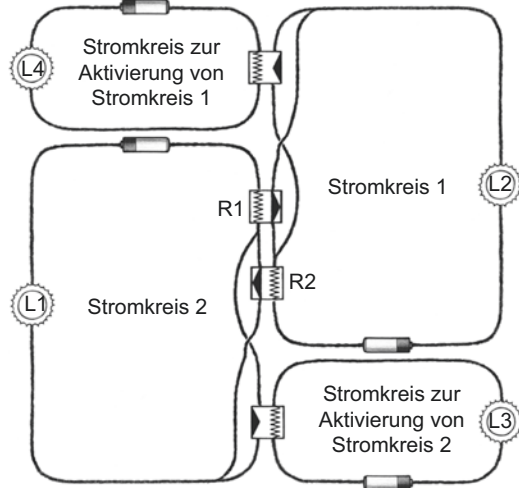


Abb. 2.30 Zwei rückgekoppelte Stromkreise. (© Susanne Staubli, Eckart Zitzler)

1 und 2 sind die vorherigen beiden rückgekoppelten Stromkreise, die die Aufgabe haben, einen der zwei möglichen Zustände zu speichern. Über die zwei separaten Stromkreise lässt sich sowohl Stromkreis 1 als auch Stromkreis 2 unabhängig voneinander aktivieren. Die zusätzlichen zwei Stromkreise dienen also der Änderung des gespeicherten Zustands.

Wir können das Ganze auch in Bausteinweise darstellen, wie es in Abb. 2.31 illustriert ist. Die Rolle der rückgekoppelten Stromkreise übernehmen hier zwei ODER-Gatter, ihre Ausgangswerte sind jeweils komplementär zueinander und repräsentieren den gespeicherten Zustand. Sollte Ihnen diese Schaltung suspekt vorkommen, so seien Sie beruhigt. Man muss sie schon etwas studieren, um sicher zu sein, dass sie identisch mit der vorherigen ist und das Richtige tut. Bsp. 2.12 hilft dabei, das Schaltverhalten zu verstehen.

Beispiel 2.12: Zwei rückgekoppelte Stromkreise im Zeitraffer

Betrachten Sie Abb. 2.32. Dort sehen Sie die Schaltung aus Abb. 2.31 ergänzt um zwei manuelle Schalter, mit denen wir den Systemzustand von außen einstellen können. Des Weiteren sehen Sie eine Abfolge von Aktionen, d. h. Schaltereinstellungen, die wir vornehmen, sowie das entsprechende Verhalten der Schaltung – immer in Abhängigkeit der vorherigen Schritte.

Zu Beginn schließen wir beide Schalter, d. h., beide ODER-Gatter geben eine 1 aus und sowohl Lampe L1 als auch Lampe L2 leuchtet. Jetzt öffnen wir im zweiten Schritt den unteren Schalter, woraufhin das untere ODER-Gatter eine 0 ausgibt – beide seiner Eingänge sind 0. Im dritten Schritt öffnen wir auch noch den oberen Schalter, die zwei rückgekoppelten ODER-Gatter sind nun nicht mehr von außen beeinflusst. Das System behält daher seinen vorherigen Zustand, d. h., Lampe L1 ist ausgeschaltet, während Lampe L2 brennt. Dieser Zustand, so kann man sagen, ist nun gespeichert. Im vierten Schritt ändern wir den Zustand, indem wir aktiv den Ausgang des unteren ODER-Gatters über das Schließen des unteren Schalters auf 1 setzen. Jetzt kippt der Zustand: Da das untere ODER-Gatter eine 1 ausgibt, sind beide Eingänge des oberen ODER-Gatters auf 0 gesetzt und der Ausgang ist ebenfalls 0. Daher leuchtet jetzt Lampe L1, während Lampe L2 erlischt. Dieser Zustand wird wiederum beibehalten, solange beide Schalter geöffnet sind, wie es im fünften Schritt der Fall ist.

Jetzt haben wir also ein rückgekoppeltes Stromkreissystem, das gesamthaft zwei Zustände annehmen und festhalten kann: Entweder Lampe L1 brennt und Lampe L2 nicht oder umgekehrt. Prinzipiell lässt sich auf diese Weise eine Dualziffer speichern, aber so ganz sind wir noch nicht dort, wo wir hinwollen, oder? Wir wollen ja eigentlich ein digitales Speicherelement. Da sollte es einen Ausgang haben, über den der gespeicherte Wert ausgegeben wird, zudem einen Eingang, über den ein neuer, zu speichernder Wert eingespeist werden kann, sowie eine Steuerleitung, über die angegeben werden kann, wann der Speicher neu gesetzt werden soll. Genau so einen Baustein werden wir jetzt konzipieren.

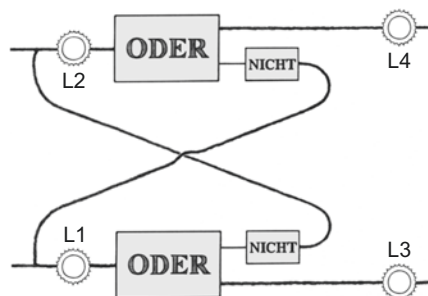


Abb. 2.31 Rückkopplung mittels Gattern (Batterie und Batterieverbindungen sind nicht eingezeichnet). (© Susanne Staubli, Eckart Zitzler)

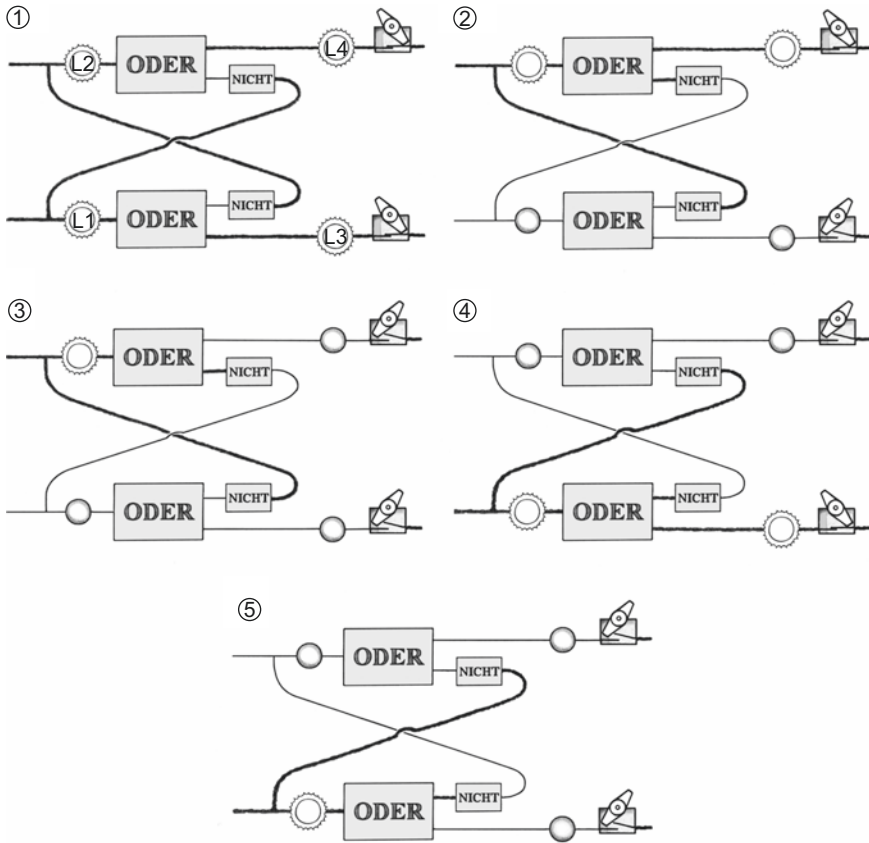


Abb. 2.32 Das Verhalten der rückgekoppelten Schaltung über die Zeit. (© Susanne Staubli)

Alles andere als ein Flop: das Flipflop

Der elementare Speicherbaustein, um den es hier geht, trägt den lautmale-
rischen Namen Flipflop und ist in Abb. 2.33 abgebildet. Ein Flipflop hat
direkt nichts mit den bekannten Sandaletten zu tun; einzig die Laute beim
Laufen, das Flip und das Flop, erinnern an die zwei Zustände des Flipflop-
Bausteins.

Der Flipflop-Baustein hat einen Ausgang, der den gespeicherten Wert, 0
oder 1, verfügbar macht. Über die unteren zwei Eingänge, die mit „=“ und
„C“ angeschriebenen Steuerleitungen, kann der Speicherinhalt überschrie-
ben werden. Ist die „C“-Leitung auf 1 gestellt, so wird der Wert 0 gespei-
chert und ausgegeben. Ist die „=“-Leitung auf 1 gesetzt, so wird der Wert



Abb. 2.33 Flipflop – das elementare digitale Speicherelement. (© Susanne Staubli, Eckart Zitzler)

am Eingang „E“ in den Speicher geschrieben und ausgegeben. Bsp. 2.13 zeigt, wie dieser Baustein auf der Basis unserer rückgekoppelten Schaltung realisiert werden kann.

Beispiel 2.13: Eine einfache Flipflop-Variante

Um die Schaltung aus Abb. 2.31 in einen einfachen Flipflop-Baustein überführen zu können, benötigen wir drei zusätzliche Gatter, wie in Abb. 2.34 zu sehen ist. Als Baustein-Ausgang verwenden wir den Ausgang des oberen ODER-Gatters. Die „C“-Leitung ist mit einem Eingang des unteren ODER-Gatters verbunden; führt die „C“-Leitung eine 1, so gibt das untere ODER-Gatter eine 1 aus, womit wiederum der Ausgang des oberen ODER-Gatters auf 0 gesetzt wird. Die Eingangsleitung „E“ dient dazu, den Ausgang eines der beiden ODER-Gatter explizit auf 1 zu setzen. Daher wird das Eingangssignal direkt – einmal abgesehen von den UND-Gattern – zum oberen ODER-Gatter bzw. negiert zum unteren ODER-Gatter weitergeleitet; eine 1 am Eingang „aktiviert“ also das obere ODER-Gatter, während eine 0 das untere ODER-Gatter „aktiviert“. Die UND-Gatter schließlich haben den Zweck, dass die Aktivierung der ODER-Gatter über den Eingang nur erfolgt, wenn auch die „=“-Leitung eine 1 führt.

Auf der rechten Seite in Abb. 2.34 sehen Sie noch das Zeitverhalten des einfachen Flipflop-Bausteins. Wenn immer die „=“-Leitung auf 1 gesetzt ist, wird der Wert am Eingang unmittelbar übernommen und auch auf die Ausgabeleitung gelegt. Erst wenn die „=“-Leitung wieder eine 0 führt, wird der zuletzt gespeicherte Wert behalten, unabhängig davon, welche Werte am Eingang anliegen.

In der Praxis kommt jedoch zumeist eine andere Implementierung zum Zuge als die, die ich Ihnen in Bsp. 2.13 vorgestellt habe. Ein Flipflop-Baustein ist einfacher zu nutzen, wenn es einen klar definierten Zeitpunkt gibt, zu dem der Wert am Eingang übernommen wird. Bei der einfachen Flipflop-Variante ergibt sich das Problem, dass sich der gespeicherte Wert im Laufe einer Zeitspanne ändern kann, nämlich solange die „=“-Steuerleitung auf 1 steht. Die Lösung besteht nun darin, zwei einfache Flipflop-Bausteine hintereinanderschalten und dabei das „=“-Signal bei beiden Bausteinen

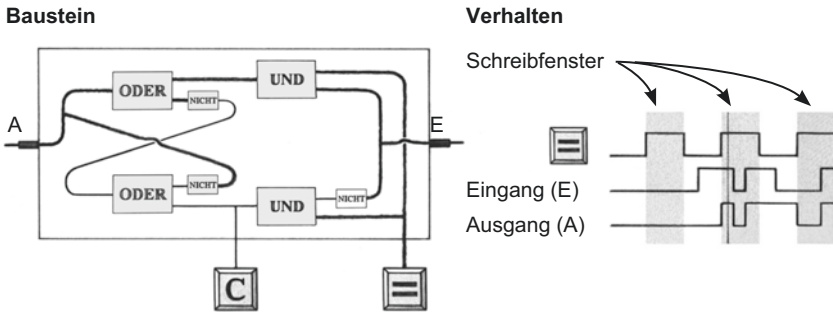


Abb. 2.34 Das Innenleben eines einfachen Flipflop-Bausteins. (© Susanne Staubli, Eckart Zitzler)

in jeweils entgegengesetzter Richtung anzulegen. Diese Konstruktion nennt sich im Fachjargon D-Flipflop und führt dazu, dass der Eingangswert immer dann übernommen wird, wenn das „=“-Steuersignal vom Zustand 0 auf den Zustand 1 wechselt. Bsp. 2.14 erläutert, wie das genau funktioniert.

Beispiel 2.14: Das D-Flipflop im Detail

Der vollständige D-Flipflop-Baustein ist in Abb. 2.35 abgebildet. Der linke einfache Flipflop-Baustein dient der eigentlichen Speicherung einer Dualziffer, der zweite, vorgeschaltete einfache Flipflop-Baustein dient dem „Einfrieren“ des Eingangswerts. Bei Letzterem ist daher das „=“-Signal umgekehrt angelegt, es wird also vor den UND-Gattern negiert. Wie verhält sich nun dieser zusammengesetzte D-Flipflop-Baustein?

Das linke Flipflop ist, wie gehabt, durchlässig, wenn das „=“-Signal auf 1 steht; jede Änderung am Eingang des Flipflops bewirkt auch eine Änderung an seinem Ausgang. Das Flipflop auf der rechten Seite hingegen ist nur durchlässig, wenn das „=“-Signal auf 0 gesetzt ist. Wann immer die „=“-Leitung von 0 auf 1 wechselt, wird das rechte Flipflop gesperrt und das linke Flipflop öffnet sich. Das ist der einzige Moment, in dem der Wert am Eingang „E“ zum linken Flipflop gelangt und dort gespeichert wird. Danach behält das rechte Flipflop seinen Wert, unabhängig von Änderungen an seinem Eingang. Es öffnet sich erst wieder, wenn die „=“-Leitung auf 0 gesetzt wird; gleichzeitig wird dann allerdings das linke Flipflop gesperrt. Erst beim nächsten Wechsel der „=“-Leitung von 0 auf 1 kann der Speicherinhalt des linken Flipflops erneut überschrieben werden.

Das Verhalten wird klarer, wenn Sie sich den zeitlichen Verlauf in Abb. 2.35 unten rechts anschauen. Der Eingangswert wird immer zu dem Zeitpunkt übernommen, wenn auf der „=“-Leitung der Wechsel von 0 auf 1 erfolgt.

Jetzt haben wir also unser digitales Speicherelement, den Flipflop-Baustein, der genau eine binäre Ziffer speichern kann: 0 oder 1. Er besitzt einen

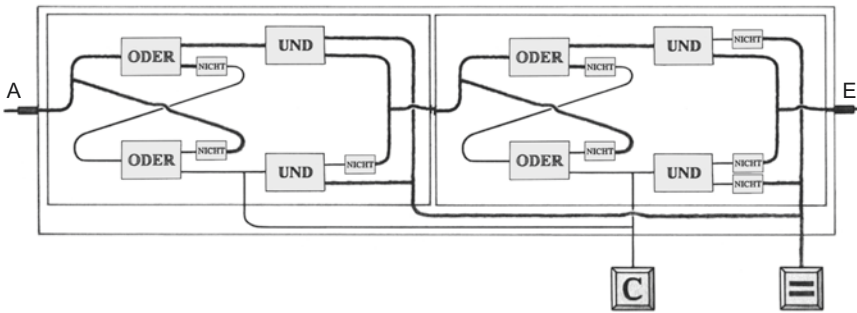
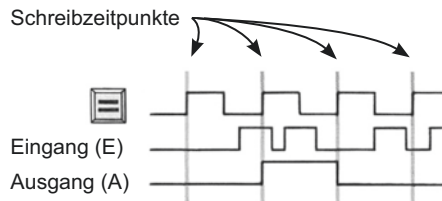
Baustein**Verhalten**

Abb. 2.35 Struktur und Verhalten eines D-Flipflop-Bausteins. (© Susanne Staubli, Eckart Zitzler)

Eingang, einen Ausgang und zwei Steuereingänge, die kontrollieren, wann und mit welchem Wert der Inhalt überschrieben wird. Mit diesem Baustein können wir die Eigenschaft der Speicherfähigkeit implementieren. Die anderen Bausteine, die wir uns vorher angeschaut hatten, sind für die Automatik zuständig. Indem wir beide Arten von Bausteinen kombinieren, lassen sich auch mehrschrittige Rechenvorgänge realisieren – mehr dazu in Kap. 3.

So, und jetzt gebe ich Ihnen endlich die Auflösung, wie wir den erweiterten Taschenrechner intern ausgestalten können. In Abb. 2.36 ist exemplarisch der Einbau des Flipflops für die erste Stelle dargestellt; für die restlichen Stellen werden weitere Flipflops analog verbaut. An diesem Beispiel lässt sich auch noch einmal der Unterschied zwischen den beiden Flipflop-Implementierungen veranschaulichen. Verwenden wir das einfache Flipflop, dann können wir die „=“-Taste gedrückt halten, den ersten Summanden laufend verändern und der Wert des zweiten Summanden ändert sich ebenfalls laufend. Das D-Flipflop hingegen bewirkt, dass die Änderung nur beim Herunterdrücken der „=“-Taste erfolgt.

Übrigens – wenn wundert’s – gibt es das Flipflop auch in der Biologie.

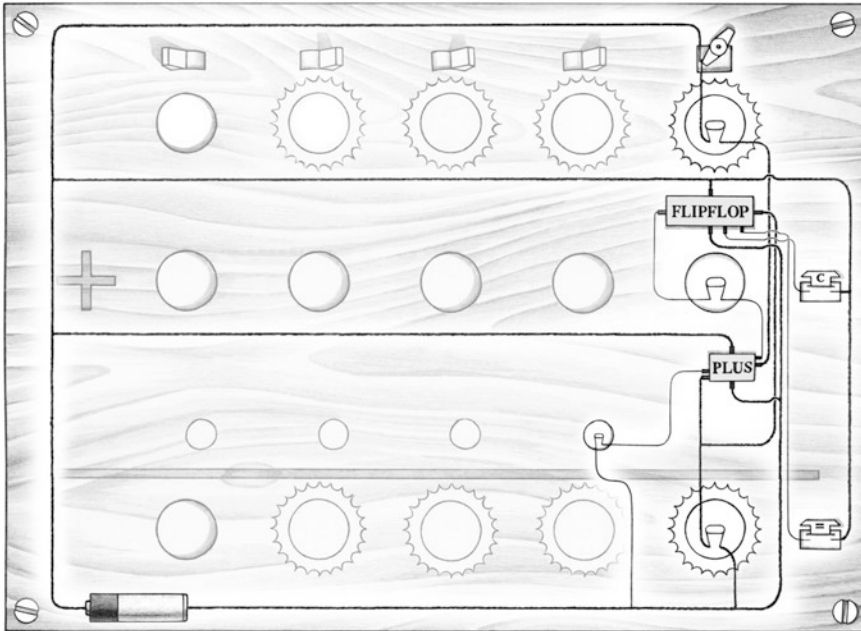


Abb. 2.36 Der erweiterte Taschenrechner von innen. (© Susanne Staubli, Eckart Zitzler)

Wenn Viren Zustände haben

Kolibakterien können uns krank machen, doch auch Kolibakterien selbst können erkranken. Sie können beispielsweise von einem Virus, dem Lambda-Phagen, befallen werden (Abb. 2.37). Der Lambda-Phage besteht aus einem Kopf, in dem sich ein doppelsträngiges DNA-Molekül befindet, und einem Schwanzteil. Als solches ist das Virus kein Lebewesen, es kann sich nicht selbstständig reproduzieren, sondern braucht dafür einen Wirt, eben die Kolibakterie.

Wenn der Lambda-Phage eine Kolibakterie befällt, dann wird die in seinem Kopf enthaltene DNA in die DNA der Kolibakterie eingebaut und wie ein Teil der Kolibakterie behandelt. Das heißt, der Virus bedient sich der Maschinerie der Kolibakterie, um seine eigenen Proteine herzustellen und über sie in das Leben der Kolibakterie einzugreifen. Recht ausgebufft, nicht wahr?

Nun gibt es zwei Situationen. Anfangs verhält sich der Virus still, d. h., seine Gene werden nicht abgelesen und der Virus vermehrt sich als Teil

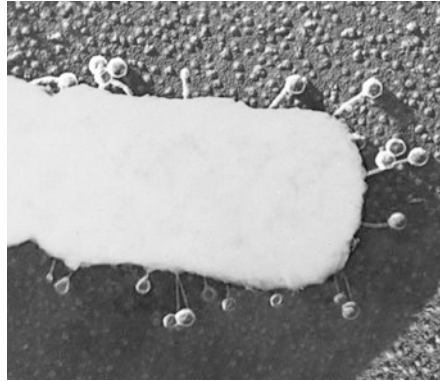


Abb. 2.37 Viren bei der Arbeit: Lambda-Phagen befallen eine Kolibakterie. (© Science Source/mauritus images)

der Kolibakterien-DNA zusammen mit der Kolibakterie. Irgendwann, wenn ein äußeres Signal wie z. B. ein Stressfaktor wie UV-Strahlung oder Nahrungsmangel in Erscheinung tritt, wird der Virus aktiv: Auf einmal werden seine Gene aktiviert. Das Resultat ist, dass neue Viren hergestellt werden und danach die Wirtszelle zerstört wird, damit diese austreten können. Der Virus hat quasi zwei Zustände: still sein oder sich selbst reproduzieren und freisetzen. Diese zwei Zustände werden über zwei rückgekoppelte Proteine gesteuert, die – jetzt kommt's – in Form eines Flipflops verbunden sind (siehe Bsp. 2.15).

Beispiel 2.15: Ein biologisches Flipflop

In Abb. 2.38 sehen Sie die zwei rückgekoppelten Proteine, die den Zustand des Virus bestimmen, zusammen mit ihren codierenden Genen. Das eine Protein wird mit der Abkürzung *cl* bezeichnet. Wird es hergestellt, so schläft der Virus und ist ein stiller Gast in der Wirtszelle; der Grund ist, dass *cl* als Schalterprotein fungiert und alle anderen Virengene abschaltet. Ein anderes Protein, mit *Cro* bezeichnet, bewirkt das Gegenteil, es schaltet die für die Reproduktion des Virus notwendigen Gene an.

Cro ist nun mit *cl* über eine Rückkopplung verbunden, wie in Abb. 2.38 erkennbar ist. Wird das *cl*-Protein produziert, so lagert es sich auch an die DNA vor dem *Cro*-Gen an, was wiederum verhindert, dass die RNA-Polymerase das *Cro*-Gen ablesen kann. Wird das *Cro*-Protein produziert, so lagert sich dieses an den Promotor des *cl*-Gens an und verhindert so das Ablesen des *cl*-Gens. Nur von außen kann der Zustand geändert werden. Damit hätten wir unser biologisches Flipflop.

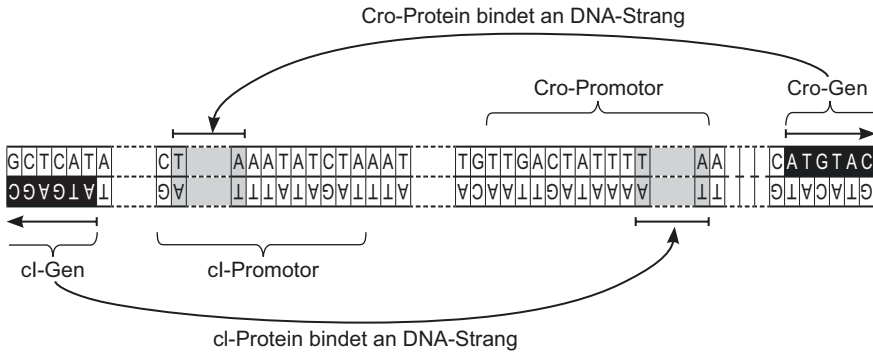


Abb. 2.38 Ein Flipflop aus DNA und Proteinen: Zwei Proteine auf der Virus-DNA schalten sich gegenseitig aus. (© Susanne Staubli, Eckart Zitzler)

Die Zelle verfügt natürlich noch über ganz andere Speichermöglichkeiten als durch die Rückkopplung zweier Proteine. Beispielsweise gibt es auch Markierungen an der DNA selbst, die bewirken, dass bestimmte Gene gar nicht erst aktiviert werden. Das ist das Grundkonzept bei komplexeren Lebewesen, bei denen sich Zelltypen herausbilden und entsprechend nur bestimmte Gengruppen abgelesen werden.

Das Feuerwerk im Kopf

Am Ende dieses Kapitels kommen wir noch auf das Nervensystem zu sprechen – dieses Wunderwerk der Natur darf nicht fehlen, wenn von Mustern und deren Verarbeitung die Rede ist. Primär hat sich das Nervensystem bei Tieren gebildet, um das Zusammenspiel der verschiedenen Zellen im Körper zu koordinieren und aus Sinneswahrnehmungen angemessene Reaktionen zu generieren. Durch die Herausbildung eines Verarbeitungszentrums, dem Gehirn, wurden enorme geistige Anpassungsfähigkeiten geschaffen, die die Eroberung neuer Lebensräume ermöglichten. Vorausschauendes Handeln, Lernen und Selbstbewusstsein gehören zu den erstaunlichsten Errungenschaften dieses Denkkorgans.

Was wir uns jetzt noch anschauen wollen, sind die Grundprinzipien, nach denen das Gehirn Informationen verarbeitet. Im Zentrum steht dabei die Nervenzelle, und auch hier gibt es spannende Parallelen zu den digitalen Schaltungen.

Die Nervenzelle, das Super-Relais

Was das Relais bzw. der Transistor für den Computer, ist die Nervenzelle, die auch als Neuron bezeichnet wird, für das Nervensystem. Neuronen stellen die elementaren Verarbeitungseinheiten des Gehirns dar und sind letztlich nichts anderes als Schalter mit Leitungen, wenn auch äußerst ausgeklügelte Schalter. Ganze Bücher widmen sich einzig der Nervenzelle. So ein faszinierendes und unglaublich komplexes System präsentiert sich uns hier, dass wir im Folgenden nur das Wichtigste in Kürze betrachten können.

Zoomen wir also einmal in unser Gehirn und schauen uns eine Nervenzelle genauer an (Abb. 2.39). Ein Neuron besteht im Wesentlichen aus einem Zellkörper, in dem die DNA beherbergt ist, und mehreren verästelten Fortsätzen – wir könnten auch sagen: Leitungen –, die von dem Zellkörper ausgehen. Über diese Fortsätze verbindet sich eine Nervenzelle mit anderen Nervenzellen oder anderweitigen Zellen wie beispielsweise den Muskelzellen. Das Vernetztsein ist zentral, denn das Neuron ist darauf spezialisiert, Informationen aufzunehmen, zu verarbeiten und weiterzuleiten; mit bis zu 100.000 anderen Zellen kann eine einzelne Nervenzelle in Kontakt stehen. Die meisten der Fortsätze sind Eingabeleitungen und dienen dazu, Informationen von anderen Nervenzellen aufzunehmen. Man nennt sie Dendriten, häufig verästeln sie sich weiter und bilden einen Baum von Fortsätzen. Nur ein Fortsatz ist dafür vorgesehen, Informationen an andere Nervenzellen weiterzuleiten; diese Ausgabeleitung heißt Axon. Das Axon kann sehr kurz, aber auch sehr lang sein (bis zu einem Meter, das entspricht dem 30.000-Fachen des Zellkörperdurchmessers).

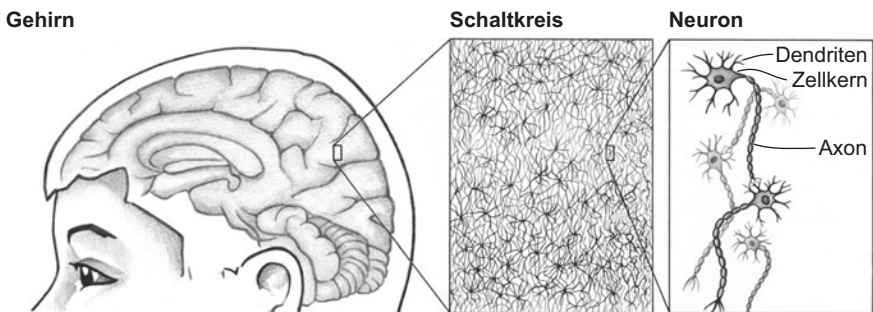


Abb. 2.39 Nervenzellen – Neuronen genannt – stellen die elementaren informationsverarbeitenden Elemente des Gehirns dar. (© Susanne Staubli, Eckart Zitzler)

Im Nervensystem werden Zeichen über elektrische Signale repräsentiert – wie bei den digitalen Schaltungen. Ein Signal ist hierbei eine elektrische Spannung zwischen Zellinnerem und Zellumgebung: Entweder ist das Innere negativ oder positiv geladen. In der Regel befindet sich ein Neuron im Zustand des Negativ-Geladenseins. Wird die Ladung jedoch positiv, so entlädt sich diese in Form eines Nervenimpulses, der sich blitzschnell entlang des Axons ausbreitet. Ein Nervenimpuls ist ein digitales Signal, d. h., entweder er ist vorhanden oder nicht – genauso wie in einem Stromkreis Strom fließen kann oder nicht. Zudem ist ein Nervenimpuls nur ein lokaler Zustand von kurzer zeitlicher Dauer. Wenn wir uns die elektrischen Signale als Töne vorstellen, so gibt es bei den Stromkreisen zwei Dauertöne; bei der Nervenzelle hingegen gibt es nur einen Dauerton, der unterbrochen wird durch einen Knall, den Nervenimpuls.

Wie funktioniert nun die Informationsverarbeitung? Ein Neuron nimmt über seine Dendriten, die Eingangsleitungen, Signale von anderen Neuronen entgegen und wird dadurch erregt, sprich: Die innere Ladung verändert sich. Steigt das Erregungsniveau über eine gewisse Schwelle, entsteht auf einmal ein starker Spannungsaufbau zwischen Zellkörper und Umgebung, der schließlich in den besagten Nervenimpuls mündet – man sagt auch: Das Neuron feuert. Der Nervenimpuls wird über das Axon an die anderen Neuronen weitergeleitet. Die Signalübertragung zwischen Axon und Dendriten erfolgt dabei auf chemischen Weg, wie Bsp. 2.16 aufzeigt.

Beispiel 2.16: Synapsen: Wie Neuronen Informationen austauschen

Genau genommen bildet sich dort, wo ein Axon mit einem Dendrit in Verbindung tritt, eine Kontaktstelle, die Synapse genannt wird. Synapsen werden extra bezeichnet, weil Axon und Dendrit nicht, wie man sich das vorstellen könnte, zusammenwachsen, sondern separat bleiben und durch einen Spalt, der extrem klein ist, getrennt sind.

Was passiert nun an den Synapsen, wenn der elektrische Impuls das Axon an dieser Stelle passiert? Es werden chemische Vorgänge in Gang gesetzt, bei denen spezielle Moleküle, die Neurotransmitter, freigesetzt werden, die über den Spalt zum Dendrit gelangen. Das Überqueren des Spaltes geht relativ schnell, weil der Spalt so extrem schmal ist. Am Ziel angekommen, bewirken die Neurotransmitter wiederum eine Spannungsänderung; die Informationsübertragung im Nervensystem arbeitet also elektrochemisch.

Vor diesem Hintergrund kann man durchaus sagen, ein Neuron ist eine Art Schalter. Ebenso wie beim Relais kann ein Neuron auf eine andere Nervenzelle einen aktivierenden Einfluss haben (die Spannung wird an der

entsprechenden Synapse erhöht) oder einen hemmenden (die Spannung wird an der entsprechenden Synapse vermindert). Allerdings gibt es nicht nur zwei Formen der Einflussnahme – „schließend“ oder „öffnend“ –, sondern ein Kontinuum an Möglichkeiten. Der Grund ist, dass Synapsen verschieden stark ausgeprägt sein können. Dieser feine Unterschied – neben viele weiteren – macht das Gehirn ungleich komplexer als einen Computer.

Neuronale Verschaltungen

Faszinierend ist, dass ein einfaches Element wie das Relais bzw. ein verhältnismäßig einfaches Element wie das Neuron ausreichend ist, um ein enorm komplexes Verarbeitungsverhalten zu erzeugen. Schalten und walten: Der Schalter ist der Schlüssel zum Verständnis von komplexen Verschaltungen. Diese Verschaltungen sind im Gehirn äußerst effizient, kein Schaltungsvorgang erfolgt da zu viel. Denn an sich arbeitet unser Gehirn recht langsam mit maximal 500 Schaltungsvorgängen pro Sekunde. Gleichzeitig ist es enorm schnell, wenn wir uns nur überlegen, wie wir in Sekundenbruchteilen ein Gesicht erkennen können. Das lässt sich nur dadurch erreichen, dass die Neuronen einen extrem hohen Verschaltungsgrad aufweisen und die Schaltungen sehr kompakt sind, also möglichst wenige Schaltvorgänge beinhalten. Dabei geht es nicht einzig um Geschwindigkeit, sondern auch darum, Platz und Energie zu sparen. Die Schaltungsvorgänge in Neuronen benötigen nur einen Bruchteil der Energie einer vergleichbaren digitalen Schaltung.

Wie bei den digitalen Schaltungen gibt es im Nervensystem verschiedene Möglichkeiten der Kopplung. Da hat es NICHT-, UND- sowie ODER-Verschaltungen und zudem noch unendlich viele Varianten dazwischen: z. B. wenn ein Nervenimpuls genau dann ausgelöst wird, wenn mindestens drei eingehende Neuronen feuern. Denn letztlich addiert eine Nervenzelle die erhaltenen Eingangssignale auf, gewichtet über die Synapsen: Einige Synapsen haben ein starkes Gewicht, also einen starken Einfluss darauf, ob das Zielneuron feuert oder nicht, andere Synapsen transportieren nur einen schwachen Impuls. Insofern kann ein Neuron zusammen mit den Synapsen an seinen Dendriten als eine Art Gatter betrachtet werden, wobei jede Spielart der Verschaltung der Eingänge realisiert werden kann. Das Neuron ist also ein enorm mächtiges Instrument der natürlichen Informationsverarbeitung.

Ob es auch hier Plus-Bausteine oder anders geartete Bausteine gibt? Klar ist: Das Gehirn weist eindeutig Strukturen auf und verschiedene Areale

verrichten spezifische Aufgaben, z. B. für die Bewertung von Alternativen oder die Verknüpfung von Wahrnehmungen (Abb. 2.40). Das betrifft allerdings mehr die Gesamtarchitektur. Doch auch auf der unteren Ebene der Neuronen existieren etablierte Strukturen. In der Großhirnrinde – das ist die evolutionär jüngste Region unseres Gehirns, die seine zerfurchte, gefaltete äußere Gestalt ausmacht – sind Neuronen in Schichten organisiert, darüber hinaus bilden vertikale Neuronengruppen sogenannte Säulen aus, die Verarbeitungseinheiten, also Bausteine gleichkommen (siehe Bsp. 2.17). Unser Gehirn arbeitet demnach mehrstufig, und das erinnert an das Konstruktionsprinzip für digitale Schaltungen in Abb. 2.24.

Beispiel 2.17: Bausteine in der Großhirnrinde

Die Großhirnrinde ist randvoll mit Nervenzellen – da gibt es verschiedene Formen mit eigenen Namen –, die hochgradig miteinander vernetzt sind. Die Nervenzellen liegen außen, eben auf der Rinde, während die Nervenfasern im Innern des Großhirns liegen. Dabei ist die Rinde in sechs regelmäßige Schichten geordnet. Diese Schichtenstruktur hat ihre ganz eigene Logik. Zum Beispiel sitzen in den Schichten 3–5 Pyramidenzellen – ihr Zellkörper ähnelt Pyramiden –, die aus den Schichten 1–4 Signale empfangen und aus den Schichten 5–6 heraus ihre eigenen Impulse versenden.

Neben diesen horizontalen Strukturen gibt es in der Großhirnrinde auch vertikale Strukturen: Neuronen können säulenartige Gruppen bilden, die miteinander in hohem Maße verbunden sind. Diese Säulen – beim Menschen umfassen sie einige zehntausend Neuronen und sind ca. 2 mm hoch bei einem Durchmesser von 0,5 mm – stellen quasi Verarbeitungseinheiten, also Bausteine oder Module dar. Auch die Säulen kommunizieren miteinander, sodass typische Aktivierungsmuster entstehen, bei denen eng beieinanderliegende Säulen aktiv sind, während die umgebenden Säulen inaktiv sind.

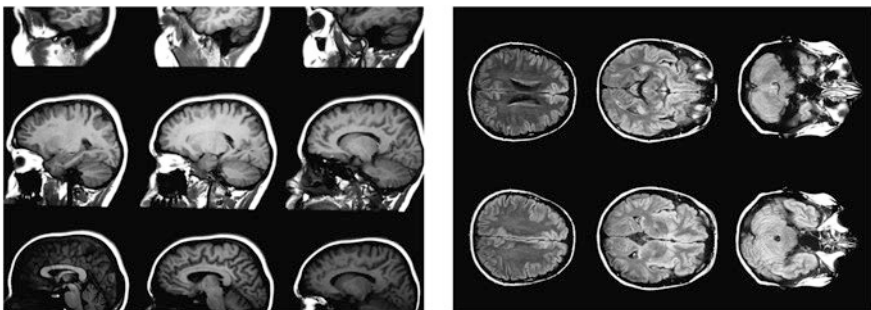


Abb. 2.40 Unser Gehirn weist vielfältige Strukturen auf. (© gmstockstudio/stock.adobe.com)

Natürlich ist auch in unserem Nervensystem die Rückkopplung das A und O. Eine Theorie des Gedächtnisses besagt, das unser Kurzzeitgedächtnis ähnlich funktioniert wie das Flipflop, sodass sich gewisse Aktivitätsmuster durch anhaltendes Feuern und Feedback erhalten. Das Langzeitgedächtnis funktioniert hingegen anders: Hier werden Verbindungen zwischen Neuronen aufgebaut oder verstärkt, d. h., die Struktur des Gehirns ändert sich. Dazu ist Plastizität erforderlich. Diese Formbarkeit ist insbesondere bei uns Menschen ausgeprägt: Unzählige Verbindungen werden im Laufe des Lebens verändert, vor allem in der Kindheit. Das trifft natürlich nicht auf alle Nervensysteme zu: Der Fadenwurm *Caenorhabditis elegans* besitzt genau 302 Nervenzellen, die fix zu einem Nervenring zusammengeschlossen sind; hier geht es nicht um kognitive Fähigkeiten, sondern um reine Koordination durch eine fest verdrahtete Verschaltung.

Plastizität, die Veränderbarkeit des Gehirns, ist folglich zentral für die Anpassungsfähigkeit, und – kaum verwunderlich – hat man diese Idee auch bei den Schaltungen aufgegriffen. Es gibt spezielle Bausteine, die sich intern beliebig verschalten und mit denen sich beliebige Musterzuordnungen realisieren und nachträglich auch verändern lassen. Bereits Ende der 1970er-Jahre kamen die ersten dieser Bausteine unter der Bezeichnung *programmable logic array* (PLA) auf den Markt. Allerdings ist diese Neuverdrahtung nur von außen möglich, die Schaltungen können sich nicht selbst modifizieren. Das bleibt – bislang – dem Gehirn vorbehalten.

Fazit

Es braucht nicht viel: einen kleinen Schalter, der einen Stromkreis aktivieren bzw. deaktivieren kann und selbst wiederum mittels Strom betrieben wird. Mit so einem Bauteil – in Form eines Relais oder Transistors – lässt sich eine Vielzahl an Stromkreisen miteinander verschalten und lassen sich Informationen verarbeiten. Dabei geht die Idee zurück auf die Turing-Maschine, bei der Zeichen schrittweise durch andere Zeichen ersetzt werden. So lassen sich beliebige Eingangsmuster auf Ausgangsmuster abbilden, und zwar einzig dadurch, dass sich elektrische Signale fortpflanzen und mit anderen kombiniert werden. Dieses Prinzip erinnert sehr stark an die Arbeitsweise des menschlichen Gehirns, das Informationen auf elektrochemischem Wege verarbeitet.

Durch die Einführung von Schleifen in der Verarbeitungs- bzw. Schaltungskette ergeben sich wiederum neue Möglichkeiten, z. B. jene, Informationen zu speichern. Das Flipflop ist das Kernelement der digitalen

Speicherelemente und überall in einem Computer anzutreffen. Und auch im Gehirn sowie in Zellen ist Rückkopplung ein zentrales, weit genutztes Prinzip, um die Lebensvorgänge selbstregulativ zu steuern.

Damit haben wir zwei der drei Kernprinzipien eines Computers, Automatik und Speicherfähigkeit, praktisch umgesetzt. Die Programmierbarkeit werden wir uns im nachfolgenden Kapitel anschauen, wo sich alles um den Prozessor dreht.

Zum Nach- und Weiterlesen

- Alon, U. (2006). *An introduction to systems biology: Design principles of biological circuits*. Boca Raton: Chapman & Hall/CRC.
- Beck, H., Anastasiadou, S., & Reckendorf, C. M. Z. (2015). *Faszinierendes Gehirn: Eine bebilderte Reise in die Welt der Nervenzellen*. Berlin: Springer Spektrum.
- Berg, J. M., Stryer, L., & Tymoczko, J. L. (2012). *Stryer Biochemie* (7. Aufl.). Berlin: Springer Spektrum.
- Floyd, T. L. (2012). *Digital fundamentals: A systems approach*. Upper Saddle River: Pearson Education.
- Hoffmann, D. W. (2014). *Grundlagen der Technischen Informatik* (4. Aufl.). München: Hanser.
- Thompson, R. (2016). *Das Gehirn: Von der Nervenzelle zur Verhaltenssteuerung* (3. Aufl.). Berlin: Springer.

Bildquellen

- Abb. 2.10: pioneer111, Bildnummer 102614417, <https://stock.adobe.com> (abgerufen am 30.05.2017)
- Abb. 2.28: Jeffrey J. Tabor, Rice University, <http://www.bactograph.org> (abgerufen am 28.05.2017)
- Abb. 2.37: Science Source/mauritus images, Bildnummer 05073338, <http://mauritus-images.com> (abgerufen am 30.05.2017)
- Abb. 2.40: gmstockstudio, Bildnummer 82959563, <https://stock.adobe.com> (links, abgerufen am 30.05.2017); gmstockstudio, Bildnummer 81772781, <https://stock.adobe.com> (rechts, abgerufen am 30.05.2017)

3

Verschlungene Pfade

Überblick

In diesem Kapitel geht um den Prozessor, das Herzstück – oder besser gesagt das Gehirn eines jeden Computers. Wir werden sehen, wie sich auf der Basis elektrischer Schaltungen, insbesondere den Gatter- und Flipflop-Bausteinen, eine kleine Datenfabrik bauen lässt, die erstens eine Folge elementarer Rechenschritte automatisiert abarbeiten kann und zweitens erlaubt, den Berechnungsweg individuell einzustellen. Ein langer Weg dahin, doch eine Analogie wird uns helfen: Ein Prozessor lässt sich, ebenso wie eine Zelle, im weitesten Sinne mit einer Küche vergleichen. Wir werden uns die Werkzeuge und Ablagen im Detail anschauen und schließlich auf einem Pfad miteinander verbinden. Darüber hinaus werden wir uns mit der Steuerung befassen, die den Datenfluss auf dem Pfad steuert. Ausgehend von einer fixen Berechnung werden wir den Rechenvorgang flexibilisieren und treten damit über von der Sphäre der Hardware in die Sphäre der Software: Wir schaffen die Maschinensprache, mit der sich Computer programmieren lassen. Zum Schluss befassen wir uns noch einmal mit der Kolibakterie als Stoffverarbeitungsfabrik und stellen sie dem Prozessor gegenüber.

Jeder Computer besitzt in seinem Innern eine Verarbeitungszentrale; dort findet das eigentliche Rechnen statt. Diese Zentrale ist in der Lage, elementare Rechenoperationen wie z. B. die Addition umsetzen und Zahlen zwischenzuspeichern. Darüber hinaus kann sie Folgen elementarer Rechenoperationen Schritt für Schritt abarbeiten und somit komplexe Berechnungen durchführen. Man könnte sie auch als nackten Computer bezeichnen oder als elektronisches Pendant zur Turing-Maschine, weil andere Komponenten wie Tastatur oder Bildschirm sie mit Daten versorgen bzw. die von ihr produzierten Daten abnehmen müssen.

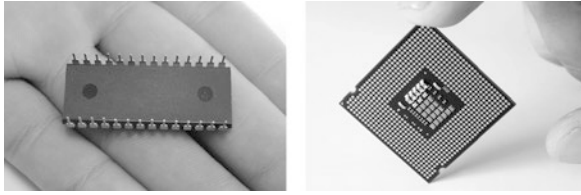


Abb. 3.1 Wie Prozessoren in den 1970er Jahren (links) und 40 Jahre später aussahen (rechts). (© Tobif82/stock.adobe.com (links), olvius/stock.adobe.com (rechts))

Geläufiger ist Ihnen vermutlich der Begriff des Prozessors, jenes Bauteils, auf dem diese zentrale Verarbeitungseinheit – im Englischen heißt sie *central processing unit* oder kurz CPU – untergebracht ist. Prozessoren waren nicht immer eigenständige Bausteine, erst in den 1970er Jahren gelang es, sie in Miniaturbauweise als sogenannte Mikroprozessoren zu fertigen. Die ersten Mikroprozessoren integrierten einige Tausend Transistoren auf einem kaum fingernagelgroßen Halbleiter-Chip, der in einem Plastikgehäuse eingefasst war und über seitlich angebrachte Metallbeinchen mit der Außenwelt kommunizierte (Abb. 3.1 links). Bereits 30 Jahre später waren es viele Millionen Transistoren, die so auf winzigstem Raum platziert werden konnten, und 2010 wurde schließlich die Milliarden Grenze geknackt (Abb. 3.1 rechts). Heutzutage werden in einem Prozessor zumeist mehrere Verarbeitungseinheiten nebeneinander verbaut.

Wir werden im Folgenden einen einfachen Prozessor (mit einer einzigen Verarbeitungseinheit) entwerfen – eine anspruchsvolle Aufgabe, die wir Schritt für Schritt meistern werden. Wenn es wie im vorherigen Kapitel hin und wieder etwas zäh wird, so ist das der Komplexität des Themas geschuldet. Behalten Sie dies im Hinterkopf und lassen Sie sich nicht entmutigen, sollte Ihnen zwischendurch der Schädel brummen. Zur Vorbeugung nehmen wir einen sanften Einstieg und beginnen bei etwas Vertrautem: der Küche.

Vorneweg: das Küchen-Einmaleins

Was eine Küche mit einem Computer gemein hat? Nun, im weitesten Sinn handelt es sich in beiden Fällen um Verarbeitungsanlagen, einmal im kleinen, einmal im großen Stil. Aufbau und Arbeitsweise sind ähnlich, wie Sie gleich sehen werden.

Eine Fabrik im Kleinen: die Küche

Um aus einer Vielfalt von Zutaten ein Gericht zu zaubern, braucht es einerseits geeignete Werkzeuge und andererseits eine fähige Köchin oder einen fähigen Koch sowie ein Rezept (Abb. 3.2). Mit anderen Worten: Es bedarf einer Infrastruktur und einer Ablaufsteuerung.

Die Infrastruktur definiert das Spektrum an Möglichkeiten, nach denen Speisen zubereitet werden können. Da gibt es zunächst einmal die Küchengeräte und -werkzeuge, mit denen die Zutaten bearbeitet werden: der Herd zum Erhitzen, das Messer mit Schneidebrett zum Zerkleinern, der Quirl zum Vermengen usw. Daneben braucht es auch Aufbewahrungsbehältnisse wie Schüsseln, um zubereitete Teile des Essens für spätere Arbeitsschritte zwischenzulagern, und einen Vorratsspeicher, um die gesamten Zutaten und fertigen Speisen aufzubewahren. Schließlich gehört, besonders in einer Großküche, der ganze Raum dazu, in dem die erwähnten Elemente angeordnet und über geeignete Verbindungswege erreichbar sind.

Die Ablaufsteuerung ist verantwortlich für die Dynamik und legt fest, in welcher Reihenfolge welche Zutaten mit welchen Werkzeugen verarbeitet werden. Stehen wir selbst in der Küche, sind wir es, die die Infrastruktur



Abb. 3.2 Die Küche als Beispiel einer Verarbeitungsstätte. (© Susanne Staubli, Eckart Zitzler)

nutzen und die einzelnen Zubereitungsschritte vollziehen. Der Ablauf folgt einem bewährten Schema, dem Rezept, und wir sind – salopp gesagt – Ausführende der Kochvorschrift. Sicherlich, wir können auch improvisieren, doch je weniger wir ein Rezept kennen, desto mehr ist das haargenaue Befolgen der im Rezept definierten Arbeitsschritte nötig.

Letztlich ist es das Zusammenspiel von Infrastruktur und Ablaufsteuerung, welches die Kochkunst ausmacht. Dieses allgemeine Prinzip finden wir überall, selbst in der Musik: Der Flügel setzt den Rahmen der möglichen Klänge, die Pianistin spielt auf der Klaviatur und setzt Töne zu einer Melodie zusammen. Dabei hat es immer unveränderliche Anteile, z. B. die Küchengeräte bzw. die Flügelmechanik, und veränderliche Anteile, z. B. das Rezept bzw. die Partitur. Kaum verwunderlich, dass es beim Prozessor ganz ähnlich ist.

Was Prozessoren und Zellen mit Küchen gemein haben

In seiner einfachsten Form weist ein Prozessor drei Infrastruktur-Bestandteile auf: ein Rechenwerk, verschiedene Speicherbausteine und Leitungen, die diese Elemente miteinander verbinden (Abb. 3.3 rechts).

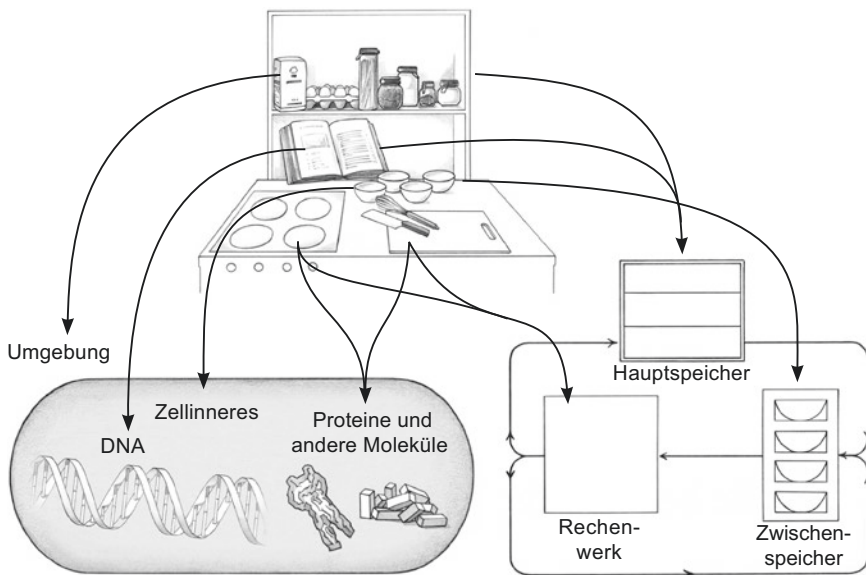


Abb. 3.3 Analogien zwischen Küche und Prozessor (rechts) sowie Küche und Zelle (links). (© Susanne Staubli, Eckart Zitzler)

Das Rechenwerk ist die Einheit, die die elementaren Rechenoperationen wie Addition und Subtraktion durchführt und somit die möglichen Verarbeitungsschritte festlegt; es verkörpert quasi Herd, Messer und Quirl in einem.

Schnelle Zwischenspeicher, Register genannt, dienen dazu, Teilergebnisse in einem Berechnungsstrang temporär festzuhalten; sie sind die Schüsseln in der Datenküche. Weil es von ihnen nur wenige gibt, existiert daneben noch eine Art Vorratskammer: der Hauptspeicher. Er kann größere Datenmengen aufnehmen und befindet sich streng genommen außerhalb des Prozessors, weil sich so Größe und Geschwindigkeit je nach Anwendungszweck anpassen lassen. Um die Dinge nicht unnötig zu verkomplizieren, gehen wir hier davon aus, dass der Hauptspeicher Teil des Prozessors ist.

Die Leitungen schließlich definieren die Pfade, entlang derer die Daten zwischen den Rechen- und Speicherkomponenten transportiert werden können. Aus diesem Grund wird die Gesamtschaltung, die die Infrastruktur eines Prozessors realisiert, auch Datenpfad genannt.

Für die Ablaufsteuerung ist in einem Prozessor eine weitere Schaltung zuständig, die als Kontrollpfad bezeichnet wird. Der Kontrollpfad besteht im Wesentlichen aus dem Steuerwerk, das dem Koch entspricht und dessen Aufgabe darin besteht, den Rechenvorgang gemäß einem Programm abzuwickeln. Das Programm – das Gegenstück zum Rezept – ist im Hauptspeicher abgelegt und damit beliebig austauschbar.

Spannend ist in diesem Zusammenhang wiederum der Vergleich zur Kolibakterie aus Kap. 1 oder zu lebenden Zellen allgemein. Die Küchenelemente lassen sich auch dort wiederfinden (vgl. Abb. 3.3 links): Das Zellinnere fungiert u. a. als Aufbewahrungsort, die Proteine (und auch andere Moleküle) bewirken die Bearbeitung von Stoffen und die DNA stellt die Rezeptur des Verarbeitungsvorgangs zur Verfügung.

Selbstverständlich ist es etwas anderes, ob elektrische Signale oder Moleküle verarbeitet werden. Doch im übertragenen Sinne sind die Operationen recht ähnlich, wie in Abb. 3.4 ersichtlich ist. Im Prozessor werden Zahlen miteinander vermengt, wenn wir eine Addition ausführen, oder eine Zahl lässt sich aufteilen, wenn wir nur die untere oder die obere Hälfte betrachten wollen; es gibt sogar die Möglichkeit, eine Zahl umzuwandeln – so wie wir eine Speise durch Kochen transformieren –, indem bei einer Dualzahl jede Ziffer durch ihr Gegenstück ersetzt wird. Demgegenüber gibt es in der Zelle Proteine zum Verbinden, zum Zerteilen, zum Umwandeln usw. Wir sehen also, das Prinzip ist dasselbe.

Wir kommen am Ende dieses Kapitels noch einmal auf die Parallelen zwischen Prozessor und Zelle zu sprechen, wenn wir den biologischen

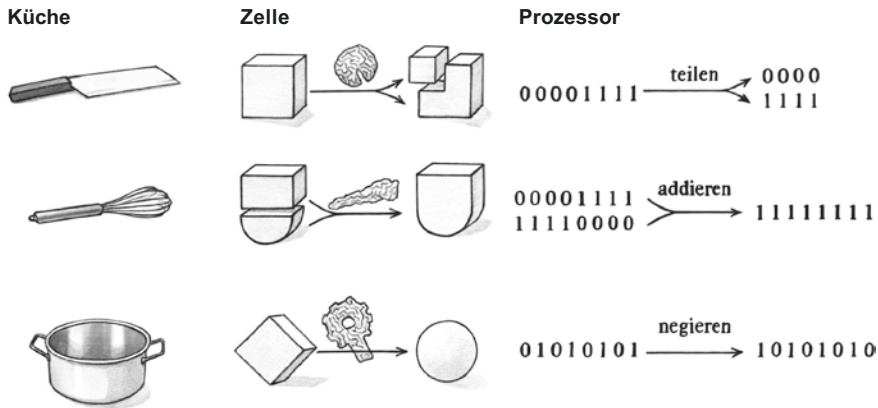


Abb. 3.4 Arten von Verarbeitungswerkzeugen in Küche, Zelle und Prozessor. (© Susanne Staubli, Eckart Zitzler)

Stoffwechsel beleuchten. Doch vorher gilt es, den skizzierten Aufbau der zentralen Verarbeitungseinheit zu verfeinern und zu verstehen, wie das Ganze im Einzelnen funktioniert.

Pfade für Daten

Zunächst einmal konstruieren wir den Datenpfad für unseren einfachen Beispielprozessor, ausgehend vom Rechenwerk über die Speicherbausteine bis zu den Transportleitungen.

Ein Werk zum Rechnen

Das Rechenwerk ist die zentrale Baugruppe in einem Prozessor, sie setzt die elementaren Rechenoperationen um. Über die Eingangsleitungen werden die zu verarbeitenden Zahlen sowie die Art der Rechenoperation übermittelt, über die Ausgangsleitungen lassen sich sowohl das Ergebnis selbst als auch Informationen über das Ergebnis (z. B. ob die verfügbare Stellenanzahl überschritten wurde) ablesen. Insofern ähnelt das Rechenwerk dem Taschenrechner aus Kap. 2, nur dass die Schalter und Lampen durch hinein- bzw. hinausführende Leitungen ersetzt sind.

Das Rechenwerk, welches wir im Folgenden verwenden werden, ist sehr einfach gehalten (siehe Abb. 3.5) – ganz im Gegensatz zu realen Mikroprozessoren. Es nimmt zwei achtstellige Dualzahlen entgegen

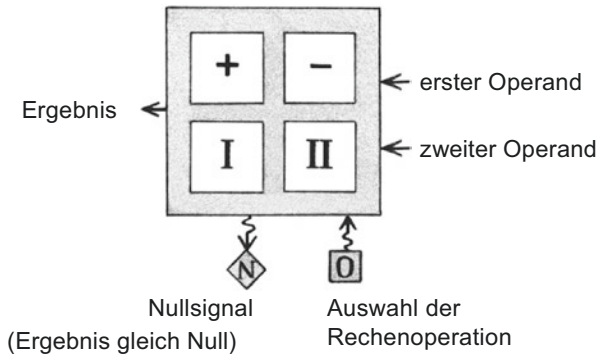
Rechenwerk

Abb. 3.5 Das Rechenwerk als eigenständiger Baustein. (© Susanne Staubli, Eckart Zitzler)

und liefert als Ausgabe das Ergebnis in Form einer achtstelligen Dualzahl sowie die Information, ob das Ergebnis gleich null ist oder nicht. Die zwei Eingangszahlen können wahlweise addiert oder subtrahiert werden, Überläufe – wenn mehr als acht Ziffern zur Darstellung nötig sind – werden hierbei ignoriert. Zudem lassen sich beide Eingangszahlen einzeln, sozusagen unverarbeitet durchschalten; so lässt sich feststellen, ob eine Null vorliegt. Diese vier Rechenoperationen werden wir mit den Symbolen +, –, I und II bezeichnen. Bsp. 3.1 zeigt exemplarisch, was sie bewirken. Viel kann unser Rechenwerk folglich nicht, aber das braucht uns nicht zu stören, schließlich steht das Verstehen im Vordergrund.

Beispiel 3.1: Das Rechenwerk in Aktion

Nehmen wir an, die zwei Dualzahlen 00010000 und 00000001 (dezimal 8 und 1) würden dem Rechenwerk als Eingabe zugeführt. Je nach Wahl der Rechenoperation gibt es verschiedene Resultate: Bei der Addition (+) ist die Ausgabe 00010001, bei der Subtraktion (–) ist sie 00001111, beim Durchschalten des ersten Operanden (I) ist sie 00010000 und beim Durchschalten des zweiten Operanden (II) ist sie 00000001. In allen vier Fällen signalisiert das Rechenwerk, dass das Ergebnis ungleich 00000000 ist.

Entsprechend simpel ist auch die Umsetzung als Baustein. Das Rechenwerk führt alle möglichen Rechenoperationen parallel durch, und nur das benötigte Ergebnis wird zum Ausgang weitergereicht (Bsp. 3.2).

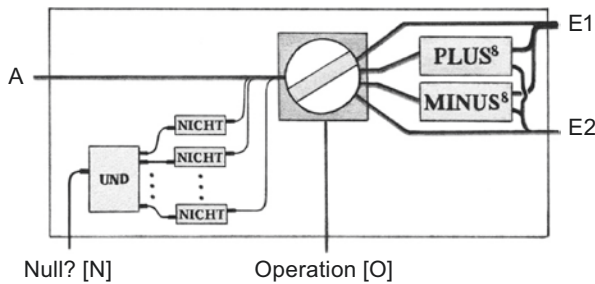


Abb. 3.6 Das Rechenwerk als zusammengesetzte Schaltung; Plus- und Minus-Baustein sind jeweils mit einer 8 markiert, weil sie achtstellige Dualzahlen verarbeiten. (© Susanne Staubli, Eckart Zitzler)

Beispiel 3.2: Das Rechenwerk als Schaltung

Das Rechenwerk besitzt folgende Schaltungskomponenten (siehe Abb. 3.6): jeweils eine für die Plus- und die Minus-Operation – sie verarbeiten achtstellige Dualzahlen und werden gemäß den Prinzipien aus Kap. 2 konstruiert –, eine Drehschalter-Komponente in der Mitte, die das Ergebnis auswählt, und eine Gatterkombination auf der linken Seite, die das Ergebnis mit der Null vergleicht. Die beiden Operanden werden über die zwei Eingänge E1 und E2 – die dicken Linien in der Abbildung stehen für Leitungsbündel mit jeweils acht Leitungen – eingespeist und anschließend den zwei Rechenbausteinen und dem Drehschalter zugeführt. Letzterer symbolisiert eine Weiche, über die sich eines der vier Leitungsbündel durchschalten lässt; welches Leitungsbündel am Ausgang A sichtbar wird, legen die mit „Operation“ bezeichneten Leitungen fest (auch hier sind aus Gründen der Übersichtlichkeit zwei Leitungen in einer Linie dargestellt). Die nachgelagerte Schaltung, ein UND-Gatter mit negierten Eingängen, liefert über eine separate Ausgangsleitung (die „Null?“-Leitung) genau dann eine 1, wenn die resultierende Dualzahl die Null darstellt.

Damit hätten wir die Kochwerkzeuge der Datenküche realisiert. Jetzt kommen wir zu den Schüsseln und der Vorratskammer.

Zahlenbehälter

Ein Register dient dazu, eine Dualzahl zwischenspeichern. Die gespeicherte Zahl kann über die Ausgangsleitungen abgegriffen werden, über entsprechende Eingangsleitungen lässt sich eine neue Zahl in das Register einspeisen. Demgegenüber ist der Hauptspeicher darauf ausgelegt, eine Vielzahl von Dualzahlen zu speichern. Sie können sich ihn als eine fächerartige Anordnung von vielen Registern vorstellen, wobei jedes Register eine eindeutige Nummer – man sagt auch Adresse – besitzt. Die Funktionsweise

ist gleichartig zu der eines Registers, nur dass zusätzlich noch anzugeben ist, welches Speicherfach gerade angesprochen wird.

Die Speicherbausteine, die in unserem Beispiel-Prozessor zum Einsatz kommen, sind in Abb. 3.7 skizziert. Analog zum Rechenwerk, das achtstellige Dualzahlen verarbeitet, sind die gespeicherten und zu speichernden Zahlen jeweils über Bündel von acht Leitungen repräsentiert. Der Hauptspeicher besitzt 256 Speicherzellen, die von 0 bis 255 durchnummeriert sind; die Speicherzelle, auf die sich Lese- und Schreibvorgang beziehen sollen, wird über acht Adressleitungen eingestellt. Bitte beachten Sie, dass beim Hauptspeicher die Eingangsleitungen links hineingehen und die Ausgangsleitungen rechts hinausführen – einfach weil wir es in der Datenpfad-Skizze (Abb. 3.3) auch so gehandhabt haben.

Eine Besonderheit gibt es beim Schreibvorgang: Es braucht zwei Signale, um das Überschreiben des Register- bzw. Speicherzelleninhalts auszulösen. Ein Schreibsignal zeigt an, ob die gespeicherte Zahl überhaupt überschrieben werden soll. Die Schreibaufforderung kommt jedoch nur zum Tragen, wenn ein zweites Signal, das Taktsignal, vorliegt. Letzteres ist rein zeitgesteuert und wird in regelmäßigen Zeitabständen gesetzt. Es bestimmt also, wann das Überschreiben grundsätzlich stattfinden kann, und dient dazu, die gesamten Speicherbausteine im Datenpfad zu synchronisieren: Schreibvorgänge finden immer gleichzeitig und nur zu Beginn eines Takts statt. Die Trennung von Schreibbefehl und zeitlicher Synchronisation erleichtert die Konstruktion des Prozessors. Wir werden das Thema im Zusammenhang mit dem Kontrollpfad noch eingehender beleuchten.

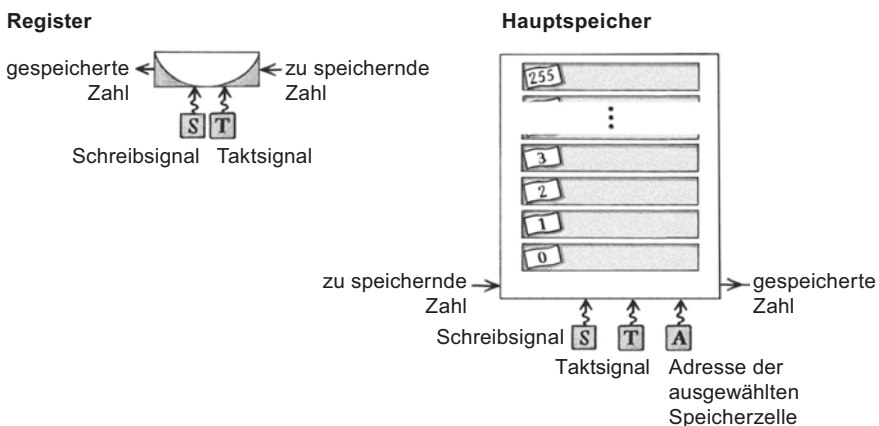


Abb. 3.7 Zwei Typen von Speicherbausteinen: Register und Hauptspeicher.
(© Susanne Staubli, Eckart Zitzler)

Werfen wir noch einen Blick in das Innere der Speicherbausteine. Sowohl die Register als auch der Hauptspeicher sind auf der Basis desselben Speicherelements konstruiert, das beim erweiterten Taschenrechner in Kap. 2 zum Einsatz kam: dem Flipflop. Ein Register ist letztlich nichts anderes als eine Kombination von acht Flipflops, deren Schreibleitungen miteinander verknüpft sind (siehe Bsp. 3.3).

Beispiel 3.3: Ein Register von innen

In Abb. 3.8 sehen Sie die Flipflops nebeneinander angeordnet. Diese sind synchronisiert, d. h., sie erhalten den Befehl zum Überschreiben des Inhalts über dieselbe Leitung. Letztere ist mit dem Ausgang eines UND-Gatters verbunden, das die beiden zum Schreiben notwendigen Signale empfängt. Nur wenn sowohl auf der „Schreiben“-Leitung als auch auf der „Takt“-Leitung eine 1 anliegt, werden alle Flipflop-Inhalte überschrieben.

Der Aufbau des Hauptspeichers ist etwas komplizierter, weil wir zusätzlich noch die Logik einbauen müssen, die die jeweilige Speicherzelle über die Adressleitungen ansteuert. Wie das konkret als Schaltung realisiert wird, erläutert Bsp. 3.4.

Beispiel 3.4: Funktionsweise des Hauptspeichers

In Abb. 3.9 unten ist der Hauptspeicheraufbau skizziert – aus Gründen der Übersichtlichkeit für einen kleineren Speicher, der nur vier einstellige Dualzahlen aufnehmen kann. Die vier Speicherzellen – jede von ihnen umfasst ein Flipflop – sind untereinander angeordnet, die entsprechenden Adressen sind als Dualzahlen rechts aufgeführt. Die Komponenten auf der rechten Seite fassen die zwei „Adresse“-Leitungen in einer Leitung pro Zeile zusammen; die Leitung pro Zeile ist nur dann auf 1 gesetzt, wenn genau die spezifische Adresse der Speicherzelle ausgewählt wurde. Jeweils ein UND-Gatter vor und nach jedem Flipflop garantiert, dass das Schreibsignal bzw. der Ausgabewert nur für diejenige Speicherzelle durchgeschaltet wird, die über die Adressleitungen ausgewählt ist.

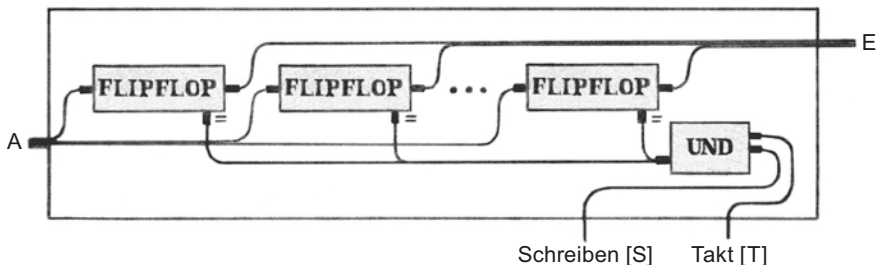


Abb. 3.8 Aufbau eines Registers. (© Susanne Staubli, Eckart Zitzler)

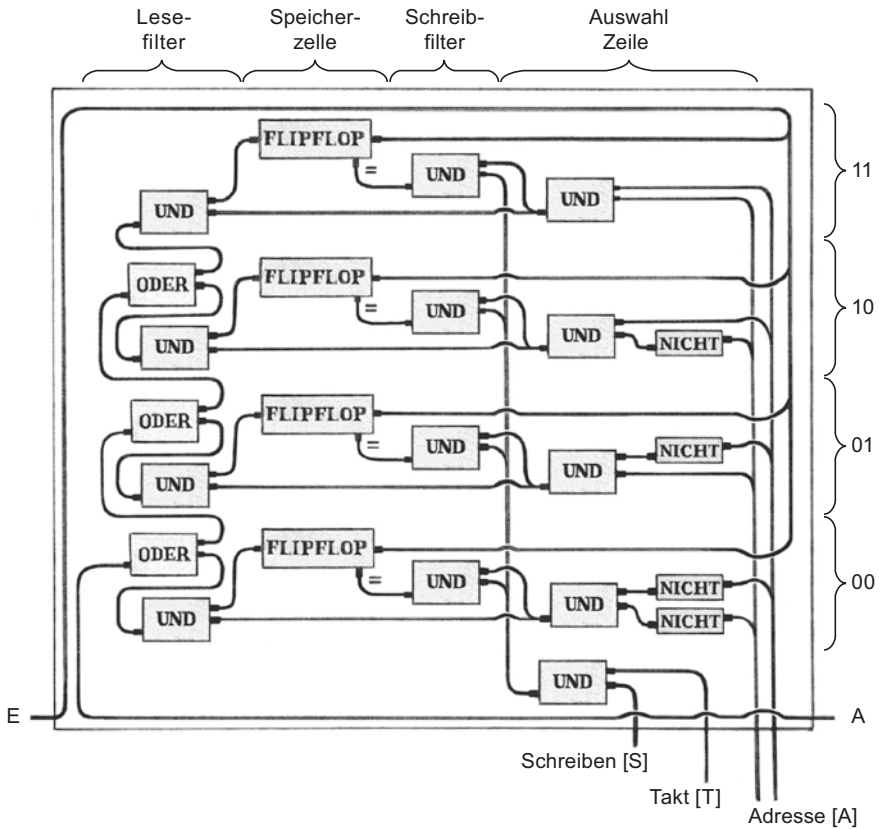


Abb. 3.9 Aufbau des Hauptspeichers. (© Susanne Staubli, Eckart Zitzler)

Vielleicht fragen Sie sich an dieser Stelle, warum es eigentlich zwei Typen von Speicherbausteinen braucht. Die Turing-Maschine kommt schließlich auch mit einem Band bzw. Arbeitsspeicher aus. Letztlich geht es um den Kompromiss zwischen Speicherkapazität und Zugriffsgeschwindigkeit. Register sind darauf ausgelegt, die Operanden für die Rechenoperationen zur Verfügung zu stellen und Teilergebnisse für nachfolgende Rechenschritte zwischenzuspeichern. Damit die Berechnung schnell ablaufen kann, befinden sich die Register nahe beim Rechenwerk auf dem Prozessor. Der Platz auf dem Chip ist jedoch begrenzt, große Speicher lassen sich dort neben den Verarbeitungseinheiten nicht unterbringen. Aus diesem Grund ist der Hauptspeicher ausgelagert in Form von externen Speicherbausteinen – auch wenn wir diesen Umstand der Einfachheit halber außer Acht lassen. So kann er große Datenmengen aufnehmen und ist erweiterbar. Das Wichtigste jedoch:

Er kann im Gegensatz zu den Registern auch von anderen Komponenten des Computers benutzt werden; der Hauptspeicher fungiert als Vermittler zwischen Prozessor und Außenwelt.

Transportgleise

Jetzt können wir den Datenpfad für den fiktiven Prozessor zusammensetzen und die entsprechenden Transportwege ergänzen, wie es in Abb. 3.10 dargestellt ist. All die Komponenten, die wir im Vorfeld besprochen haben, finden Sie wieder: das Rechenwerk, vier Register und den Hauptspeicher. Diese sind über achtadrige Leitungen – in der Abbildung der Übersichtlichkeit halber als einzelne Linien wiedergegeben – miteinander verbunden. Darüber hinaus sehen Sie unter den vier Registern ein fünftes Element, das wie ein

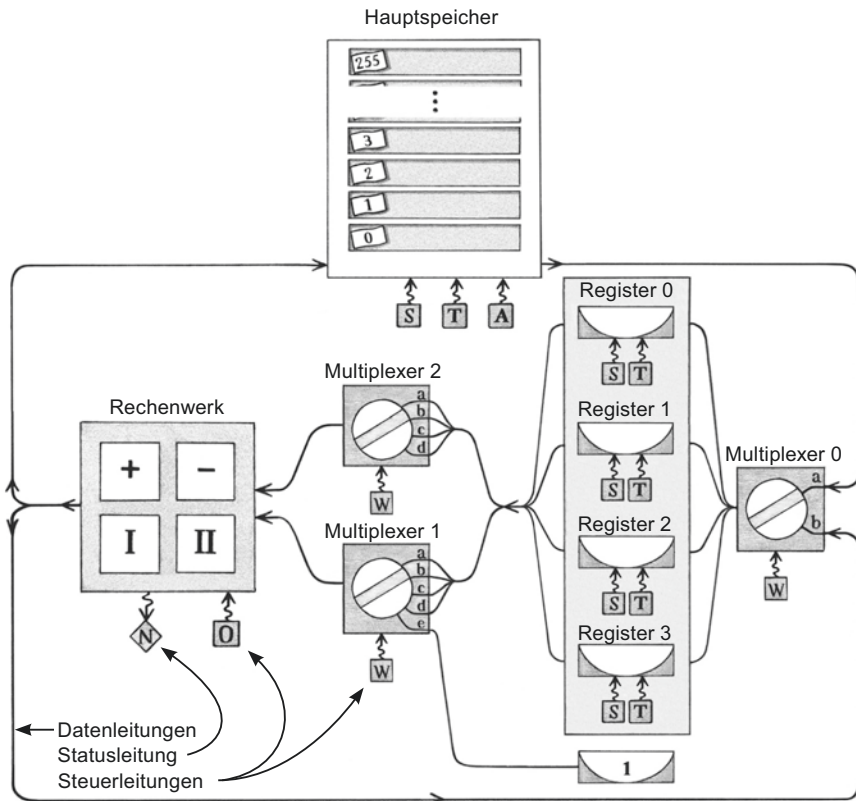


Abb. 3.10 Die Grundversion des Datenpfads für den Beispiel-Prozessor. (© Susanne Staubli, Eckart Zitzler)

Register aussieht. Es handelt sich dabei um eine fixe Schaltung, die keine Eingabeleitungen hat, nur acht Ausgangsleitungen, die konstant die Dualzahl 00000001 ausgeben. Wir können uns das vorstellen als voreingestelltes Register, das sich nicht überschreiben lässt, sondern konstant den Wert 1 beinhaltet. Wofür wir die Zahl 1 brauchen, wird später noch klarer werden. Schließlich hat es noch drei als Drehschalter symbolisierte Weichen, auch Multiplexer genannt – wofür sind die gut?

Dem Multiplexer sind wir bereits beim Rechenwerk über den Weg gelaufen: Er ist eine Art Türsteher-Baustein, weil er von mehreren Eingangssignalen nur eines, nämlich das ausgewählte, hindurchlässt. Die Funktionalität benötigen wir beispielsweise, wenn wir die Registerausgänge mit den Eingängen des Rechenwerks verbinden wollen. Über die Multiplexer 1 und 2 können wir jeweils festlegen, welche Registerinhalte als Operanden für den nächsten Rechenschritt verwendet werden sollen. Multiplexer 0 hat eine ähnliche Aufgabe: Er steuert, ob der Ausgang des Hauptspeichers oder des Rechenwerks an die Registereingänge gelegt wird. Wenn Sie wissen möchten, wie der Multiplexer als Schaltung umgesetzt wird, hilft Ihnen Bsp. 3.5 weiter.

Beispiel 3.5: Der Multiplexer konkret

In Abb. 3.11 sehen Sie einen vereinfachten Multiplexer mit zwei Eingangsleitungen, einer Ausgangsleitung sowie einer „Wahl“-Leitung, die festlegt, welcher von beiden Eingängen durchgeschaltet wird. Nehmen wir an, die „Wahl“-Leitung sei auf 0 gesetzt. Dann liegt am oberen UND-Gatter eine 1 an und am unteren UND-Gatter eine 0; nur wenn die obere Eingangsleitung eine 1 führt, wird auch der Ausgang auf 1 gesetzt. Analog wird der Wert der unteren Eingangsleitung durchgereicht, wenn die „Wahl“-Leitung auf 1 gesetzt wird.

Dieses Prinzip ist achtmal zu duplizieren, will man zwischen zwei achtstelligen Dualzahlen auswählen. Für jede Stelle braucht es einen vereinfachten Multiplexer, und die „Wahl“-Leitung wird an jeden Baustein mit dem gleichen Signal angelegt.

Sollen noch weitere Eingänge hinzukommen, dann kann man mehrere des vereinfachten Multiplexers hintereinanderschalten. Die erste Ebene von Multiplexern unterscheidet zwischen jeweils zwei Eingängen, die nächste Ebene zwischen jeweils zwei vorgeschalteten Multiplexern usw. So lassen sich auch Multiplexer bauen, um zwischen beliebig vielen Eingängen hin- und herzuschalten.

An sich kann man den Datenpfad mit einem Schienennetz vergleichen (Abb. 3.12), auf dem die Daten hin- und hertransportiert werden. Die Datenleitungen entsprechen den Schienen, die Multiplexer sind die Weichen

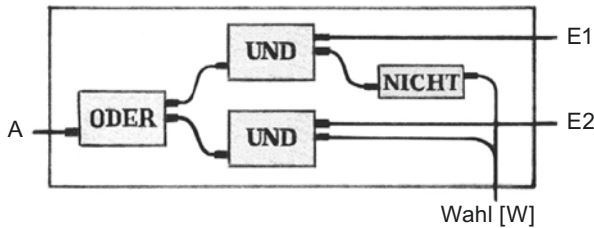


Abb. 3.11 Die elektrische Weiche: Wie sich mittels Gatter-Bausteinen ein Multiplexer realisieren lässt. (© Susanne Staubli, Eckart Zitzler)



Abb. 3.12 Schienennetze sind Alltagsbeispiele für Datenpfade. (© Judith Büsser)

und die anderen Bauelemente sind die Umschlagplätze. Darüber hinaus hat es noch Signale, die den Verkehrsfluss steuern. Im Datenpfad sind das die mittels Rechtecken markierten Steuerleitungen. Statusleitungen – diese sind mit einer Raute gekennzeichnet – liefern hingegen Informationen über den Verkehrsfluss, so wie beispielsweise die Warnlampen an Bahnübergängen.

Selbstverständlich muss der dargestellte Datenpfad nicht so aussehen, wie er aussieht. Letztlich sind es Designentscheidungen, die darüber bestimmen, welche Art von Berechnungen der Prozessor durchführen kann. Je mehr Register eingebaut, je mehr Rechenoperationen unterstützt werden, desto mächtiger, aber desto komplexer, desto stromfressender, desto heißer,

desto größer usw. wird der Prozessor. Hier versuchen Ingenieure permanent an die physikalischen Grenzen zu gehen: möglichst viel Leistung auf kleinstem Raum. Und auch die Wahl der Stellen für die intern zu verarbeitenden Zahlen spielt eine wichtige Rolle.

Je mehr Stellen die intern verarbeiteten Zahlen aufweisen, desto größere Zahlen können in gleicher Zeit verarbeitet werden. Übersteigt die Zahlengröße die interne Darstellungsgröße im Prozessor, so müssen elementare Rechenschritte wie Addieren aufwändig in die Addition von Zahlenkomponenten zerlegt werden. Aus diesem Grund möchte man die Stellenzahl möglichst groß halten, was wiederum zu besagten Platzproblemen führt. Doch die Entwicklung ist rasant: Erste Mikroprozessoren arbeiteten noch mit vierstelligen Dualzahlen, 20 Jahre später lassen sich bereits 64-stellige Dualzahlen verarbeiten. Man spricht in diesem Zusammenhang auch von 4-Bit- bzw. 64-Bit-Prozessoren; die Anzahl der Bits bezeichnet dabei die Anzahl der Stellen der zu verarbeitenden Dualzahlen.

Bevor wir weitergehen, noch eine Randbemerkung: Wie bei der Turing-Maschine blenden wir den Aspekt der Eingabe- und Ausgabegeräte hier aus. Wir gehen einfach davon aus, dass wir irgendwie von außen auf den Hauptspeicher zugreifen, also Zahlen ablegen und auslesen können.

Nicht Steuermann, sondern Steuerwerk

Damit kommen wir nun zur Ablaufsteuerung, dem Kontrollpfad. Als Erstes konzentrieren wir uns auf die Frage, wie sich eine spezifische Berechnung mittels einer Schaltung automatisieren lässt. Später werden wir den Kontrollpfad dann dahingehend erweitern, dass sich beliebige Berechnungsvorgänge – also Programme – ausführen lassen.

An den Fäden ziehen – und rechnen

Der Datenpfad stellt die Infrastruktur zum Rechnen zur Verfügung. Der Kontrollpfad nutzt diese Infrastruktur, indem er über bestimmte Leitungen – die Steuerleitungen – die einzelnen Komponenten und deren Zusammenspiel steuert: die Operation des Rechenwerks, die Weichenstellungen der Multiplexer, das Verhalten der Register usw. Wenn wir uns die Steuerleitungen als Fäden vorstellen, über die wir kontrollieren können, was im Datenpfad passiert, dann erinnert das unweigerlich an eine

Marionette. Der Marionettenspieler, und um den geht es hier, steuert durch geschicktes, koordiniertes Ziehen oder Erschlaffenlassen der Fäden den Berechnungsvorgang. Und wenn wir uns einmal unser eigenes Nervensystem und die Nervenbahnen in unserem Körper vor Augen führen, dann haben wir zuweilen selbst den Eindruck, da sitze ein Marionettenspieler in unserem Kopf – nur sind wir Marionette und Marionettenspieler gleichzeitig (Abb. 3.13).

Wenn der Kontrollpfad nun einen Berechnungsablauf vollzieht, dann heißt das eigentlich: Er arbeitet eine Liste von Schritten ab, wobei jeder Schritt eine bestimmte Belegung der Steuerleitungen repräsentiert. Ein Rechenschritt – das, was bei der Turing-Maschine durch eine Tabelle beschrieben wird – ist demnach nichts anderes als eine Steuerleitungsbelegung, eine Zuweisung von 0 oder 1 zu jeder einzelnen Steuerleitung. Ein Rechenschritt kann beispielsweise daraus bestehen, die Dualzahlen aus zwei Registern mittels des Rechenwerks zu addieren und die Summe wiederum in einem Register abzulegen. Abgeschlossen ist ein Rechenschritt dann, wenn das produzierte Zwischenresultat gespeichert ist, im obigen Fall die Summe in das Zielregister geschrieben wurde. Letzteres wird durch das bereits erwähnte Taktsignal erreicht: Es kommt am Schluss jedes Rechenschritts und veranlasst alle Speicherbausteine, bei denen das Schreibsignal gesetzt ist, einen neuen Wert zu übernehmen. Bsp. 3.6 demonstriert die Idee an einem konkreten Rechenschritt.

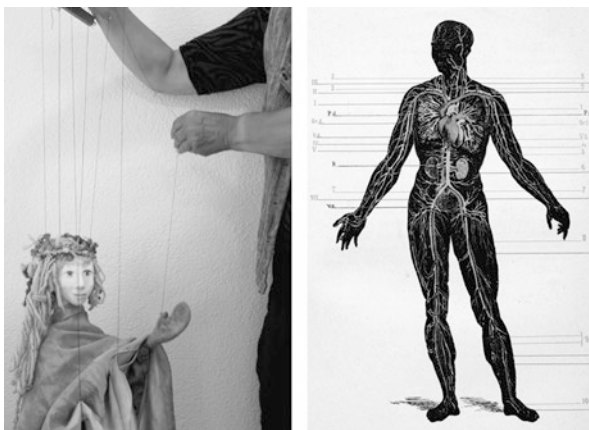


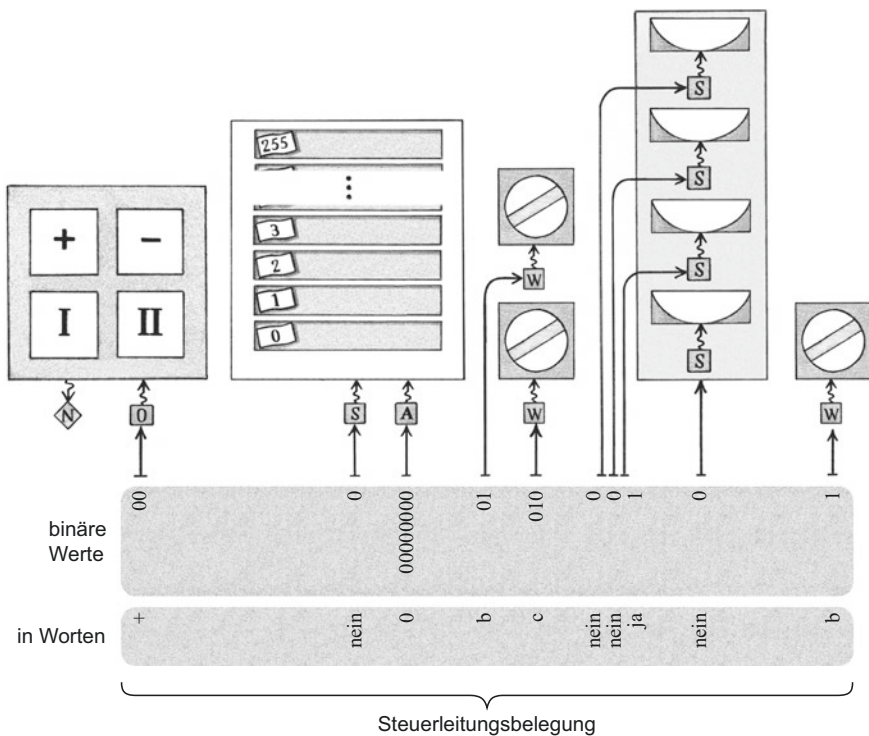
Abb. 3.13 Eine Marionettenspielerin und unser Nervensystem als Analogien zum Kontrollpfad. (© Judith Büsser (links), ilbusca/Getty Images/iStock (rechts))

Beispiel 3.6: Steuerleitungsbelegungen sind elementare Rechenoperationen

Nehmen wir an, in den Registern 1 und 2 seien die Zahlen 5 bzw. 0 gespeichert und wir wollten diese addieren und das Ergebnis wiederum in Register 2 ablegen. Wie müssen wir die Fäden ziehen, um diese Rechenoperation zu realisieren?

Abb. 3.14 zeigt die entsprechende Steuerleitungsbelegung. Damit die zwei Registerinhalte zum Rechenwerk gelangen, muss Multiplexer 1 den Eingang c und Multiplexer 2 den Eingang b durchschalten. (Anmerkung: Es ginge auch umgekehrt.) Das Rechenwerk ist mittels seiner Steuerleitungen auf + einzustellen. Da das Ergebnis wieder Register 2 zugeführt werden soll, ist Multiplexer 0 so zu konfigurieren, dass Eingang b durchgeschaltet wird. Des Weiteren müssen wir für Register 2 das Schreibsignal setzen; sobald wir dann zusätzlich das Taktsignal senden, übernimmt Register 2 den Wert an seinem Eingang. Wichtig ist dabei, dass der Schreibimpuls erst erfolgt, wenn die Signale das Rechenwerk passiert haben, also nicht zu früh.

Die anderen Steuerleitungen können wir nach Belieben setzen, mit einer Ausnahme: Wir müssen sicherstellen, dass alle anderen Speicherelemente, die nicht verändert werden sollen, ihre Werte behalten. Für diese Speicherbausteine ist das Schreibsignal explizit abzustellen.



Merken wir uns also: Eine Steuerleitungsbelegung realisiert im Datenpfad einen Rechenschritt, der abgeschlossen wird durch das Taktsignal, das die Sicherung des Ergebnisses einleitet. Eine Sequenz solcher elementarer Schritte definiert eine Berechnung. Die Abfolge der Rechenschritte ist in der Regel nicht fix, sondern hängt von den Eingabedaten sowie Zwischenergebnissen ab. Das haben wir bereits bei der Turing-Maschine gesehen: Dort lässt sich pro Tabelle und gelesenen Zeichen angeben, welche Tabelle als Nächstes verwendet werden soll. Diese Anpassungsfähigkeit erreichen wir im Prozessor durch die Statusleitungen, mit denen der Kontrollpfad in den Datenpfad hineinhorchen und situativ entscheiden kann, welcher Rechenschritt nun auszuführen ist. Man könnte das vergleichen mit dem Abschmecken, das der Koch bei der Zubereitung eines Gerichts vornimmt. In unserem fiktiven Prozessor gibt es nur eine Statusleitung, nämlich jene, die vom Rechenwerk ausgeht und übermittelt, ob das Ergebnis gleich null ist. Im nachfolgenden Beispiel können Sie sie in Aktion erleben.

Malnehmen mal umständlich

Spiele wir doch einmal einen konkreten Berechnungsablauf durch und gehen für den Moment davon aus, wir selbst wären die Datenköchin oder der Datenkoch, wir selbst stellen die Steuerleitungen mittels Schaltern ein und lösen das Taktsignal aus. Es geht darum, zwei beliebige Zahlen miteinander zu multiplizieren. Dazu müssen wir uns genau überlegen, wie sich dies mittels des vorgegebenen Datenpfads realisieren lässt. Er unterstützt ja nur die Addition und Subtraktion als elementare Rechenoperationen, entsprechend müssen wir eine Multiplikation durch das Hintereinanderausführen mehrerer Additionen realisieren. Des Weiteren müssen wir uns um diverse Details kümmern, angefangen vom Einlesen von Multiplikator und Multiplikand bis zur Abspeicherung des Endprodukts.

Eine Möglichkeit eines Ablaufs, das Produkt in unserer Datenküche herzustellen, ist in Abb. 3.15 visualisiert – aber erschrecken Sie nicht: Es sieht komplizierter aus, als es ist. Der Ablauf umfasst sieben Rechenschritte, die in Computermanier von 0 bis 6 (und nicht von 1 bis 7) durchnummeriert sind; Schritt 0 markiert den Startpunkt. In den Kästchen sind die Rechenschritte grafisch dargestellt, unten sehen Sie die entsprechenden Steuerleitungsbelegungen. Die Pfeile zwischen den Kästchen verweisen auf den jeweils nächsten Rechenschritt.

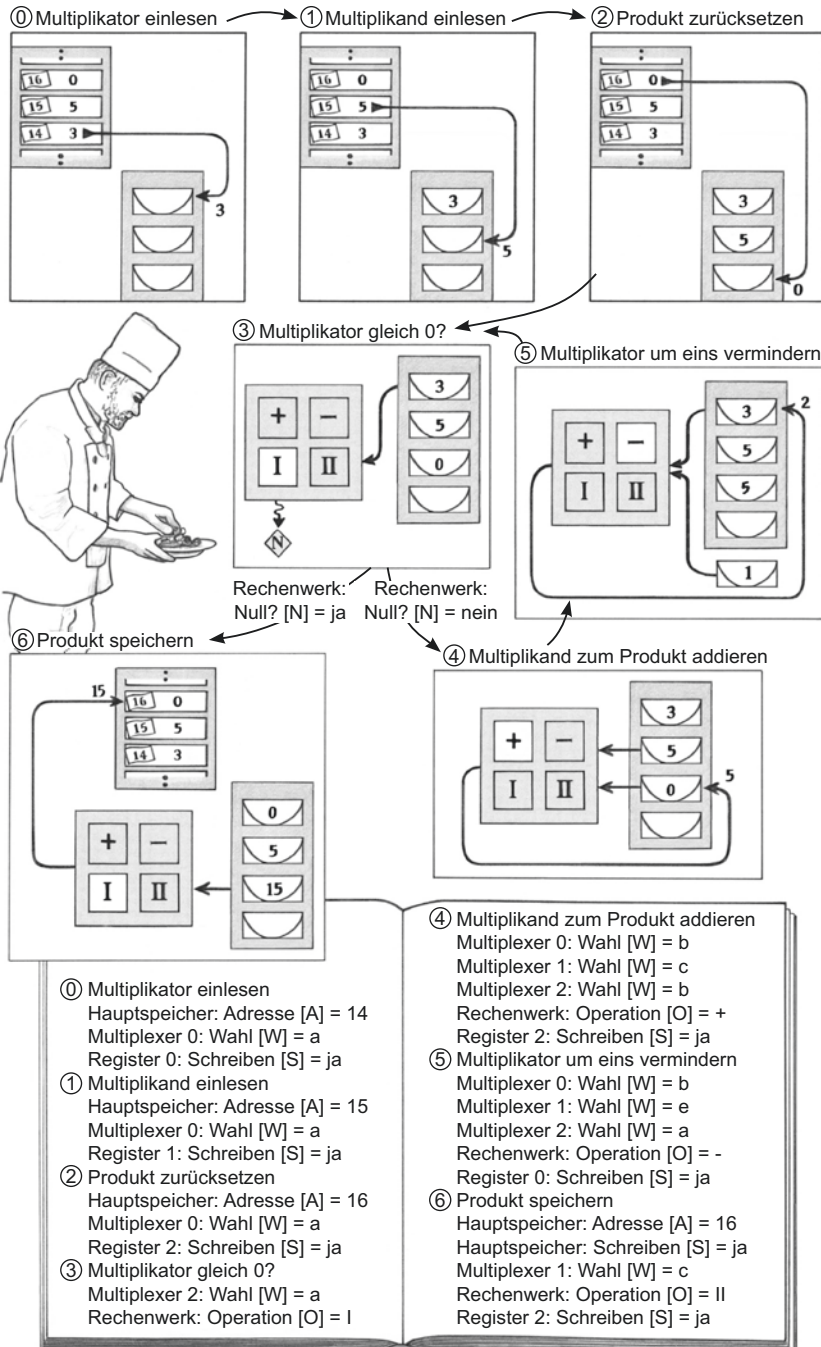


Abb. 3.15 Die einzelnen Schritte zur Durchführung einer Multiplikation im Datenpfad. (© Susanne Staubli, Eckart Zitzler)

Der Kern der Berechnung ist simpel. Schrittweise addieren wir den Multiplikanden zu einer Zwischensumme hinzu, die anfangs 0 ist; dabei zählen wir den Multiplikator bei jeder Addition um eins herunter. Wenn der Multiplikator die Null erreicht, wird die Berechnung gestoppt und die letzte Summe ist das Ergebnis. Hinzu kommt das Übertragen bei der Eingabewerte am Anfang und des Produkts am Ende – aus dem Hauptspeicher in die Register bzw. aus dem Register in den Hauptspeicher. Die Einzelheiten des Ablaufs erklärt Bsp. 3.7.

Beispiel 3.7: Ein Ablauf zum Multiplizieren

Der Ablauf geht davon aus, dass der Multiplikator und der Multiplikand im Hauptspeicher an den Adressen 14 bzw. 15 (die Adressen sind der Lesbarkeit halber dezimal angegeben) abgelegt sind und dass das Ergebnis im Hauptspeicher an Adresse 16 gespeichert werden soll. Weiterhin gehen wir davon aus, dass zu Beginn des Ablaufs in Speicherzelle 16 eine Null steht; das wird benötigt, um die Summe zu Beginn zurückzusetzen.

Die ersten drei Schritte dienen dazu, den Multiplikator in Register 0, den Multiplikanden in Register 1 und die 0 in Register 2 zu kopieren. Letzteres ist erforderlich, weil Register 2 für das Festhalten der Zwischensumme eingesetzt wird.

Die eigentliche Berechnung startet in Schritt 3 mit der Prüfung, ob der Multiplikator gleich null ist. Dies wird erreicht, indem Register 0 an den ersten Eingang des Rechenwerks durchgeschaltet und als Rechenoperation 1 gewählt wird. Liegt auf der Statusleitung des Rechenwerks eine 1, heißt das, Register 0 enthält die Null und die Berechnung ist vollzogen; in Register 2 liegt nun das Endprodukt, das in Schritt 6 im Hauptspeicher abgelegt wird. Andernfalls wird der Multiplikand zum in Register 2 gespeicherten Wert hinzuaddiert und der Multiplikator um eins vermindert. Anschließend beginnt der Kreislauf von Neuem.

Zwei Dinge sind bzgl. Abb. 3.15 noch anzumerken. Bei den Steuerleitungsbelegungen ist immer nur ein Ausschnitt angezeigt; alle nicht aufgeführten „Schreiben“-Leitungen müssen auf „nein“ (0) gesetzt, die restlichen Steuerleitungen können beliebig gesetzt werden. Des Weiteren ist das Taktsignal, das zwischen den Rechenschritten erfolgt, nicht wiedergegeben.

Etwas umständlich das Ganze? In der Tat. Wie bei der Turing-Maschine erfordern bereits einfache Rechnungen aufwändige Abläufe. Doch wir dürfen nicht vergessen, dass wir uns hier auf der untersten Ebene des Computers bewegen. Viele weitere Ebenen sind notwendig, damit wir komfortabel „3 mal 5“ in ein Mikrofon sprechen können und der Computer „15“ auf dem Bildschirm ausspuckt. Wie auch immer: Glücklicherweise ist es ja nicht wirklich so, dass wir den Ablauf von Hand steuern müssen. Es gibt dafür eine Schaltung, die wir uns jetzt anschauen werden.

Der elektrische Datenkoch

Der Kontrollpfad, der den soeben entwickelten Berechnungsablauf automatisieren soll, muss genau das können, was wir auch als Datenköche bewerkstelligen müssten: 1. sich die Nummer des Rechenschritts merken, 2. in Abhängigkeit des Rechenschritts die Steuerleitungen setzen und 3. aus der aktuellen Schrittnummer und dem Zustand der Rechenwerk-Statusleitung die Nummer des nachfolgenden Rechenschritts ermitteln. Für alle drei Aufgaben gibt es eigene Schaltungskomponenten. Zusammengesetzt sieht das Steuerwerk dann so aus, wie es in Abb. 3.16 skizziert ist.

In der Mitte befindet sich ein Register, das sogenannte Schrittregister. Es speichert die Nummer des aktuellen Rechenschritts und wird am Anfang, beim Einschalten des Computers, auf 0 gesetzt, den ersten Rechenschritt. Auf Basis der Nummer des Rechenschritts berechnet ein Baustein „Steuerfunktion“ die entsprechende Belegung aller Steuerleitungen.

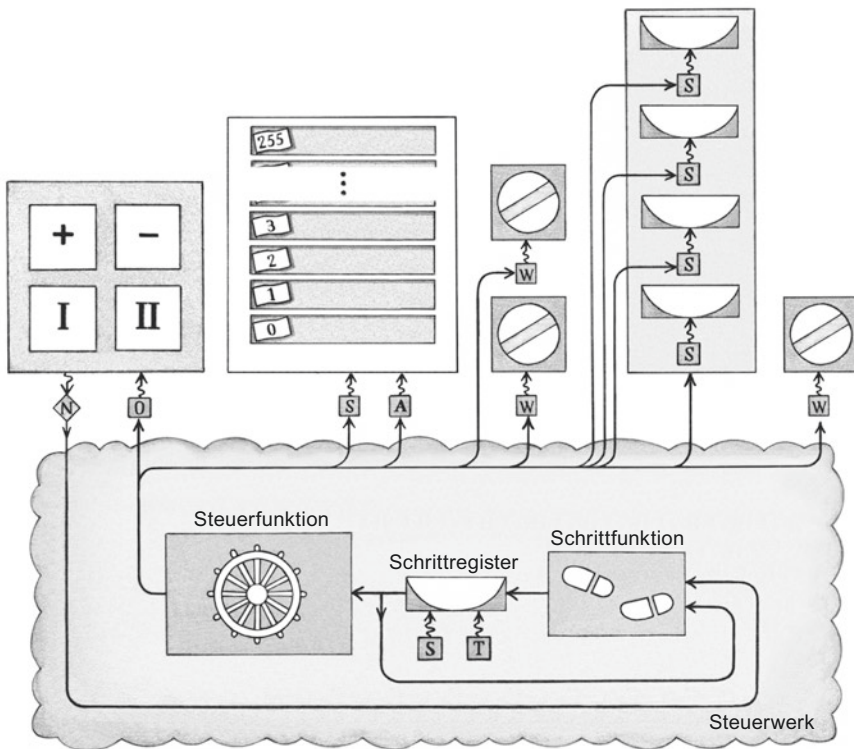


Abb. 3.16 Struktur des Steuerwerks. (© Susanne Staubli, Eckart Zitzler)

Die Ausgabeleitungen dieses Bausteins münden also direkt in den Steuerleitungen des Datenpfads, führen somit in den Datenpfad hinein. Des Weiteren gibt es einen Baustein „Schrittfunktion“; die darin verborgene Schaltung berechnet auf der Basis der Nummer des aktuellen Rechenschritts und dem Zustand der „Null“-Statusleitung die Nummer des als Nächstes auszuführenden Rechenschritts. Dieser Baustein horcht also in den Datenpfad hinein, die Ausgabe wiederum wird an den Eingang des Schrittregisters gelegt.

Im Steuerwerk sind demnach zwei Funktionalitäten getrennt: Die Steuerfunktion zieht die Fäden und kontrolliert den Datenfluss, d. h. Transport und Verarbeitung der Zahlen; die Schrittfunktion bestimmt den Kontrollfluss, die Abfolge der Rechenschritte. Das macht das Ganze handhabbar. Doch wie implementieren wir diese beiden Funktionen als Schaltungen?

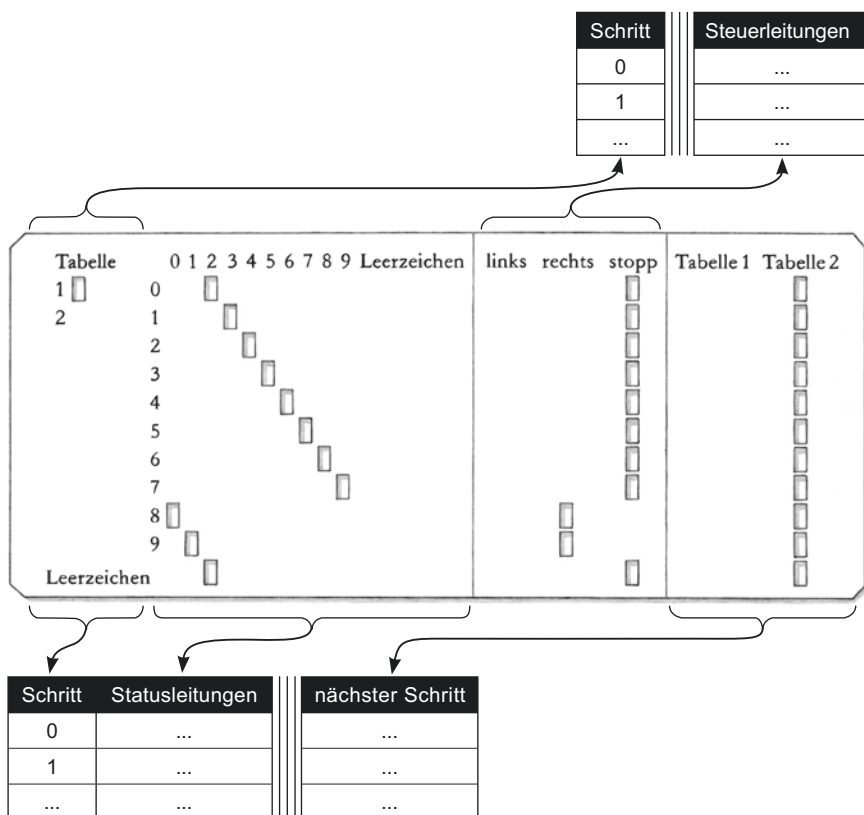


Abb. 3.17 Die Wahrheitstabellen für die Steuer- und Schrittfunktion im Vergleich zu einer Turing-Maschinen-Tabelle. (© Susanne Staubli, Eckart Zitzler)

Dazu müssen wir die Darstellung in Abb. 3.15 leicht anpassen und in die Form zweier Wahrheitstabellen überführen. Für die Steuerfunktion erstellen wir eine Wahrheitstabelle, die für jeden Rechenschritt angibt, wie die einzelnen Steuerleitungen zu belegen sind. Analog konstruieren wir für die Schrittfunktion eine Wahrheitstabelle, die für jeden Rechenschritt und jeden Zustand der „Null?“-Statusleitung die Nummer des nachfolgenden Rechenschritts aufführt. Abb. 3.17 illustriert die Idee im Vergleich zu einer Turing-Maschinen-Tabelle. Aus diesen Wahrheitstabellen können wir schließlich zwei Schaltungen konstruieren, wie wir das ausführlich in Kap. 2 diskutiert haben.

Damit wären wir fast am Ziel, doch eine Frage stellt sich noch: Wer treibt das Ganze an, wer dirigiert das Wechselspiel zwischen Rechen- und Speicheroperationen?

Im Gleichschritt – Marsch!

Es ist das Taktsignal, von dem schon die Rede war. Es kommt auch im Steuerwerk zum Einsatz und bewirkt dort, dass der nächste Rechenschritt eingeläutet wird. Für das Schrittregister ist das Schreibsignal nämlich permanent gesetzt. Dies hat zur Folge, dass die Nummer des aktuellen Rechenschritts beim Eintreffen des Taktsignals durch die Nummer des nachfolgenden Rechenschritts ersetzt wird. Wird die neue Schrittnummer am Ausgang des Schrittregisters sichtbar, ändern sich auch über die Steuerfunktion die Steuerleitungsbelegungen und der nächste Rechenschritt ist in vollem Gang.

Wo das Taktsignal herkommt? Es wird von einem dedizierten Baustein im Prozessor generiert, dem Taktgeber, der die gleiche Funktion wie ein Metronom oder ein Schlagmann in einem Ruderboot hat (Abb. 3.18): gleichförmig den Takt vorzugeben. Dazu verändert er periodisch den Zustand der „Takt“-Leitung, 0 und 1 alternierend. Ein Takt beginnt, wenn die „Takt“-Leitung von 0 auf 1 wechselt; er endet, wenn die „Takt“-Leitung das nächste Mal von 0 auf 1 wechselt. Dieser Takt ist derselbe für alle Speicherbausteine im Prozessor, das Taktsignal ist quasi der Herzschlag der Zentraleinheit.

Mit diesem Rhythmus treibt der Taktgeber den Berechnungsablauf an und prägt ihm ein zeitliches Raster auf. Zu Taktbeginn wechseln selektierte Speicherelemente ihre Ausgangswerte, die sich daraufhin während des Takts entlang des Datenpfads ausbreiten, bis sie in verarbeiteter Form wieder am Eingang eines Speicherbausteins anliegen. Letztlich gibt



Abb. 3.18 Metronom und Schlagfrau geben den Takt an. (© Gudellaphoto/stock.adobe.com (links), Christian Schwier/stock.adobe.com (rechts))

es verschiedene Ausbreitungswege im Prozessor, und zwar immer von Speicherelement zu Speicherelement. In unserem Beispielprozessor gibt es vier dieser Wege, die in Abb. 3.19 dargestellt sind; teilweise können sie parallel benutzt werden. Die Taktdauer muss folglich so gewählt werden, dass die Daten, egal welchen Weg sie nehmen, genügend Zeit haben, die vollständige Strecke zu absolvieren; letztere ist im Voraus aus den maximalen Verzögerungszeiten der Schaltungen zu berechnen. Was das Ganze mit der Prozessorgeschwindigkeit zu tun hat, erläutert Anm. 3.1.

Anmerkung 3.1: Prozessoren haben Hertz

Je kürzer die Taktdauer, desto mehr Rechenschritte kann ein Prozessor in einer Sekunde ausführen und desto schneller ist eine Berechnung abgeschlossen. Nachvollziehbarerweise versucht man, die Taktdauer zu minimieren. Hersteller werben jedoch mit einer anderen Kennzahl, der Taktfrequenz. Die Taktfrequenz steht für die Anzahl der Takte pro Sekunde und wird in der Einheit Hertz, abgekürzt als Hz, angegeben. Bei den ersten Mikroprozessoren bewegte sich die Taktfrequenz noch im Bereich von einigen hundert kHz – sprich: Kilohertz, also tausend Takte pro Sekunde –, 20 Jahre später waren es dann GHz – sprich: Gigahertz, eine Milliarde Takte pro Sekunde.

Die Taktfrequenz ist ein wichtiger Indikator für die Geschwindigkeit eines Prozessors. Doch generell hängt die Leistungsfähigkeit eines Prozessors und insbesondere eines Computersystems von vielen weiteren Faktoren ab: Wie mächtig sind die Rechenoperationen, die das Rechenwerk unterstützt, wie viele Register bietet der Datenpfad usw.? Auf einem Prozessor mit größerer Taktdauer lässt sich eine Berechnung möglicherweise in wesentlich weniger Rechenschritten realisieren als auf einem anderen mit einer kleineren Taktdauer. Ein Computer muss folglich nicht zwingend schneller sein als ein Konkurrenzprodukt, nur weil er eine höhere Prozessortaktfrequenz aufweist.

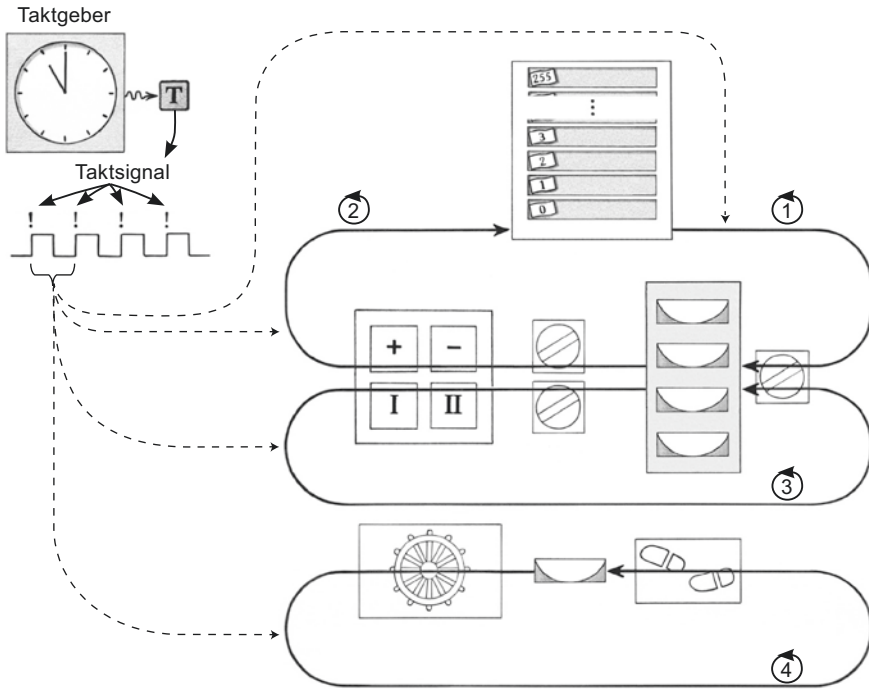


Abb. 3.19 Der Taktgeber als treibende Kraft im Prozessor. (© Susanne Staubli, Eckart Zitzler)

Jetzt können wir den ersten Zwischenerfolg feiern! Wir haben eine Schaltung entworfen, bestehend aus Datenpfad, Kontrollpfad und Taktgeber, die eine bestimmte Berechnung automatisiert durchführen kann: zwei Zahlen miteinander multiplizieren. Nach diesem Schema können wir für beliebige Berechnungen eine elektronische Rechenmaschine konstruieren. Allerdings erfordern unterschiedliche Rechenaufgaben auch unterschiedliche Konstruktionen, d. h., für jede spezifische Berechnung ist eine eigene Schaltung zu implementieren. Wir müssen unseren Beispiel-Prozessor noch dahingehend erweitern, dass der Ablauf der Rechenschritte flexibel eingestellt werden kann.

Die Software betritt die Bühne

Bislang hatten wir nur eine Stufe der Flexibilität, dass nämlich beliebige Daten eingespeist und nach derselben Prozedur verarbeitet werden können. Diese Flexibilität haben wir erreicht, indem die Zahlen nicht fix in der

Schaltung eingebaut waren, sondern aus dem Hauptspeicher eingelesen wurden. Warum nicht dieses Konzept auf den Berechnungsablauf übertragen? Das hätte zur Konsequenz, dass der Kontrollpfad nicht starr konstruiert ist, sondern einstellbare Anteile besitzen muss. Nur so lassen sich die einzelnen Rechenschritte und ihre Abfolge nach Belieben festlegen.

Die Idee ist bestechend einfach: Anstatt die Tabellen für die Steuerleitungsbelegungen und die nächsten Schritte, also Steuerfunktion und Schrittfunktion, in Form von Bausteinen zu realisieren, werden sie veränderbar im Hauptspeicher abgelegt. Für das Steuerwerk heißt das, dass diese Informationen nicht per Schaltung „berechnet“, sondern direkt aus dem Hauptspeicher abgegriffen und anschließend wie gehabt umgesetzt werden – so wird aus Hardware, also unveränderlichen, „harten“ Komponenten, Software, also veränderliche, „weiche“ Einstellungen. Schauen wir uns das im Detail an und beginnen bei der Steuerfunktion.

Befehle erfinden

Wollten wir die Tabelle mit den Steuerleitungsbelegungen eins zu eins im Hauptspeicher ablegen, müssten wir pro Rechenschritt und pro Leitung einen entsprechenden Wert – 0 oder 1 – speichern. In unserem Beispiel könnten wir acht Werte pro Speicherzelle spezifizieren, bräuchten also mehrere Speicherzellen, um einen einzigen Rechenschritt zu definieren. Das ist ungünstig, denn Zugriffe auf den Hauptspeicher sind aufwändig und verlangsamen die Berechnung. Bei realen Mikroprozessoren, die wesentlich mehr Steuerleitungen besitzen, würde sich das Problem noch verschärfen.

Doch eigentlich brauchen wir gar nicht so viel Flexibilität, wir müssen nicht bei jedem Schritt jede einzelne Steuerleitung individuell einstellen können. Zum einen sind immer nur ein paar der Leitungen von Interesse. Wenn beispielsweise in einem Rechenschritt der Hauptspeicher nicht benutzt wird, können die Adressleitungen beliebig gesetzt werden. Zum anderen sind viele Kombinationen von Steuersignalen unsinnig, z. B. wenn kein einziges Schreibsignal enthalten ist. Aus diesem Grund wählt man aus allen möglichen Steuerleitungsbelegungen ein paar sinnvolle Muster aus, die die verfügbaren Rechenoperationen repräsentieren. Diese Rechenoperationen – quasi die Befehle, die der Prozessor versteht – werden so codiert, dass ihre Speicherung möglichst wenig Platz in Anspruch nimmt.

Lassen Sie uns die Idee an einem Beispiel konkretisieren. Ein Befehl, den unser Prozessor unterstützen wird, lautet: „Vermindere den Inhalt eines

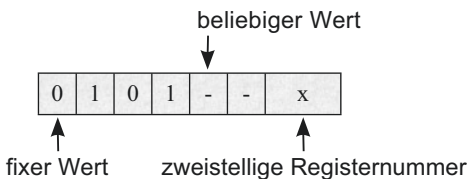
Registers um eins.“ Wir kürzen ihn mit der etwas kryptisch anmutenden Bezeichnung `dec Rx` ab, wobei `dec` für das englische Verb *decrement* (vermindern), `R` für Register und `x` für die Nummer des betroffenen Registers steht. Solche Kürzel verwenden Informatiker, um sich Befehle besser merken und sie übersichtlicher darstellen zu können. Entscheidend ist jedoch der Code, der im Hauptspeicher hinterlegt wird. Im Folgenden verwenden wir eine Codierung, die jedem Befehl eine achtstellige Dualzahl zuordnet und nur eine Speicherzelle pro Rechenschritt erfordert (Bsp. 3.8).

Beispiel 3.8: Wie ein Befehl codiert wird

Das Schema in Abb. 3.20 oben illustriert, wie eine Steuerleitungsbelegung, die den Inhalt des Registers 0 um eins vermindert, über einen Code verkürzt dargestellt werden kann. Unten in der Abbildung sehen Sie das Format, nach dem ähnliche Operationen als achtstellige Dualzahlen codiert werden. Die am weitesten links stehenden Ziffern sind fixiert und kennzeichnen eindeutig, dass es sich um den `dec`-Befehl handelt; die zwei rechten Ziffern bezeichnen die Nummer des Registers, auf das sich die Operation bezieht; die restlichen zwei Ziffern können beliebig gewählt werden. Folglich steht das Kürzel `dec Rx` eigentlich für eine Gruppe von vier Befehlen, von denen jeder auf mehrere Arten codiert werden kann. Doch so genau wollen wir es hier nicht nehmen, das Wort Befehl kann sowohl für die Gruppe als auch für einen einzelnen Befehl stehen.

Nach dem gleichen Prinzip konstruieren wir drei weitere Rechenbefehle, mit denen wir zwei Registerwerte addieren, zwei Registerwerte subtrahieren und einen Registerwert um eins erhöhen können. Sie sind in Abb. 3.21 beschrieben und werden ebenfalls über achtstellige Dualzahlen codiert. In Bsp. 3.9 können Sie sehen, welchen Bezug diese Befehle zum Multiplikationsablauf haben.

Format:



Beispiele:

0	1	0	1	0	0	0	0	dec R0
0	1	0	1	1	0	0	0	dec R0
0	1	0	1	1	0	0	1	dec R1
0	1	0	1	1	1	1	0	dec R2
0	1	0	1	0	1	1	1	dec R3

Abb. 3.20 Prinzip, wie eine Steuerleitungsbelegung durch einen Code repräsentiert wird, am Beispiel des Befehls `dec Rx`, der den Inhalt eines Registers um eins vermindert. (© Susanne Staubli, Eckart Zitzler)

Befehl	Abkürzung	Codierung
Addiere die in Register Rx abgelegte Zahl zu der in Register Ry abgelegten Zahl und schreibe das Ergebnis in Register Rx.	add Rx,Ry	1 0 1 0 y x
Subtrahiere die in Register Ry abgelegte Zahl von der in Register Rx abgelegten Zahl und schreibe das Ergebnis in Register Rx.	sub Rx,Ry	1 0 1 1 y x
Erhöhe den Wert in Register Rx um eins.	inc Rx	0 1 0 0 - - x
Erniedrige den Wert in Register Rx um eins.	dec Rx	0 1 0 1 - - x

Abb. 3.21 Rechenbefehle und ihre Codierung. (© Susanne Staubli, Eckart Zitzler)

Beispiel 3.9: Wie Rechenbefehle benutzt werden

Werfen wir noch einmal einen Blick auf den Multiplikationsablauf in Abb. 3.15. In Schritt 4 wird der Inhalt von Register 2 zum Inhalt von Register 1 addiert und die Summe wiederum im Register 2 gespeichert; der entsprechende Befehl lautet `add R2,R1` und hat den Code 10100110. Schritt 5 hingegen entspricht dem Befehl `dec R0`, der als 01010000 codiert werden kann.

Damit Register auch mit Zahlen gefüttert und Ergebnisse im Hauptspeicher gesichert werden können, führen wir noch drei Speicherbefehle ein (siehe Abb. 3.22). Der erste erlaubt das Einlesen einer vierstelligen Dualzahl in ein bestimmtes Register. Zwei weitere ermöglichen den Datenaustausch zwischen dem Hauptspeicher und den Registern. Dabei ergibt sich das Problem, dass wir ja noch die Adresse der Speicherzelle angeben müssen; sie lässt sich zusammen mit der Befehlskodierung nicht in einer achtstelligen Dualzahl unterbringen. Der Trick ist, dass wir einfach davon ausgehen, dass die Adresse in einem anderen Register spezifiziert ist (vgl. Bsp. 3.10).

Beispiel 3.10: Wie Speicherbefehle benutzt werden

In Schritt 0 des Multiplikationsbeispiels (Abb. 3.15) wird der Inhalt von Speicherzelle 14 in das Register 0 kopiert. Für diesen Vorgang benötigen wir zwei Befehle: `load R3,14` lädt die Adresse der gewünschten Speicherzelle in das Register 3, das wir hier zur Hilfe nehmen; `load R0,(R3)` überträgt schließlich den Inhalt von Speicherzelle 14 in Register 0. Die Codierung der beiden Befehle lautet 11111011 bzw. 10001100.

Befehl	Abkürzung	Codierung						
Schreibe die Zahl z in Register R_x .	load $R_x, \#z$	<table><tr><td>1</td><td>1</td><td colspan="2">z</td><td>x</td></tr></table>	1	1	z		x	
1	1	z		x				
Schreibe die Zahl, die im Hauptspeicher an der in Register R_y spezifizierten Adresse abgelegt ist, in Register R_x .	load $R_x, (R_y)$	<table><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>y</td><td>x</td></tr></table>	1	0	0	0	y	x
1	0	0	0	y	x			
Schreibe die in Register R_x abgelegte Zahl in den Hauptspeicher an der in Register R_y spezifizierten Adresse.	store $R_x, (R_y)$	<table><tr><td>1</td><td>0</td><td>0</td><td>1</td><td>y</td><td>x</td></tr></table>	1	0	0	1	y	x
1	0	0	1	y	x			

Abb. 3.22 Speicherbefehle und ihre Codierung. (© Susanne Staubli, Eckart Zitzler)

Befehl	Abkürzung	Codierung
Tue nichts.	nop	0 0 - - - - 0 -

Abb. 3.23 Ein Befehl, der nichts tut: der nop-Befehl und seine Codierung. (© Susanne Staubli, Eckart Zitzler)

Zuletzt kommen wir noch zu einem Befehl, der eigentlich keiner ist: dem nop-Befehl (siehe Abb. 3.23). Das Kürzel steht für den englischen Ausdruck *no operation* und bezeichnet eine Operation, die nichts bewirkt und auch nichts verändert. So unsinnig das klingt, benötigt man doch gelegentlich diesen Befehl. Beispielsweise um eine Warteschleife zu realisieren, in der nichts passiert, wie wir später noch sehen werden.

Mit diesem Satz an Befehlen – man spricht auch von einem Instruktionssatz, der aus verschiedenen Instruktionen besteht – haben wir eine Möglichkeit geschaffen, Steuerleitungsbelegungen kompakt zu beschreiben. Doch wie wird nun die gesamte Tabelle für die Steuerfunktion im Hauptspeicher gespeichert? Ganz einfach, für jeden Rechenschritt legen wir die codierte Steuerleitungsbelegung an derjenigen Speicheradresse ab, die der Nummer des Rechenschritts entspricht. Zu Rechenschritt 0 finden wir die entsprechende Steuerleitungsbelegung in Speicherzelle 0, Rechenschritt 1 ist in Speicherzelle 1 spezifiziert usw. Oder andersherum formuliert: Wir nutzen die Speicheradressen, um die Rechenschritte eindeutig zu identifizieren.

Die Kontrolle haben

Die einzelnen Schritte einer Berechnung können wir nun flexibel einstellen, doch wie spezifizieren wir den Ablauf, die Verknüpfungen zwischen diesen Schritten? Auch hier vereinfacht ein Kniff die Handhabung. Wir gehen

kurzerhand davon aus, dass auf einen Rechenschritt normalerweise der Rechenschritt mit der nächsthöheren Nummer folgt, unabhängig von dem Zustand der Statusleitungen. Die Befehle werden im Standardfall also in der Reihenfolge ausgeführt, wie sie im Hauptspeicher abgelegt sind. Das ist eine Konvention, nichts anderes. Damit wir diese Konvention auch durchbrechen können, führen wir neue Befehle ein (siehe Abb. 3.24).

Mit dem `bneqz`-Befehl können wir eine Verzweigung einbauen; welchen Weg der Kontrollfluss nimmt, hängt vom Inhalt eines ausgewählten Registers ab. Ist der Registerinhalt gleich null, so wird die Standardreihenfolge eingehalten: Der nächste Befehl ist derjenige in der nächsthöheren Speicheradresse. Ist der Registerinhalt ungleich null, so wird ein Befehl übersprungen: Die Berechnung wird beim Befehl in der um zwei höheren Speicheradresse fortgesetzt. Der `bneqz`-Befehl – die Abkürzung steht für den englischen Ausdruck *branch if not equal zero* – verändert den Kontrollfluss in Abhängigkeit der Statusleitung und wird deswegen auch „bedingter Sprungbefehl“ genannt.

Der `jump`-Befehl hingegen ist an keine Bedingungen geknüpft. Er lenkt den Kontrollfluss um an eine beliebige Speicheradresse im Umfeld des aktuellen Befehls. Die Speicheradresse des als Nächstes auszuführenden Befehls lässt sich dabei auf zwei Arten berechnen. In der einen Variante zählen wir eine bestimmte Anzahl zur nächsten Speicheradresse hinzu, in der anderen Variante ziehen wir diese ab. Mit anderen Worten: Der `jump`-Befehl springt in der Befehlsreihenfolge um eine gewisse Schrittzahl vorwärts oder rückwärts.

Wem das alles noch etwas spanisch vorkommt, dem wird vielleicht Bsp. 3.11 helfen; dort werden die Befehle konkret eingesetzt.

Befehl	Abkürzung	Codierung
Überspringe den nachfolgenden Befehl genau dann, wenn Rx nicht die Zahl 0 enthält.	<code>bneqz Rx</code>	0 1 1 - - - x
Springe ausgehend vom nachfolgenden Befehl z Befehle vorwärts.	<code>jump #z</code>	0 0 z 1 0
Springe ausgehend vom nachfolgenden Befehl z Befehle rückwärts.	<code>jump #-z</code>	0 0 z 1 1

Abb. 3.24 Sprungbefehle zur Änderung des Kontrollflusses. (© Susanne Staubli, Eckart Zitzler)

Beispiel 3.11: Mit Sprungbefehlen den Kontrollfluss ändern

Die Verzweigung in Schritt 3 des Multiplikationsbeispiels (Abb. 3.15) lässt sich mittels der Sprungbefehle folgendermaßen realisieren. Mit dem Befehl `bneqz R3` fragen wir zunächst ab, ob der in Register R3 gespeicherte Multiplikator der Null entspricht. Wenn ja, springen wir mit einem anschließenden `jump`-Befehl mehrere Schritte vorwärts an das Ende der Berechnung. Wenn nein, wird dieser `jump`-Befehl ausgelassen und die Berechnung an der übernächsten Speicheradresse fortgesetzt.

Mit den Sprungbefehlen haben wir nun eine elegante Möglichkeit gefunden, den Ablauf festzulegen. Wir müssen die Tabelle für die Schrittfunktion demnach nicht mehr gesondert im Hauptspeicher ablegen, sie ergibt sich aus der Anordnung der Befehle im Hauptspeicher.

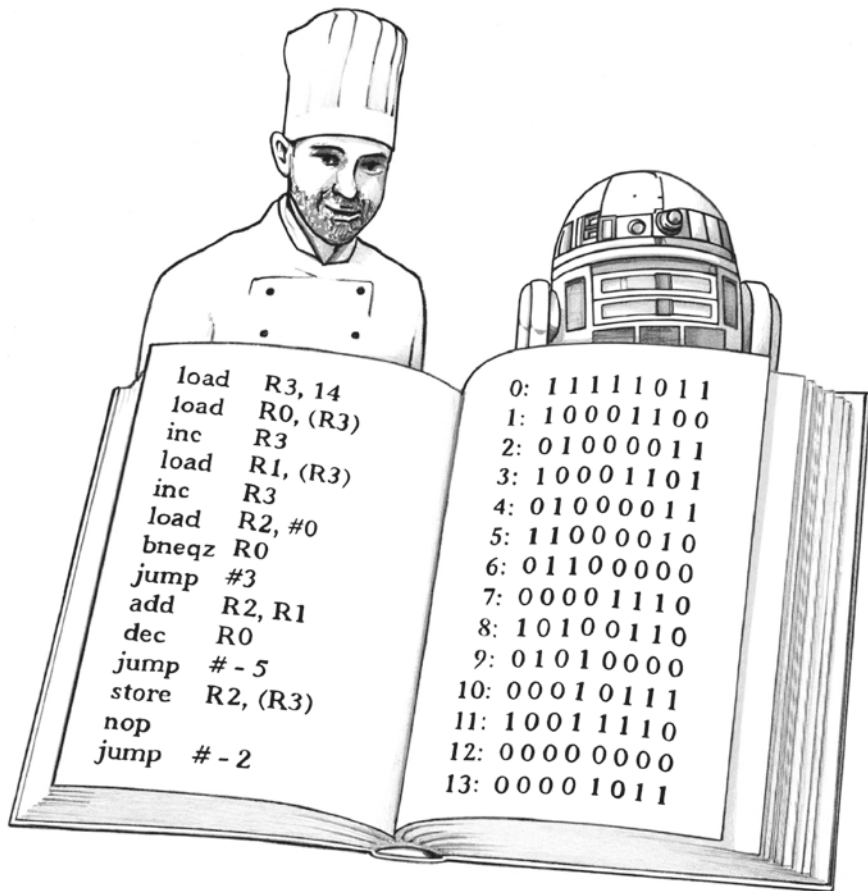
Maschinensprech

Jetzt sind wir endlich so weit: Wir können von einem Programm sprechen. Es handelt sich dabei um eine Sequenz von Befehlen, die der Prozessor gemäß der vorgegebenen Reihenfolge und gemäß eingeflochtener Sprunganweisungen abarbeitet. Beispielsweise können wir den Multiplikationsablauf (Abb. 3.15) als Programm formulieren. Die in Abb. 3.25 gezeigte Umsetzung – rechts die Befehlscodes zusammen mit den vorangestellten Speicheradressen, links die Kürzel – sieht etwas anders aus als der Originalablauf. Das liegt daran, dass die zur Verfügung stehenden Befehle die Möglichkeiten einschränken. Das macht aber nichts, wir müssen halt etwas improvisieren (siehe Bsp. 3.12).

Beispiel 3.12: Ein Programm zur Multiplikation

Die ersten vier Befehle des Programms aus Abb. 3.25 dienen dazu, Multiplikator und Multiplikand in Register 0 bzw. 1 einzulesen; das Register 3 fungiert dabei als Container für die Speicheradresse. Der nachfolgende Befehl bewirkt, dass Register 3 die Speicheradresse enthält, an der zum Schluss das Produkt abgelegt werden soll. Nachdem Register 2, das die Zwischensumme enthält, auf null zurückgesetzt wurde, kommen wir zu der bereits diskutierten Abfrage, ob der Multiplikator gleich null ist. Falls nein, wird der Multiplikand zur Zwischensumme addiert und der Multiplikator um eins vermindert; ein Sprungbefehl verzweigt anschließend wieder zur Abfrage. Am Ende der Berechnung wird das Ergebnis im Hauptspeicher abgelegt und danach eine sogenannte Endlosschleife ausgeführt.

Eine Anmerkung: Irritierend ist an dieser Stelle vielleicht, dass wir die Befehlssequenz von oben nach unten aufschreiben, so wie wir das gewohnt sind und wie es üblich ist. Im Hauptspeicher hingegen wird sie von unten nach oben abgelegt, also von den niedrigeren hin zu den höheren Speicheradressen.



Assemblersprache

Maschinensprache

Abb. 3.25 Der Multiplikationsablauf als Programm formuliert. (© Susanne Staubli, Eckart Zitzler)

Mit dem Entscheid, achtstellige Dualzahlen zur Kennzeichnung standardisierter Steuerleitungsbelegungen zu verwenden, haben wir einen Code geschaffen, einen Maschinencode. Dieser Maschinencode ist spezifisch für unseren Prozessor, gleichzeitig abstrahiert er von seinem Innenleben und kapselt unnötige Details ab. Häufig spricht man in diesem Zusammenhang auch von einer Maschinensprache, einer Sprache, die alle möglichen Befehle als Dualzahlenwörter enthält. Das ist vielleicht etwas hoch gegriffen, schließlich kommunizieren hier nicht zwei Gesprächspartner miteinander, sondern die Programmiererin instruiert einseitig den Computer. Letztlich geht es um einen Formalismus, über den wir Einstellungen am Prozessor

beschreiben können – Einstellungen, die den Berechnungsablauf definieren. Die Maschinensprache markiert daher den Übergang von der Welt der Hardware zur Welt der Software.

Doch machen wir uns nichts vor: Maschinencode ist für uns Menschen denkbar ungeeignet. Keiner kann oder will sich all die Zahlenformate merken, mit denen sich Befehle ausdrücken lassen. Die Kürzel sind da schon angenehmer, und deswegen entwickelte man Programme, die Texte mit Kürzelfolgen direkt in Maschinencode übersetzen können. Diese Programme nennt man Assembler und die Ebene der Kürzel Assemblersprache. Assemblersprachen sind ein erster Schritt, die Programmierung für den Menschen einfacher zu gestalten; doch auch sie werden nur selten benutzt. Stattdessen verwendet man Ausdrucksformen, mit denen sich komplexe Abläufe übersichtlich und kompakt darstellen lassen. Klar, auch hier braucht es Programme, die die abstrakten Formulierungen in Maschinencode übersetzen können.

Programme ausführen

So weit zur Theorie: Wir können nun eine Berechnung in Form eines Programms flexibel im Hauptspeicher spezifizieren. Was heißt das jetzt für Datenpfad und Kontrollpfad, wenn ein Programm ausgeführt werden soll? Was ändert sich?

Neu ist, dass um die eigentliche Berechnung herum quasi eine weitere Berechnung notwendig wird. Für jeden Rechenschritt muss der Prozessor 1. den Befehl aus dem Hauptspeicher lesen, 2. den Befehlscode entschlüsseln, 3. den Befehl in Steuerleitungsbelegungen umsetzen und ausführen sowie 4. die Nummer des nächsten Befehls ermitteln. Das, was vorher in einem Schritt möglich war, erfordert nun mehrere Arbeitsschritte. Die Flexibilität geht – das ist der Preis – einher mit zusätzlichem Verwaltungsaufwand.

Als Konsequenz muss der Datenpfad zwei Arten von Verarbeitungen unterstützen. Einerseits gilt es die Programmdaten aus dem Hauptspeicher – das Programm – zu verarbeiten und andererseits die durch das Programm definierte Berechnung auszuführen und dabei Zahlen bzw. Daten aus dem Hauptspeicher zu verarbeiten. Damit beide Vorgänge nicht miteinander kollidieren und sich ins Gehege kommen, führen wir zwei neue Register ein, die einzig der Verarbeitung der Programmdaten dienen. Eines speichert die Speicheradresse des aktuellen Befehls, ein anderes speichert den Befehl selbst. Bsp. 3.13 erläutert, wie die Erweiterungen des Datenpfads im Detail aussehen.

Beispiel 3.13: Den Datenpfad erweitern

Der erweiterte Datenpfad, in Abb. 3.26 zu sehen, enthält also zwei zusätzliche Register. Der Programmzähler (Program Counter im Englischen oder kurz PC) merkt sich, wo im Programm wir uns aktuell befinden, und kann sowohl mit dem Hauptspeicher als auch mit dem Rechenwerk verbunden werden. Das Befehlsregister ist ein Zwischenspeicher für den aktuellen Befehl, sodass der Befehlscode interpretiert und in Steuerleitungsbelegungen übersetzt werden kann.

Neben den zwei Registern gibt es einen zusätzlichen Multiplexer, in Abb. 3.26 mit „Multiplexer 3“ angeschrieben. Er dient dazu, dass sowohl der Programmzähler als auch die Register 0 bis 3 die Hauptspeicheradresse definieren können. Ersteres brauchen wir, um den nächsten Befehl aus dem Hauptspeicher zu holen, Letzteres wird für die `load`- und `store`-Befehle benötigt, denn dort wird ja auf eine Adresse in einem Register Bezug genommen.

Schließlich gibt es noch einen neuen Baustein, den Decoder. Er extrahiert die verschiedenen Informationen aus der Dualzahl, die den aktuellen Befehl codiert: um welchen Befehl es sich handelt („Befehlstyp“-Leitungen, abgekürzt mit „B“), die Registernummern x und y sowie die Konstante z bei `load`- und `jump`-Befehlen. Die „ z “-Leitungen führen über einen Multiplexer zum Rechenwerk, die anderen Ausgänge des Decoders werden an das Steuerwerk weitergeleitet und fungieren als Statusleitungen.

Am Aufbau des Kontrollpfads ändert sich nichts, abgesehen von zusätzlichen Steuerleitungen für die neuen Bausteine und weiteren Statusleitungen vom Decoder. Was gänzlich anders aussieht, ist hingegen der Ablauf, den das Steuerwerk implementiert; er ist in Abb. 3.27 visualisiert. Die Reihenfolge ist etwas anders als vorhin besprochen: Erst wird der aktuelle Befehl aus dem Hauptspeicher gelesen, dann wird der Programmzähler um eins erhöht, anschließend wird in Abhängigkeit des Befehls innerhalb des Steuerwerk-Ablaufs verzweigt und der aktuelle Befehl ausgeführt, und zum Schluss geht es wieder von vorne los. Wer es genauer wissen möchte, kann in Bsp. 3.14 nachlesen, was im Einzelnen passiert.

Beispiel 3.14: Einen Befehl abarbeiten

Spiele wir doch den Ablauf in Abb. 3.27 einmal im Detail durch. Nehmen wir an, der Programmzähler enthalte den dezimalen Wert 6, an der Speicheradresse 6 stünde der Befehl `bneqz R0`, das Schrittnummer 0 im Steuerwerk enthalte den Wert 0 und die Taktflanke sei soeben erfolgt.

Das Steuerwerk befindet sich zu diesem Zeitpunkt intern im Arbeitsschritt 0 (nicht zu verwechseln mit den Rechenschritten des Programms, jeder Rechenschritt erfordert mehrere Arbeitsschritte des Steuerwerks). In diesem Arbeitsschritt wird der Programmzähler an die Adressleitungen des Hauptspeichers angelegt und Multiplexer 0 schaltet den Hauptspeicherausgang durch zum Eingang des Befehlsregisters. Bei der nächsten Taktflanke ist dann der Befehl `bneqz R0`, codiert über die Dualzahl 01100000, im Befehlsregister

gespeichert und das Steuerwerk befindet sich im Arbeitsschritt 1. Die Steuerleitungen sind nun so gesetzt, dass der Inhalt des Programmzählers bei der nächsten Taktflanke um eins erhöht ist. Welches der nachfolgende Arbeitsschritt ist, hängt schließlich von den „Befehlstyp“-Statusleitungen des Decoders ab: Da es sich um den Befehl `bneqz R0` handelt, geht es bei Arbeitsschritt 9 weiter: Dort wird Register 0 auf null überprüft. Enthält es eine Null, dann ist nichts weiter zu tun, der Befehl ist abgearbeitet und der nächste Arbeitsschritt ist wiederum Arbeitsschritt 0. Ist der Registerinhalt ungleich null, muss der Sprung zum übernächsten Befehl ausgeführt werden, d. h., noch einmal muss eine Eins zum Programmzähler addiert werden, was genau in Arbeitsschritt 8 passiert; erst danach kann mit dem nächsten Befehl in Arbeitsschritt 0 begonnen werden.

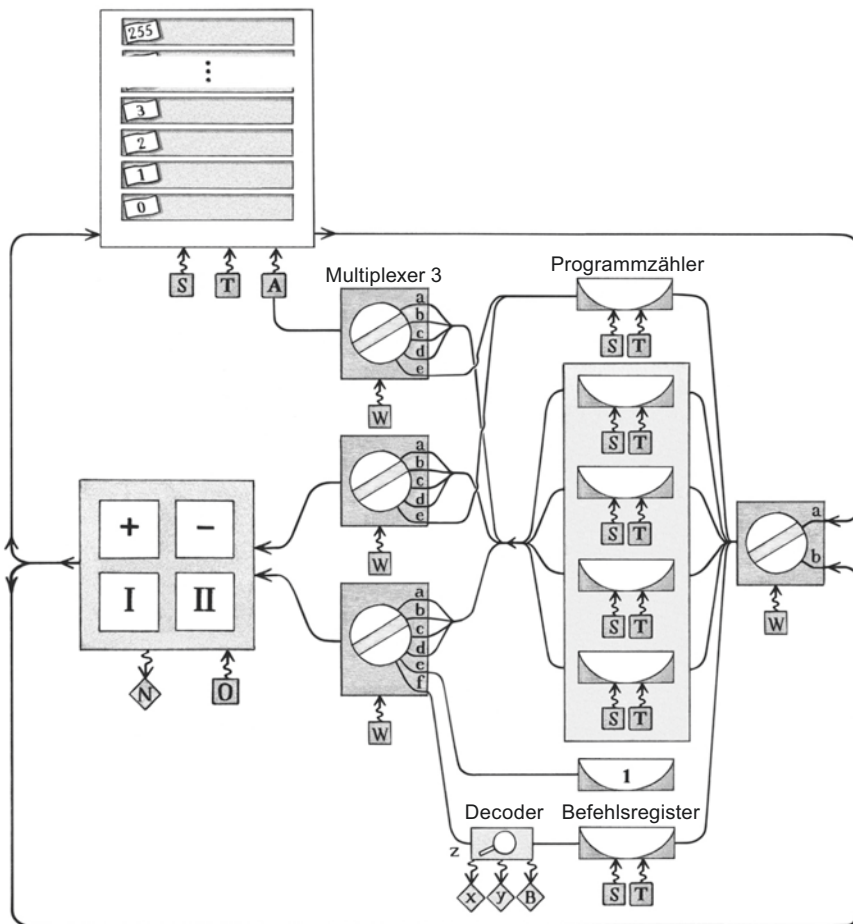


Abb. 3.26 Der erweiterte Datenpfad für den Beispiel-Prozessor. (© Susanne Staubli, Eckart Zitzler)

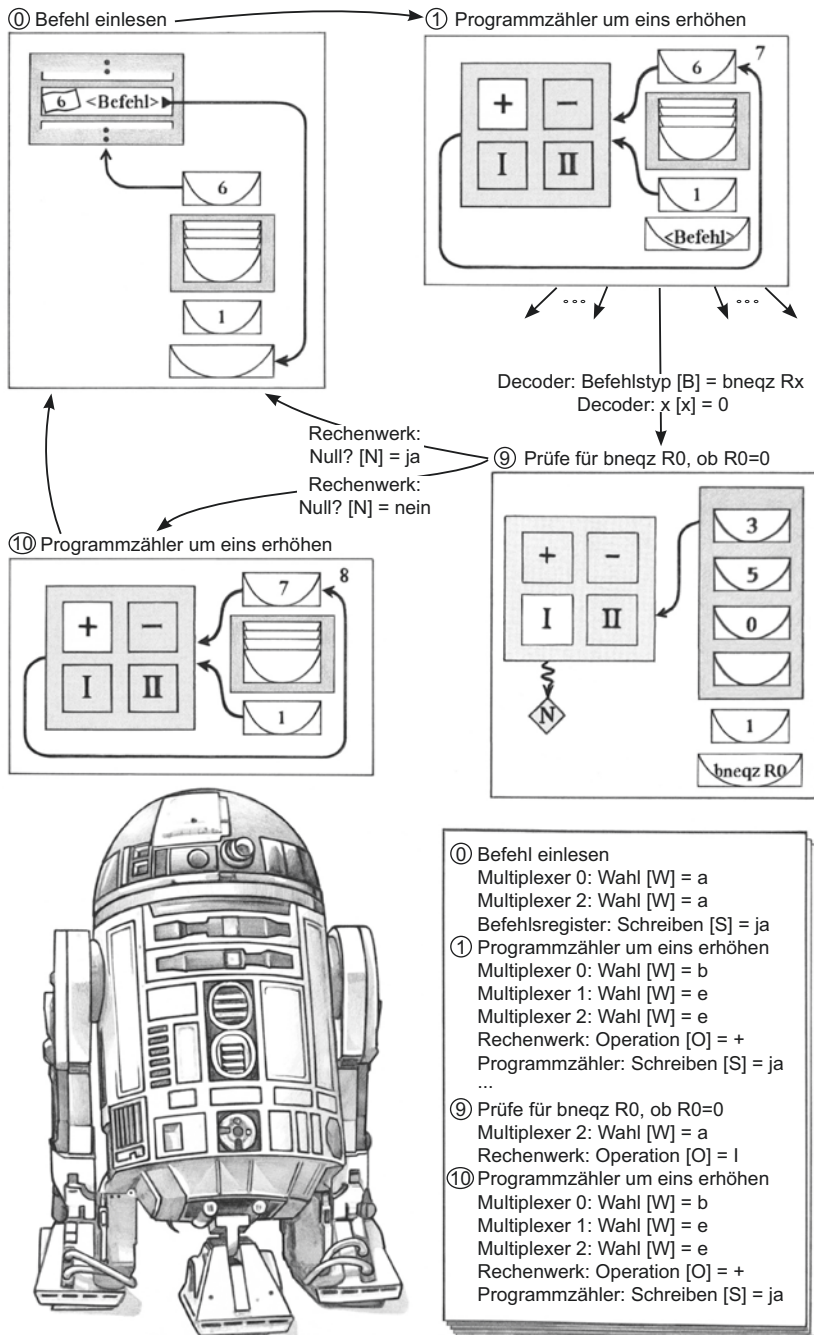


Abb. 3.27 Der Ablauf, den das Steuerwerk im programmierbaren Prozessor implementiert. (© Susanne Staubli, Eckart Zitzler)

Zum Schluss müssen wir den Ablauf in der bekannten Manier erst in entsprechende Wahrheitstabellen übersetzen und dann in Schaltungen überführen. Dabei handelt es sich um mühselige Detailarbeit – mit viel Potenzial für Fehler. Damit Sie einen Eindruck von der konkreten Ausgestaltung bekommen, sind die Tabellen für die Steuerfunktion und die Schrittfunktion in Abb. 3.28 bzw. Abb. 3.29 ausschnittsweise gezeigt. Tatsächlich sind diese wesentlich größer und umfassen mehrere Dutzend Einträge.

Damit sind wir am Ziel angekommen, unser Beispielprozessor ist komplett. So einfach er vom Prinzip her ist, so verflochten und vielschichtig präsentiert er sich in der konkreten Ausgestaltung. Reale Mikroprozessoren sind natürlich wesentlich komplexer: mächtige Rechenwerke, eine Vielzahl an Registern, Parallelbearbeitung von Befehlen, mehrere Zentraleinheiten auf

Schrittregister gespeicherter Wert	Register 0 Schreiben [S]	Register 1 Schreiben [S]	Register 2 Schreiben [S]	Register 3 Schreiben [S]	Programnzähler Schreiben [S]	Befehlsregister Schreiben [S]	Multiplexer 0 Wahl [W]	Multiplexer 1 Wahl [W]	Multiplexer 2 Wahl [W]	Multiplexer 3 Wahl [W]	Hauptspeicher Schreiben [S]	Rechenwerk Operation [O]
0	nein	nein	nein	nein	nein	ja	a	a	a	e	nein	I
1	nein	nein	nein	nein	ja	nein	b	e	e	a	nein	+
2	ja	nein	nein	nein	nein	nein	b	f	a	a	nein	II
3	ja	nein	nein	nein	nein	nein	a	a	a	b	nein	I
4	nein	nein	nein	nein	nein	nein	a	a	a	b	ja	I
5	ja	nein	nein	nein	nein	nein	b	b	a	a	nein	+
6	ja	nein	nein	nein	nein	nein	b	b	a	a	nein	-
7	ja	nein	nein	nein	nein	nein	b	e	a	a	nein	+
8	ja	nein	nein	nein	nein	nein	b	e	a	a	nein	-
9	nein	nein	nein	nein	nein	nein	a	a	a	a	nein	I
10	nein	nein	nein	nein	ja	nein	b	e	e	a	nein	+
11	nein	nein	nein	nein	ja	nein	b	f	e	a	nein	+
12	nein	nein	nein	nein	ja	nein	b	f	e	a	nein	-
13	nein	ja	nein	nein	nein	nein	b	f	a	a	nein	II
14	nein	ja	nein	nein	nein	nein	b	f	a	a	nein	II
15	...	...	...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...	...	...	...

Abb. 3.28 Die Steuerfunktion des Beispielprozessors in Form einer Tabelle.
(© Susanne Staubli, Eckart Zitzler)

Befehl einlesen
 Programmzähler erhöhen
 load R0,#z ausführen
 Programmzähler erhöhen
 load R0,(R1) ausführen
 Programmzähler erhöhen
 store R0,(R1) ausführen
 Programmzähler erhöhen
 add R0,R1 ausführen
 Programmzähler erhöhen
 sub R0,R1 ausführen
 Programmzähler erhöhen
 inc R0 ausführen
 Programmzähler erhöhen
 dec R0 ausführen
 Programmzähler erhöhen
 Inhalt von Register 0 prüfen
 Inhalt von Register 0 prüfen
 Programmzähler erhöhen
 Programmzähler erhöhen
 jump #z ausführen
 Programmzähler erhöhen
 jump #-z ausführen
 Programmzähler erhöhen
 Programmzähler erhöhen
 load R1,#z ausführen
 Programmzähler erhöhen
 load R2,#z ausführen
 Programmzähler erhöhen

Schrittregister gespeicherter Wert	Decoder Befehlstyp [B]	Decoder x [X]	Decoder y [Y]	Rechenwerk Null? [N]	Schrittregister zu schreibender Wert
0	egal	egal	egal	egal	1
1	load Rx,#z	0	egal	egal	2
2	egal	egal	egal	egal	0
1	load Rx,(Ry)	0	1	egal	3
3	egal	egal	egal	egal	0
1	store Rx,(Ry)	0	1	egal	4
4	egal	egal	egal	egal	0
1	add Rx,Ry	0	1	egal	5
5	egal	egal	egal	egal	0
1	sub Rx,Ry	0	1	egal	6
6	egal	egal	egal	egal	0
1	inc Rx	0	egal	egal	7
7	egal	egal	egal	egal	0
1	dec Rx	0	egal	egal	8
8	egal	egal	egal	egal	0
1	bneqz Rx	0	egal	egal	9
9	egal	egal	egal	ja	0
9	egal	egal	egal	nein	10
10	egal	egal	egal	egal	0
1	jump #z	egal	egal	egal	11
11	egal	egal	egal	egal	0
1	jump #-z	egal	egal	egal	12
12	egal	egal	egal	egal	0
1	nop	egal	egal	egal	0
1	load Rx,#z	1	egal	egal	13
13	egal	egal	egal	egal	0
1	load Rx,#z	2	egal	egal	14
14	egal	egal	egal	egal	0
1	...	...	...	...	...
...	...	...	...	...	...

Abb. 3.29 Die Schrittfunktion des Beispielprozessors in Form einer Tabelle; links finden sich Kommentare zu den einzelnen Schritten. (© Susanne Staubli, Eckart

demselben Chip usw. Kein Wunder, gehören sie doch zu den kompliziertesten Systemen, die der Mensch je gebaut hat. Doch lebende Zellen können es spielend mit ihnen aufnehmen.

Wenn Zellen Stoffe wechseln

Die anfangs herangezogene Analogie zur Küche trifft auf Zellen noch in viel höherem Maß zu als auf die Zentraleinheit eines Computers. Permanent werden hier Stoffe aus der Umwelt aufgenommen, gemäß bewährten Rezepturen verarbeitet und anschließend für verschiedenste Zwecke verwertet. Auch in den Zellen lassen sich Datenpfade finden, nur heißen sie dort Stoffwechselwege, und genauso wie im Prozessor gibt es eine übergeordnete Steuerung, die die Abläufe kontrolliert. Verfeinern wir also unseren Blick auf die Kolibakterie und studieren genauer, wie ihre Zellküche funktioniert.

Alles eine Frage der Chemie

Unsere Kolibakterie muss sich ernähren und aus der Vorratskammer ihrer Umgebung bedienen, so wie jedes andere Lebewesen auch. Der Nahrung entnimmt sie beispielsweise Bestandteile, die sie für die Herstellung lebensnotwendiger Zellbausteine wie Proteine und Nucleotide braucht. Mittels chemischer Reaktionen werden dabei kleinere Moleküle zu komplexeren Molekülen zusammengesetzt. Doch der Herstellungsprozess bedeutet Arbeit und bedarf Energie, genauso wie das Bewegen der Geißel oder die Vervielfältigung durch Zellteilung Energie erfordert. Letztere gewinnt die Kolibakterie ebenfalls aus ihrer Nahrung. Indem sie Brennstoffe wie z. B. Traubenzucker in kleinere Bestandteile zerlegt – wiederum durch chemische Reaktionen –, entsteht ein Molekül, das als biologische Energiewährung gilt: ATP. Die Herstellung dieses Moleküls wollen wir uns exemplarisch anschauen, um das Prinzip der Umwandlungsprozesse, das sowohl der Bausteinsynthese als auch der Energiegewinnung zugrunde liegt, zu verstehen.

ATP ist eine Abkürzung für Adenosintriphosphat. Dieses Molekül ist ein universell einsetzbarer Energieträger, den Zellen auf einfache und unmittelbare Art und Weise nutzen können, um Arbeit zu verrichten. Es ist quasi das Holz, mit dem die Kolibakterie ihren Herd befeuert, nur dass hier kein Streichholz, sondern Wasser zum Einsatz kommt, um Energie freizusetzen. Millionen von ATP-Molekülen benötigt die Kolibakterie pro Sekunde, damit sie ihre gesamte biochemische Maschinerie antreiben kann.

Beim Menschen ist es noch eindrücklicher: Wenn wir von einem durchschnittlichen Gewicht ausgehen, bauen wir bereits im Ruhezustand etwa 40 kg ATP pro Tag auf und ab. Wir sehen: ATP ist als Energiequelle unverzichtbar. Da erstaunt es nicht, dass die Kolibakterie nicht nur einen Weg kennt, um diesen Stoff aus der Nahrung zu gewinnen; Traubenzucker ist nur eine mögliche Quelle. Allerdings sind viele kleine Schritte für die Herstellung notwendig, wie Bsp. 3.15 aufzeigt. Das erinnert fast schon an die Kleinschrittigkeit der Turing-Maschine.

Beispiel 3.15: Wie aus Traubenzucker Energie gewonnen wird

Der Umwandlungsprozess des Traubenzuckers – Glykolyse genannt – ist in Abb. 3.30 skizziert und vollzieht sich in zwei Stufen, wobei jede Stufe eine Serie von Reaktionen umfasst. In der ersten Stufe wird der Traubenzucker vorbereitet und in eine Zwischenform transformiert, wobei ein Traubenzucker-Molekül in zwei Moleküle namens GAP überführt wird. In diesem Vorverarbeitungsprozess wird keine Energie gewonnen, im Gegenteil: Zwei ATP-Moleküle müssen aufgewendet werden, damit die Umwandlung vollzogen wird. Erst in der zweiten Stufe, wenn GAP zu dem Molekül Pyruvat weiterverarbeitet wird, entstehen zwei ATP-Moleküle. Da aus einem Traubenzucker-Molekül zwei GAP- bzw. Pyruvat-Moleküle entstehen, ergibt sich insgesamt eine positive Energiebilanz.

Der ganze Ablauf ist hier sehr vereinfacht dargestellt – und auch nicht vollständig, denn das Pyruvat wird noch weiterverwertet. Bereits an den vielen Namen, die Biochemiker in diesem Zusammenhang für die Moleküle verwenden, lässt sich erahnen, wie kompliziert die Vorgänge im Detail sind.

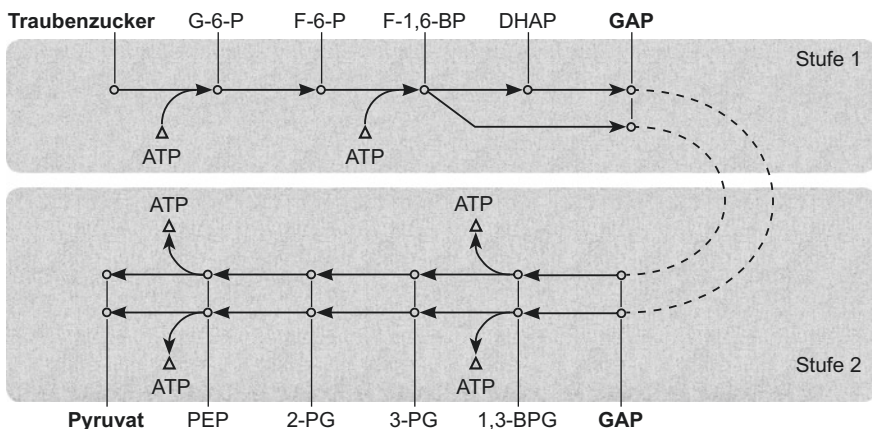


Abb. 3.30 Die Glykolyse: wie die Kolibakterie aus Traubenzucker ATP-Moleküle herstellt. (© Susanne Staubli, Eckart Zitzler)

Solch eine Abfolge von ineinandergreifenden Reaktionen, bei denen Moleküle auf bestimmte Weise in ein oder mehrere Moleküle überführt werden und so Energie umgewandelt wird, nennt man Stoffwechselweg. Der Verarbeitungspfad vom Traubenzucker zum ATP ist nur ein Beispiel, es gibt viele weitere dieser Stoffwechselwege, die auf komplexe Art und Weise miteinander interagieren. Über tausend chemische Reaktionen laufen in so einem einfachen Wesen wie der Kolibakterie ab. In seiner Gesamtheit bezeichnet der Stoffwechsel ein ganzes Netz an miteinander verwobenen chemischen Reaktionen, das für Gewinnung von Energie aus der Umgebung sowie für die Herstellung von Zellbausteinen zuständig ist. Stellt man die Wege grafisch dar (Abb. 3.31), so wird ein imposantes Verarbeitungsnetzwerk sichtbar.

Dieses Wegenetz lässt sich durchaus mit dem Datenpfad im Prozessor vergleichen. Es zeigt imaginäre Pfade, entlang derer die Stoffe – sozusagen die Daten der Zelle – umgewandelt werden. Leitungen gibt es in der Kolibakterie keine – zumindest nicht in der Form, wie wir das beim

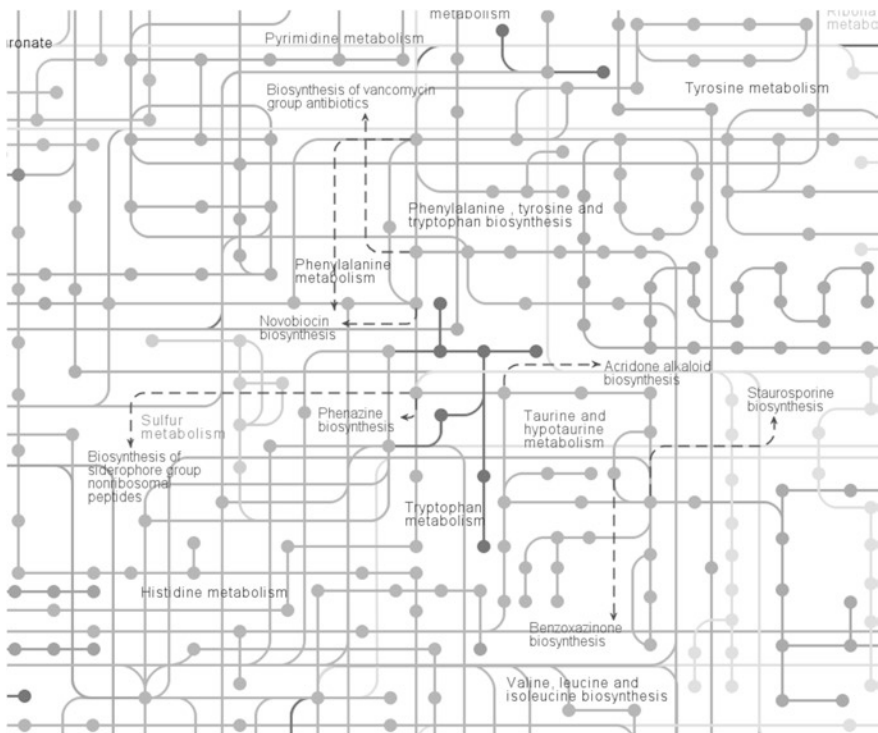


Abb. 3.31 Visualisierung von Stoffwechselwegen in einer Zelle, wobei Punkte für Moleküle und Linien für Reaktionen stehen. (© Kyoto Encyclopedia of Genes and Genomes, Kanehisa Laboratories)

Datenpfad diskutiert haben. Die Reaktionen sind spezifisch verknüpft über ihre Substrate, also jene Moleküle, die in die jeweiligen Reaktionen involviert sind. Der auffälligste Unterschied ist jedoch, dass es in der Kolibakterie kein zentrales, allgemein einsetzbares Rechenwerk gibt, das alle möglichen Rechenoperationen vereint. In dem Einzeller sind die Reaktionen verteilt und können an verschiedenen Orten stattfinden; jede von ihnen steht für einen bestimmten Verarbeitungsschritt.

Diese Gegenüberstellung zieht die Frage nach dem Kontrollpfad nach sich: Wie wird der Stoffwechsel gesteuert, und laufen die über tausend Reaktionen gleichzeitig und ständig ab? Aus Kap. 1 kennen wir die Antwort bereits teilweise: Die Kolibakterie geht äußerst effizient mit ihren Ressourcen um und kann Umwandlungsprozesse an- und abschalten. Als Schalter fungieren hierbei Proteine, sie regulieren auch die einzelnen Verarbeitungsvorgänge.

Regulieren, regulieren, regulieren

Wie kann nun ein Protein eine chemische Reaktion beeinflussen? Grundsätzlich kann jede chemische Reaktion in beide Richtungen verlaufen und es gibt ein chemisches Gleichgewicht, bei dem sich beide Richtungen die Waage halten. Für sich genommen laufen diese Reaktionen jedoch viel zu langsam ab, als dass es für die Zelle gewinnbringend wäre. Der Grund ist, dass es eine Energieschwelle zu überwinden gilt. Das bewerkstelligen Katalysatoren, also Substanzen, die Reaktionen beschleunigen können. Die Kolibakterie nutzt hauptsächlich Proteine dafür, die in dieser Funktion Enzyme genannt werden. Enzyme sind hocheffizient und können die Reaktionsgeschwindigkeit bis um das Millionenfache erhöhen. Auch für uns Menschen sind diese Beschleuniger enorm wichtig: Ungefähr ein Viertel unserer Gene codiert Enzyme.

Enzyme bilden spezielle Zentren, in denen sich Substrate einnisten und dadurch besser miteinander reagieren können. Sie können beispielsweise Moleküle räumlich optimal zueinander ausrichten, sodass die Reaktion reibungslos vonstattengehen kann; sie können Moleküle unter Spannung setzen, damit diese destabilisiert und somit reaktionsfreudiger werden, oder Molekülen zeitweilig chemische Gruppen hinzufügen, damit sie gespalten werden können. Enzyme sind hochspezifisch für die beteiligten Moleküle einer Reaktion; in dem Stoffwechselweg der Traubenzuckerwertung (Abb. 3.30) gibt es für jeden einzelnen Verarbeitungsschritt ein zuständiges Enzym. Entscheidend dabei ist die räumliche Struktur des Enzyms, denn sie

muss die richtige Passform für die Substrate haben. Bsp. 3.16 zeigt auf, wie sich diese aus der eindimensionalen Blaupause auf der DNA ergibt.

Beispiel 3.16: Molekulares Origami

Ein Enzym ist – wie jedes Protein – ein aus Aminosäuren aufgebautes Molekül, das man sich als verknäulte Perlenkette vorstellen kann. Die Aminosäuresequenz, d. h. der Bauplan eines Proteins, ist auf der DNA festgeschrieben. Wenn ein Protein hergestellt wird, werden einzig die Aminosäuren in der auf der DNA definierten Reihenfolge aneinandergelagert, eine räumliche Struktur ist damit noch nicht verbunden. Das Besondere ist nun, dass Proteine sich falten, als würden Sie eine Halskette locker auf dem Tisch zusammenlegen und so ein dreidimensionales Gebilde kreieren. Der Faltvorgang ist allerdings nicht zufällig, vielmehr ergibt sich aufgrund der chemischen Eigenschaften der Kettenglieder eine ganz bestimmte Struktur. In der Aminosäurekette bilden sich zunächst kleine, dreidimensionale Unterstrukturen, Spiralen und Faltblätter; anschließend ordnen sich die Elemente zu einem räumlichen Ganzen an. Abb. 3.32 gibt Ihnen einen Eindruck von der Struktur eines Enzyms.

Die Enzyme bilden also die Kontrollinstanz, über die die Kolibakterie den Fluss entlang der Stoffwechselwege regulieren kann; sie sind Werkzeug, Steuerleitung und Steuersignal in einem. Die Steuerung kann dabei aufgrund verschiedener Mechanismen erfolgen. Beispielsweise können Zwischen- oder Endprodukte der Umwandlungskette als Hemmstoffe wirken und die Aktivität von Enzymen mindern, indem sie an diese binden und das Funktionszentrum blockieren. Manche Substrate können auch

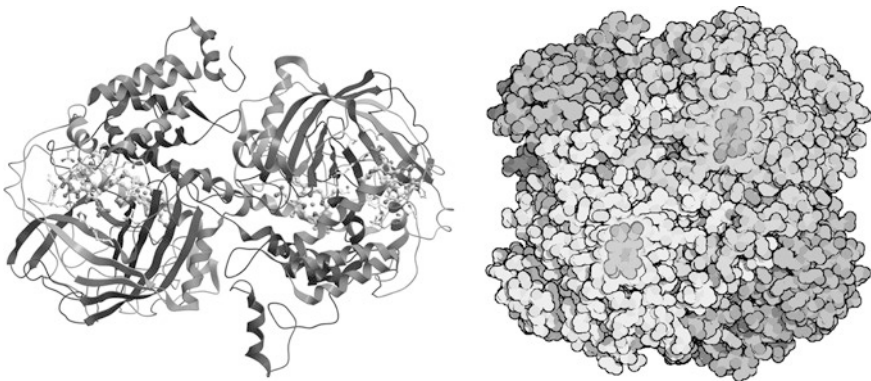


Abb. 3.32 Substrukturen (links) und Gesamtstruktur (rechts) des Enzyms Katalase, das beim Menschen vor allem in Leber und Niere vorkommt. (© Leonid Andronov/stock.adobe.com (links), David S. Goodsell, RCSB PDB (rechts))

eine Veränderung der räumlichen Struktur bewirken, sodass ein Enzym seine Funktion nicht mehr ausüben kann. Eine andere Art der Steuerung besteht darin, dass Schalterproteine den DNA-Ablesevorgang und somit die Herstellung von Enzymen beeinflussen. So etwas wie einen Takt gibt es dabei allerdings nicht: Die Verarbeitungsschritte werden individuell abgestimmt, jede Reaktion hat ihre eigene Geschwindigkeit.

Die Diskussion zeigt: Ähnlich wie beim Prozessor gibt es zwei Ebenen bei den Umwandlungsprozessen in der Kolibakterie. Auf der Verarbeitungsebene sind es die Substrate, auf der Steuerungsebene die Enzyme, die wiederum durch das genetische Programm gesteuert werden. Die Ebenen greifen ineinander, weil Enzyme mit Substraten in Wechselwirkung stehen – sowohl im Sinne von Steuerleitungen, denn Enzyme beeinflussen Substrate, als auch im Sinne von Statusleitungen, denn Substrate können auf Enzyme rückwirken. Zellfabrik und Datenfabrik teilen also gemeinsame Prinzipien.

Fazit

Ein Prozessor arbeitet eine Berechnung schrittweise in einem vorgegebenen Zeitraster ab. Der Datenpfad stellt die Infrastruktur zur Verarbeitung und Speicherung der Zahlen zur Verfügung, sie ist für alle Rechenschritte dieselbe. Der Kontrollpfad steuert den Berechnungsvorgang und weist den Datenpfad an, was zu tun ist; die konkrete Rechenvorschrift in Form eines Programms entnimmt er dem Hauptspeicher. Beide Schaltungen, Datenpfad und Kontrollpfad, kommunizieren über Steuer- und Statusleitungen, um Anweisungen bzw. Rückmeldungen zu geben. Auch bei lebenden Zellen lässt sich diese Trennung zwischen Verarbeitung und Ablaufsteuerung wiederfinden. Sie wandeln Stoffe aus ihrer Umgebung über ein Netz an miteinander gekoppelten chemischen Reaktionen um, wobei jede Reaktion einen spezifischen Verarbeitungsschritt verkörpert. Die Umwandlungsprozesse unterliegen einer strengen Regulation, in deren Zentrum hochspezialisierte Proteine stehen; diese steuern die einzelnen Reaktionen und treten in Wechselwirkung mit den zu verarbeitenden Stoffen.

Zum Nach- und Weiterlesen

- Berg, J. M., Stryer, L., & Tymoczko, J. L. (2012). *Stryer Biochemie* (7. Aufl.). Berlin: Springer Spektrum.
- Floyd, T. L. (2012). *Digital fundamentals: A systems approach*. Upper Saddle River: Pearson Education.
- Hoffmann, D. W. (2014). *Grundlagen der Technischen Informatik* (4. Aufl.). München: Hanser.
- Patterson, D., & Hennessy, J. L. (2011). *Rechnerorganisation und Rechnerentwurf: Die Hardware/Software-Schnittstelle* (4. Aufl.). München: Oldenbourg.
- Sadava, D., Orians, G., Heller, H. C., Hillis, D., & Berenbaum, M. R. (2011). *Purves, Biologie* (9. Aufl.). Heidelberg: Spektrum Akademischer.

Bildquellen

- Abb. 3.1: Tobif82, Bildnummer 36019512, <https://stock.adobe.com> (links, abgerufen am 30.05.2017); olvius, Bildnummer 84087050, <https://stock.adobe.com> (rechts, abgerufen am 30.05.2017)
- Abb. 3.13: ilbusca/Getty Images/iStock, Bildnummer 486401242, <http://www.istockphoto.com> (rechts, abgerufen am 30.05.2017)
- Abb. 3.18: Gudellaphoto, Bildnummer 92781870, <https://stock.adobe.com> (links, abgerufen am 30.05.2017); Christian Schwier, Bildnummer 24747407, <https://stock.adobe.com> (rechts, abgerufen am 30.05.2017)
- Abb. 3.31: Kanehisa, M. & Goto, S. (2000). *KEGG: Kyoto Encyclopedia of Genes and Genomes*. Nucleic Acids Research, 28, S. 27–30, <http://www.kegg.jp> (abgerufen am 25.05.2017)
- Abb. 3.32: Leonid Andronov, Bildnummer 55989796, <https://stock.adobe.com> (links, abgerufen am 30.05.2017); David S. Goodsell, RCSB Protein Data Bank, <http://pdb101.rcsb.org/motm/57> (rechts, abgerufen am 26.05.2017)

4

Computer-Anatomie

Überblick

Dieses Kapitel beleuchtet, wie es im Innern eines Computers aussieht. Dazu betrachten wir, welche Bestandteile es braucht, damit aus einem Prozessor ein vollwertiger Computer wird; wie diese Bestandteile zusammenspielen, sodass ein Computer mit seiner Umwelt interagieren kann; wie das Ganze zum „Leben“ erweckt wird und was dafür nötig ist. Zunächst lernen wir den Urbauplan kennen, der der Organisation eines Computers seit Mitte des 20. Jahrhunderts zugrunde liegt. Er zeigt auf, wie sich Prozessor, Speicher sowie Ein- und Ausgabegeräte als Gesamtsystem konzipieren lassen. Anschließend beschäftigen wir uns mit den Möglichkeiten, wie sich diese Architektur verbessern und beschleunigen lässt – auf der Hardware-Ebene. Zum Schluss wenden wir uns der Frage zu, wie sich mehrere Programme gleichzeitig auf einem Computer ausführen lassen und welche Herausforderungen es dabei zu lösen gilt – auf der Software-Ebene. Im Fachjargon ausgedrückt: Es geht um Rechnerarchitekturen und Betriebssysteme.

Anatomie kommt aus dem Griechischen, abgeleitet vom Wort für aufschneiden, und steht für die Lehre des Aufbaus der Lebewesen oder des Körperbaus allgemein. Es hat etwas gedauert, bis die Neugierde obsiegte und der Mensch wagte, sein Inneres zu erforschen. Lange Zeit war das Sezieren von Menschen verboten, erst im 16. Jahrhundert wurde die moderne Anatomie begründet. Leonardo da Vinci, das berühmte Universalgenie, war einer der Wegbereiter; er betrieb umfangreiche Körperstudien und fertigte diverse anatomische Skizzen an, z. B. „Der vitruvianische Mensch“ (Abb. 4.1 links). Heute wissen wir: Der Mensch ist ein hochkomplexes Wesen. Er besteht aus verschiedenen Körperteilen, Organen und Geweben, die in einem mannigfaltigen Wechselspiel zueinander stehen und vor

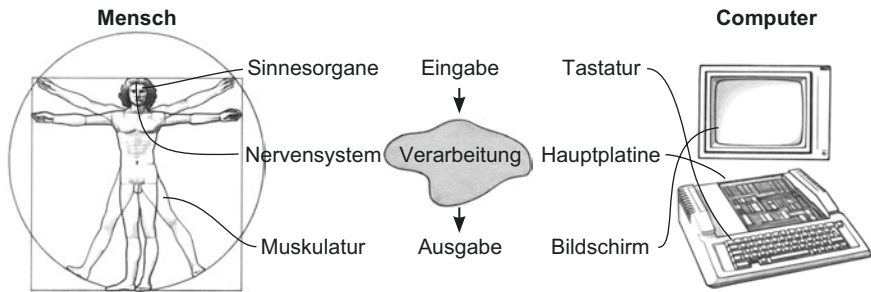


Abb. 4.1 Eine stark vereinfachte Gegenüberstellung von Mensch und Computer. (© Susanne Staubli, Eckart Zitzler)

allem über sein Nervensystem koordiniert werden. All das hat sich über Jahrmillionen entwickelt als Anpassungsleistung an unseren Lebensraum.

Auch Computer weisen eine vielschichtige Struktur auf und setzen sich aus vielen, miteinander interagierenden Komponenten zusammen. Im Zentrum steht dabei der Prozessor, die nackte Verarbeitungseinheit. Er wird angereichert mit weiteren Funktionseinheiten zur Aufnahme, Speicherung und Weitergabe von Daten. Stellen wir das System Computer dem System Mensch gegenüber (Abb. 4.1), dann entsprechen die Eingabe- und Ausgabevorrichtungen, beispielsweise Tastatur und Bildschirm, den Sinnesorganen bzw. der Muskulatur, während die Hauptplatine, auf der die meisten Schaltungskomponenten untergebracht sind, das Nervensystem verkörpert. So oberflächlich dieser Vergleich auch ist, er hilft uns zu verstehen, warum Computer so konstruiert sind, wie sie sich uns heute präsentieren.

Machen wir uns also ans Werk und sezieren das Innere eines Computers. Damit wir nicht den Überblick verlieren, werden wir bei einfachen Rechnerarchitekturen beginnen, uns langsam vorarbeiten zu schnelleren, komplexeren Computersystemen und die Parallelverarbeitung – die gleichzeitige Ausführung mehrerer Programme – dann zum Schluss studieren.

Eingebettet sein: im Austausch mit der Welt

Bislang haben wir die automatisierte Informationsverarbeitung entrückt von der Welt betrachtet, das Außen blieb außen vor. Die Daten gelangten irgendwie auf das Band bzw. in den Speicher und die Ergebnisse konnten irgendwie von dort wieder ausgelesen werden. Natürlich, es gibt da eine einfache Lösung: Wir statten den Prozessor mit Ein- und Ausgabeleitungen aus, über die Daten

in den Datenpfad eingespeist bzw. von ihm abgegriffen werden können. Das wird auch effektiv so gemacht, doch ganz so einfach ist es dann doch nicht. In der Regel müssen mehrere, bei eingebetteten Systemen sogar eine Vielzahl von Eingabe- und Ausgabeeinheiten koordiniert werden, Speicherbausteine und andere Bausteine kommen noch hinzu. Wir müssen uns folglich Gedanken darüber machen, wie wir aus all den Bestandteilen ein gegliedertes System machen, ein Computersystem. Dazu möchte ich ein paar grundsätzliche Überlegungen, wie Systeme konzipiert sind, voranstellen.

Anatomische Grundlagen

Damit aus einem Prozessor tatsächlich ein richtiger Computer wird, muss er in die Welt eingebettet und in die Lage versetzt werden, mit ihr interagieren zu können. Es bedarf also Vorrichtungen, mit denen wir – analog zu unseren Sinnesorganen und Muskeln – die Umwelt wahrnehmen und in sie eingreifen können. Die ursprüngliche Umgebung eines Computers bestand aus einem Raum, in dem eine Eingabe von einem Menschen über eine Tastatur vorgenommen wurde und eine Ausgabe auf einen Bildschirm oder Drucker erfolgte. Dieses Szenario ist in Abb. 4.2 rechts dargestellt, wo Sie einen der ersten Desktop-Computer sehen, bei dem die Verarbeitungskomponenten auf der erwähnten Hauptplatine untergebracht waren, dem *mother board*. Mittlerweile haben sich die Eingabe- und Ausgabemöglichkeiten stark erweitert, aus einem passiven Verhalten wurde zunehmend ein aktives Verhalten. Beispielsweise greifen Computer, die Autos oder Flugzeuge steuern, direkt in ihre Umwelt ein. Beim Computer von Verhalten sprechen zu wollen, erscheint etwas weit hergeholt, doch auch er interagiert mit der Umwelt und verhält sich darin. Informationen werden durch geeignete sensorische Komponenten in eine digitale Darstellung transformiert, die resultierenden Eingabemuster anschließend verarbeitet und die Ausgabemuster in entsprechende Aktionen umgesetzt, sei es nun als Leuchten von Lampen oder als Drehbewegung eines Motors. Wenn wir Mensch und Computer gegenüberstellen, können wir also sagen: Aus Wahrnehmungen werden Eingaben, aus Reaktionen Ausgaben.

Das Verhalten ist sowohl beim Menschen als auch beim Computer das, was wir von außen beobachten können. Das menschliche Verhalten ist darauf ausgerichtet, unser Leben – welches ständig gefährdet ist, von innen wie von außen – zu erhalten; beim Computer richtet sich das Verhalten nach dem Anwendungszweck. Schauen wir ins Innere hinein, so erkennen wir dort eine Struktur, eine Vielzahl verschiedener Elemente, die in einem zielgerichteten

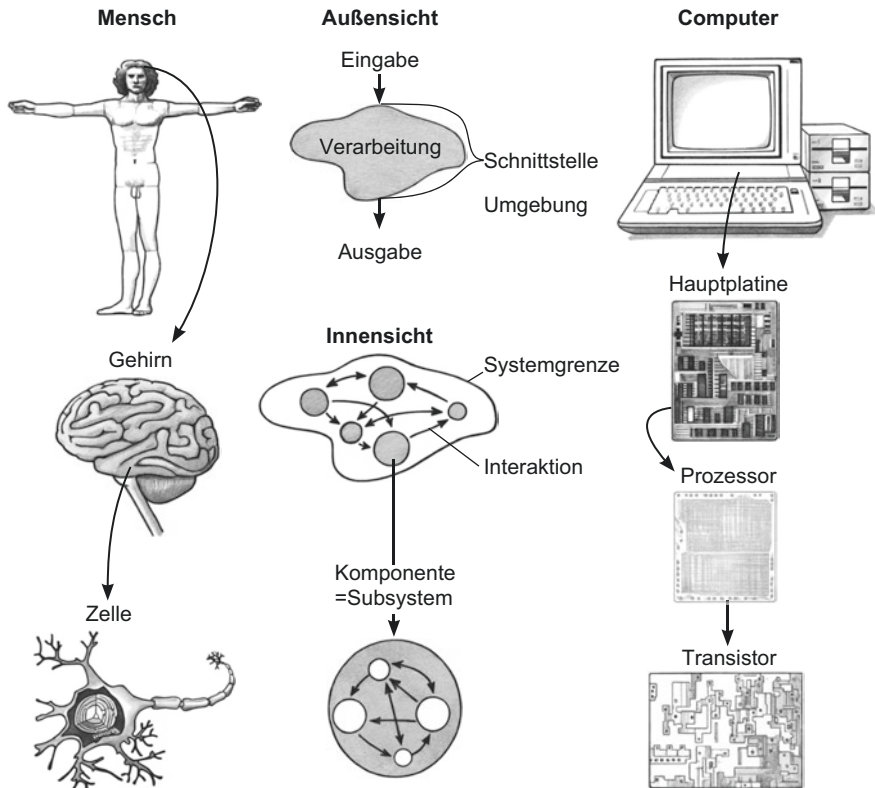


Abb. 4.2 Systeme und Subsysteme in der Gegenüberstellung von Mensch und Computer. (© Susanne Staubli, Eckart Zitzler)

Wechselspiel das beobachtbare Verhalten hervorbringen. Beim Menschen sind dies Organsysteme wie der Bewegungsapparat und Organe wie das Gehirn, die sich selbst wiederum aus kleineren Bestandteilen – Geweben und Zellen – zusammensetzen. Das ist beim Computer natürlich genauso, er besteht auch aus verschiedenen spezialisierten Funktionseinheiten und jede Funktionseinheit setzt sich wiederum zusammen aus diversen Bausteinen. Beim Prozessor hatten wir das ja durchexerziert, er besteht aus Bestandteilen wie Registern, Multiplexern, einem Rechenwerk usw., die selbst wiederum aus Gattern und – auf unterster Ebene – aus Transistoren bestehen (vgl. Abb. 4.2 rechts). Ein Computer (wie der Mensch) ist also ein System aus Systemen. Er bildet ein System aus Bausteinen, welches selbst wiederum aus kleineren Bausteinen und Schaltelementen besteht (siehe Anm. 4.1).

Anmerkung 4.1: Systeme aus Systemen

Ein System ist nichts anderes als ein gegliedertes Ganzes, dessen Komponenten in ihrem Zusammenspiel eine zweckgebundene Einheit ergeben. Die Komponenten können wiederum Systeme sein. Systeme sind in der Regel von ihrer Umwelt durch eine Hülle abgegrenzt, die das Außen von dem Innen unterscheidet. Es gibt zwar auch offene Systeme – das Immunsystem ist ein Beispiel –, doch die werden wir im Folgenden nicht betrachten. Das ist beim Herz so, beim Gehirn und auch beim Prozessor. Gleichzeitig weisen diese Systeme eindeutige Zugänge nach außen bzw. innen auf: Beim Herzen sind das die Blutgefäße, die Blut zum und vom Herzen transportieren, beim Prozessor sind es die Kontakte am Gehäuse, die die Verbindung zur Außenwelt herstellen. In der Informatik wird dafür der Begriff Schnittstelle verwendet. Er bezieht sich auf die Art und Weise, wie ein System mit seiner Umwelt in Verbindung tritt; im Englischen lautet der Begriff zutreffender *interface*. Die Schnittstelle des Systems Mensch zu seiner Umwelt ist seine Haut. Dieses zwei Quadratmeter große Organ ist einerseits Schutz und andererseits Kommunikationsmedium, das den verschiedensten Zwecken dient und verschiedenste Öffnungen nach außen aufweist wie z. B. die Hautporen, den Mund, die Ohren usw.

Im Folgenden werden wir uns die Komponenten des Systems Computer anschauen und ihr Zusammenspiel studieren. Allerdings werden wir nicht mehr alle Schichten hinunter zur Transistorebene gehen im Bewusstsein, dass viele Funktionseinheiten wie z. B. eine Kamera oder ein Greifarm für sich genommen komplexe System darstellen. Dem Speicher habe ich aus diesem Grund ein eigenes Kapitel gewidmet. Zudem müssen wir uns vor Augen halten, dass wir bei der Gegenüberstellung von Mensch und Computer vieles ausgeblendet haben. Das Leben zeichnet sich neben seiner Informationsverarbeitungs- und Kommunikationsfähigkeit durch viele andere Merkmale wie Stoffwechsel, Vermehrung sowie Wachstum und Entwicklung aus. Beim Menschen dient ein Großteil des Organismus der Energiegewinnung und dem Selbsterhalt. Beim Computer sind solche Aspekte zunächst einmal nicht von Belang – der Strom kommt ja aus der Dose. Doch sobald es um autonome Computersysteme wie einen Mars-Roboter geht, sind Aspekte wie Energieversorgung und Robustheit gegenüber Umwelteinflüssen von zentraler Bedeutung.

Historisch gesehen sind Computer vom Kopf her gedacht, per se steht die Informationsverarbeitung im Zentrum. Anfangs waren die Ein- und Ausgabemöglichkeiten sehr beschränkt, erst mit der Zeit wurde der „Körper“ um den Prozessor immer aufwändiger. Beim Menschen hat sich das Gehirn zusammen mit dem Organismus entwickelt; das Nervensystem steht nicht für sich allein, sondern ist eng mit dem Körper verwoben, steht in seinem Dienst und erweitert somit unsere Fähigkeiten.

Der Urbauplan

Unser Nervensystem bietet uns erstaunliche Möglichkeiten. Wir können mit abstrakten Zahlen im Geiste jonglieren und Hilfsmittel konstruieren, die uns diese Tätigkeit abnehmen. Doch bevor elektronische Rechenmaschinen die Welt erblickten, waren Computer menschlich: Im 19. Jahrhundert verstand man im Englischen unter dem Begriff *computer* einen Menschen, der rechnete. Solche menschlichen Rechner wurden eingesetzt, um im großen Stil zu kalkulieren. Da waren viele Menschen in einem Raum untergebracht und es gab eine strenge Aufgabenteilung (Abb. 4.3). Die einen führten klar definierte Rechenaufgaben durch (und durften nicht davon abweichen), andere versorgten diese *human computers* mit Aufgaben, wiederum andere führten die Ergebnisse zusammen und kontrollierten sie, ein paar wenige überwachten das Ganze. Im Rückblick wirkt das ganze Szenario so, als wollte jemand die Funktionsweise eines Computers in einem Theaterstück vorführen, denn die Struktur eines heutigen Computers spiegelt diese Aspekte wider. Doch auf diese Weise konnte man damals z. B. die täglichen Positionen von Himmelskörpern vorausberechnen, das hätte ein Einzelner nie geschafft (siehe Anm. 4.2).

Anmerkung 4.2: Human Computers

Der Nautical Almanac ist ein astronomisches Jahrbuch, das für die Nautik, die Navigation auf See, konzipiert wurde. Es erschien zum ersten Mal 1767 und enthält Vorausberechnungen für die Positionen von sich bewegenden Himmelskörpern wie Sonne, Mond, Planeten und Kometen. Diese Ephemeriden genannten Positionswerte sind wichtig für die Orientierung; aus der Distanz zum Mond lässt sich beispielsweise mittels der Ephemeriden der Längengrad der aktuellen Position ermitteln.

Schon früh wurden menschliche Computer eingesetzt, um diese Vorausberechnungen vorzunehmen. Anfangs arbeiteten diese von zu Hause und waren teilweise weit voneinander entfernt. Als der Nautical Almanac 1830 einer Begutachtung unterzogen wurde, an der übrigens auch Charles Babbage beteiligt war, wurde ein neues Nautical Almanac Office gegründet und die menschlichen Computer wurden gebündelt in einem Raum untergebracht. Häufig handelte es sich um junge Männer, teilweise erst 15 Jahre alt. Sie nahmen um 8 Uhr ihren Dienst auf, um 12 Uhr gab es eine Mittagspause, anschließend arbeiteten die Männer weiter bis 20 Uhr. Gut, dass heutzutage elektronische Rechenhirne solche Arbeiten übernehmen können.

Abb. 4.4 zeigt ein hypothetisches, stark vereinfachtes Beispiel, wie es in so einem Rechenzentrum damals ausgesehen haben könnte. Am Schreibtisch sehen Sie die menschlichen Computer, die effektiv die Rechenoperation



Abb. 4.3 Menschliche „Computer“ in der ersten Hälfte des 20. Jahrhunderts. (© Library of Congress (links), NASA (rechts))

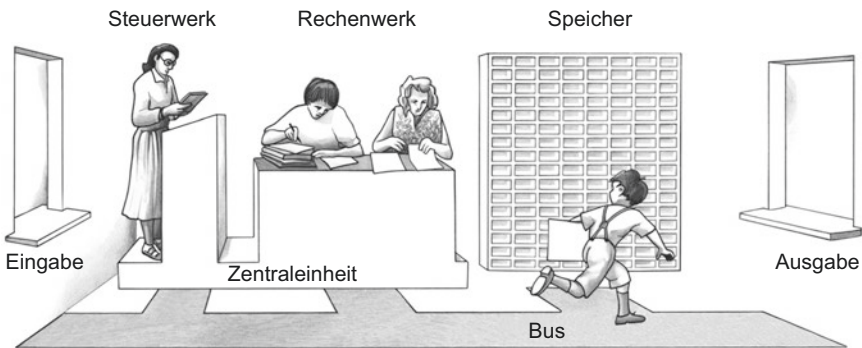


Abb. 4.4 Illustration einer Rechnerarchitektur am Beispiel der menschlichen Computer. (© Susanne Staubli, Eckart Zitzler)

durchführten. Sie wurden permanent versorgt mit neuen Zahlen, die in einer zentralen Ablage, z. B. einem Regal, aufbewahrt wurden; in diesem Regal wurden auch Zwischenergebnisse gelagert. Für den Zahlentransport war ein Laufbursche zuständig, der ständig auf den Gängen hin- und herflitzte. Er konnte auch neues „Zahlenfutter“ an einem Eingabeschalter entgegennehmen sowie Rechenergebnisse für den Druck an einem Ausgabeschalter abliefern. Schließlich brauchte es noch eine Person, die den Überblick hatte und die anderen, also menschliche Rechner und Laufbursche, entsprechend instruierte; diese Anweiserin steuerte den Ablauf der Berechnung.

Übersetzen wir dieses Szenario in die Welt des elektronischen Computers, dann verkörpern die menschlichen Rechner das Rechenwerk des Prozessors, die Anweiserin steht für das Steuerwerk des Prozessors, das

Regal repräsentiert den Hauptspeicher, die Ein- und Ausgabeschalter entsprechen den Ein- und Ausgabegeräten wie Tastatur und Bildschirm und die Gänge mitsamt Laufburschen sind ein Leitungsstrang, Bus genannt, der Prozessor, Speicher, Tastatur und Bildschirm miteinander verbindet. Tatsächlich beschreibt dieses Bild den Urbauplan eines Computers, der in den 1940er-Jahren entstanden ist. Und an dieser grundsätzlichen Architektur hat sich seither wenig geändert: Auch moderne Computer bauen auf dieser internen Organisation auf, so rasant die technologische Entwicklung auch war und ist. Werfen Sie z. B. einen Blick auf einen Desktop-Computer aus den 1980er-Jahren (Abb. 4.5), dort sind die beschriebenen Komponenten gut ersichtlich.

Das ist also das Prinzip: im Zentrum Prozessor und Speicher, daneben Ein- und Ausgabegeräte, und alle über einen Bus verbunden. Der Bus ist quasi die Datenautobahn, über die die anderen Komponenten miteinander kommunizieren. Der große Vorteil des Bus-Konzepts ist seine Flexibilität, d. h., weitere Geräte können einfach in das Gesamtsystem eingebunden werden. Das klingt schon fast trivial: Einfach ein Leitungsbündel zwischen die Geräte packen und fertig ist der Computer. Doch dieser Aufbau war zu dem Zeitpunkt, als er vorgeschlagen wurde, revolutionär, und zwar

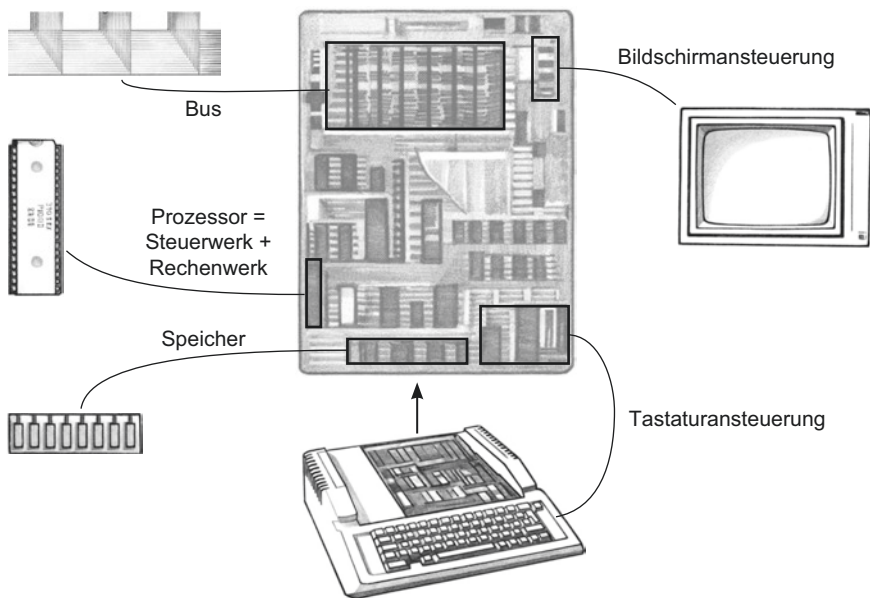


Abb. 4.5 Wie sich die Von-Neumann-Architektur in einem Desktop-Computer abbildet. (© Susanne Staubli, Eckart Zitzler)

wegen eines anderen Aspekts: Das Steuerwerk entnimmt den Rechenablauf dem Hauptspeicher, es ist keine Umverkabelung notwendig, um andere Rechenwege zu programmieren (Anm. 4.3). Diesem Konzept sind alle gängigen Computer treu geblieben und daher erscheint es uns aus heutiger Sicht auch selbstverständlich.

Anmerkung 4.3: Ein Programm ist ein Datenblock im Hauptspeicher

Das Neue an der erwähnten Architektur war, Programmdaten und zu verarbeitende Daten im gleichen Speicher abzulegen – nach diesem Konzept haben wir den Prozessor in Kap. 3 entworfen. Zu Ende gedacht heißt das, im Hauptspeicher abgelegte Daten können sowohl als Befehle wie auch als Zahlen oder anderweitige Daten interpretiert werden. Demnach können ein Text und ein Programm mit dem gleichen Eingabemedium in den Computer geladen werden, ja, ein Programm lässt sich sogar als Text anschauen und umgekehrt. Bei einer Turing-Maschine hieße das, Eingabe- und Programmdaten lägen auf dem gleichen Band, wie es in Abb. 4.6 dargestellt ist. Aus heutiger Sicht erscheint das unspektakulär, kalter Kaffee eben, aber damals war das neu. Der ENIAC beispielsweise, der erste vollelektronische Computer (vgl. Kap. 1), wurde programmiert, indem Menschen von Hand Kabel umverlegten und die Hardware manuell änderten. Da konnte es durchaus mehrere Wochen dauern, bis das Programm lief. Stellen Sie sich das mal vor: Sie müssten erst einen Fachmann kommen lassen, der Ihren Computer neu verdrahtet, wenn Sie von Foto- auf Textbearbeitung wechseln wollen! Das wäre so, als müsste Ihnen ein Arzt Ihre Nervenverbindungen im Gehirn nachts von Hand umstöpseln, statt dass sich die Lernprozesse selbstständig festigen.

Übrigens war es der in Abb. 4.7 zu sehende John von Neumann, ein Mathematiker österreichisch-ungarischer Herkunft, der diesen Urbauplan

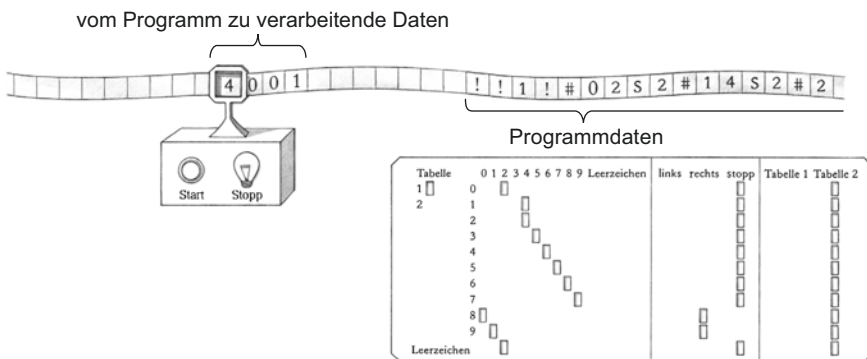


Abb. 4.6 Eine Turing-Maschine, bei der Programm und Eingabedaten auf dem gleichen Band liegen. (© Susanne Staubli, Eckart Zitzler)



Abb. 4.7 John von Neumann. (© Camille Büsser)

1945 in einem Bericht beschrieb; die Architektur wird daher auch Von-Neumann-Architektur genannt. John von Neumann gilt wie Alan Turing als einer der Väter der Informatik und hat sich wie dieser auch intensiv mit der Biologie beschäftigt. In seinem letzten Buch mit dem Titel *Die Rechenmaschine und das Gehirn* geht er den Wechselbeziehungen zwischen diesen informationsverarbeitenden Systemen nach. Er hat sich nicht nur mit dem Gehirn beschäftigt, sondern auch mit Zellen und der Zellkommunikation; das Ergebnis waren die sogenannten zellulären Automaten, die ein eigenes Forschungsgebiet repräsentieren.

Natürlich handelt es sich bei der Von-Neumann-Architektur nur um ein bewährtes Schema, im Detail können die Hardware-Architekturen sehr unterschiedlich sein. Aus diesem Grund gibt es ja auch ganz verschiedene Spielarten von Computern, wie wir bereits in Kap. 1 gesehen haben. Genauso wie wir Menschen sehr unterschiedlich sein können und wie auch unser Körperbau einem Schema folgt, dem der Säugetiere; und wie das wiederum einem übergeordneten Schema folgt, dem der Wirbeltiere. Im Folgenden nehmen wir die Von-Neumann-Architektur genauer unter die Lupe und beginnen bei den Ein- und Ausgabegeräten. Danach stelle ich Ihnen den Bus und schließlich noch das Betriebssystem vor, jenes Grundprogramm, welches die Hardware zum Leben erweckt. Die Speichergeräte werden hingegen in anderen Kapiteln behandelt, einfach weil die Thematik so reichhaltig ist: Der Aufbau des Hauptspeichers ist Gegenstand von Kap. 3, alle weiteren Arten von Speichern stelle ich Ihnen in Kap. 5 vor.

Wie die Welt in den Computer kommt – und wieder hinaus

Schauen wir also die Komponenten an, mit denen ein Computer Informationen aufnehmen und weitergeben kann: die Ein- und die Ausgabegeräte. Sie stellen die Schnittstelle zur Umwelt dar und vermitteln zwischen Außen- und Innenwelt, indem sie physikalische Größen – chemische, optische, akustische und anderweitige Signale – in Aktivitätsmuster von Stromkreisen übersetzen und umgekehrt (Abb. 4.8). Die einfachste Variante der Umweltinteraktion erfolgt über Schalter und Lampen, wie wir das beim Taschenrechner in Kap. 2 gesehen haben; auf diese Weise lässt sich der Zustand eines Stromkreises direkt manipulieren bzw. sichtbar machen. Häufig ist es jedoch etwas komplizierter. Es ist nicht nur ein Tastendruck zu erfassen, sondern möglicherweise sind Schall- oder Lichtsignale präzise zu registrieren. Zudem geht es meistens nicht nur darum, **ob** ein Reiz oder eine Aktivität – eine mechanische Berührung, ein Ton, Wärme, Licht etc. – vorhanden ist, sondern es geht auch um die Ausprägung – den Druck, die Tonhöhe und die Lautstärke, die Temperatur oder die Lichtintensität. Die Grundfrage lautet also: Wie lassen sich kontinuierliche Signale digitalisieren und aus digitalen Repräsentationen wieder kontinuierliche Signale generieren?

Beginnen wir bei der Wahrnehmung bzw. der Eingabe. Mit einem Schalter lässt sich feststellen, ob eine Taste gedrückt wurde, ob eine Tür geschlossen ist oder ob der maximale Wasserstand in einer Waschmaschine erreicht ist. Wollen wir zwischen mehr als zwei Ausprägungen unterscheiden können, z. B. die Höhe des Wasserstands erfassen, dann bedarf es einer Messvorrichtung, die das jeweilige physikalische Signal in elektrischen Strom

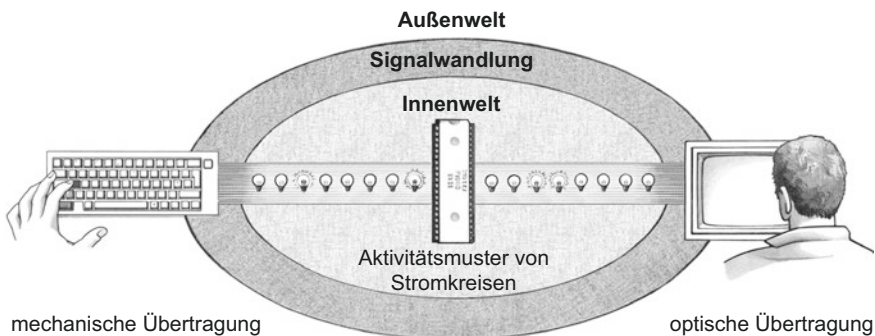


Abb. 4.8 Ein- und Ausgabegeräte vermitteln zwischen der Außen- und der Innenwelt eines Computers. (© Susanne Staubli, Eckart Zitzler)

umsetzen kann; beispielsweise einen Distanzsensor, der per Ultraschall einen Abstand in eine entsprechende elektrische Spannung übersetzen kann. Dieses kontinuierliche elektrische Signal muss wiederum in ein digitales Muster, also ein Aktivitätsmuster von Stromkreisen, die nur die Zustände „an“ und „aus“ annehmen können, transformiert werden.

Das Prinzip der Umwandlung ist simpel. Wir benutzen Spannungsfühler, die anzeigen, ob die Spannung unter oder über einem wohldefinierten Niveau liegt; indem wir mehrere von diesen Bausteinen mit verschiedenen Niveaus einsetzen, können wir Spannungsstufen detektieren und diese dann binär codieren. Abb. 4.9 skizziert das Prinzip unter der Annahme, dass wir zwei verschiedenartige Relais haben, von denen eines bei niedriger Spannung und eines bei hoher Spannung schaltet. Im rechten Stromkreis sei jeweils das zu transformierende kontinuierliche Signal eingespeist. Die zwei linken Stromkreise stellen dieses Signal digital dar, und zwar über vier mögliche Abstufungen, nämlich „keine Spannung“, „niedrige Spannung“, „mittlere Spannung“ und „hohe Spannung“.

Das Ganze erinnert an die Pegelanzeigen von Musikanlagen aus den 1980er-Jahren, bei denen mehrere Lämpchen in einer Kette nebeneinander angeordnet waren und Lichtstreifen erzeugten, die den Pegel des Tonsignals wiedergaben. Letztlich wird dabei das Signal gerastert, also in Bereiche unterteilt. Das Eingabegerät nimmt nur gewisse Abstufungen des kontinuierlichen Signals wahr. Klar, je mehr von den Spannungsfühlern eingebaut sind, desto feiner die Rasterung, doch wie bei jeder Rasterung gilt:

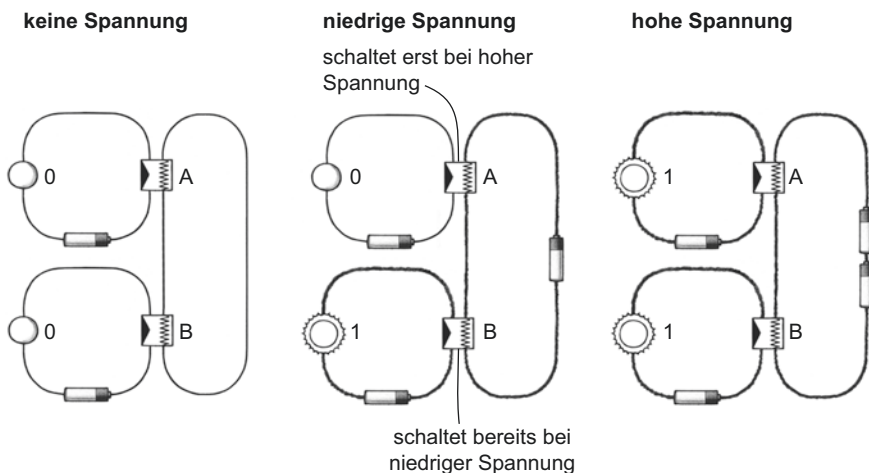


Abb. 4.9 Wie eine analoge Größe, hier eine Spannung, in ein digitales Muster transformiert werden kann. (© Susanne Staubli, Eckart Zitzler)

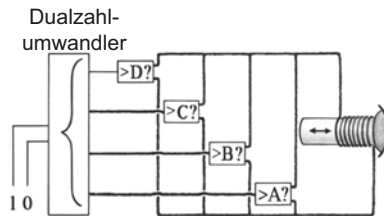
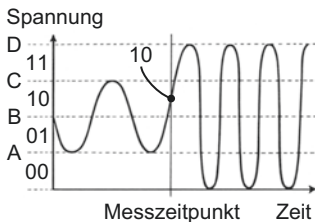
Es gehen Informationen verloren. Häufig ist das nicht weiter tragisch, weil die Abstufungen über das Differenzierungsvermögen der menschlichen Sinnesorgane hinausgehen, z. B. beim Hören von Musik. Zudem kommt eine weitere Dimension hinzu, wenn wir den zeitlichen Verlauf eines Signals erfassen wollen. Ein Ton beispielsweise kann sich über die Zeit verändern, und damit ein Computer den Tonverlauf erfassen kann, wird auch auf der Zeitachse gerastert. Bsp. 4.1 zeigt auf, wie Computer Töne wahrnehmen und wie die Stufen intern als Dualzahlen codiert werden können.

Beispiel 4.1: Wie Schall in elektrische Signale umgewandelt wird

Sowohl bei einem Computer als auch bei einem Menschen wird Schall in zwei Schritten in eine interne Repräsentation übersetzt. Beim Computer funktioniert das so, wie es in Abb. 4.10 in der oberen Hälfte visualisiert ist.

In einem ersten Schritt werden die Schallwellen – mittels einer feinen Membran aufgefangen, an deren Ende eine Spule befestigt ist. Diese Spule wiederum ummantelt einen Permanentmagneten, ist dabei aber frei beweglich. Wird nun die Membran in Schwingungen versetzt, so bewegt sich auch entsprechend die Spule im vom

Computer



Mensch

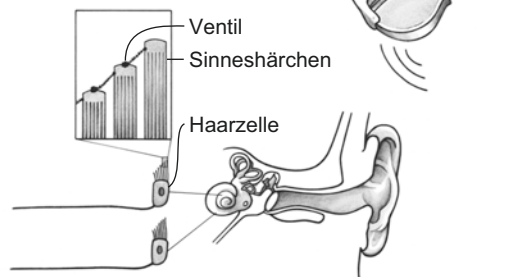
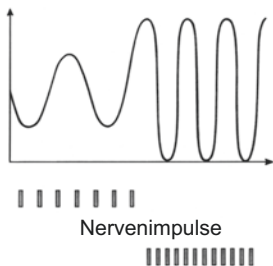


Abb. 4.10 Wie akustische Signale in digitale Muster transformiert werden – beim Computer (oben) und beim Menschen (unten). (© Susanne Staubli, Eckart Zitzler)

Magneten erzeugten Magnetfeld. Dadurch wird im Draht ein Strom erzeugt, und je stärker die Bewegung, desto größer die Spannung.

Im zweiten Schritt gilt es, die erzeugte elektrische Spannung in eine digitale Repräsentation zu überführen. Dazu wird das Spannungsspektrum in mehrere gleich große Teile unterteilt und diese Teile werden binär durchnummeriert (siehe Abb. 4.10 links oben). Für jeden Spannungsbereich, die Werte A bis D in der grafischen Darstellung, wird nun ein Spannungsfühler konstruiert, der genau dann eine 1 (Strom an) an seinem Ausgang erzeugt, wenn die gemessene Eingangsspannung **über** dem eingestellten Referenzwert (A bis D) liegt. Wir schließen also diese Spannungsmesser an den Draht und brauchen jetzt nur noch eine Schaltung, die die Ausgaben der Spannungsmesser einliest, und daraus die entsprechende Dualzahl für den Bereich zu ermitteln. Letzteres lässt sich leicht mittels der in Kap. 2 beschriebenen Methoden bewerkstelligen. Voilà! Und da ein Geräusch ja noch eine zeitliche Dimension hat, müssen wir den Ton in regelmäßigen Abständen abgreifen und erhalten somit eine Folge von Dualzahlen als Repräsentation des Geräuschs. Dabei gehen natürlich gewisse Aspekte durch die Diskretisierung verloren.

Das ist übrigens auch beim Menschen so (siehe Abb. 4.10 unten). Wir nehmen Schallwellen über Haarzellen wahr, die mit aus vielen kleinen Sinneshärcchen bestehenden Haarkämmen ausgestattet sind, welche beim Auftreffen von Schallwellen gebeugt werden. Durch die Neigung werden bestimmte Poren geöffnet, durch die positiv geladene Teilchen in das Innere der Haarzelle strömen und diese aufladen. Überschreitet das Erregungsniveau der Haarzelle eine gewisse Schwelle, entsteht ein elektrischer Nervenimpuls, der zu genau definierten Stellen im Gehirn, wo die Signale verarbeitet werden, weitergeleitet wird. Die Intensität eines Reizes wird durch die Häufigkeit widerspiegelt, wie oft die Haarzelle „feuert“; das nennt man Frequenzcodierung.

Die Transformation eines kontinuierlichen bzw. analogen Signals in ein digitales Signal nennt sich Analog-Digital-Wandlung; häufig werden dabei gleich mehrere Signale parallel erfasst, z. B. sind in einer Digitalkamera unzählige Lichtsensoren nebeneinander verbaut. Das Pendant ist die Digital-Analog-Wandlung, die ein Aktivitätsmuster von mehreren Stromkreisen in eine entsprechende elektrische Spannung übersetzt. Um anzuzeigen, dass eine computergesteuerte Waschmaschine läuft, reicht natürlich wiederum ein Lämpchen. Soll hingegen ein Automotor reguliert werden, dann gibt es nicht nur „kein Gas“ und „Vollgas“, sondern verschiedene Abstufungen. Das Problem der Umwandlung lässt sich ähnlich wie bei der Wahrnehmung lösen, nur dass anstatt verschiedener Spannungsfühler verschiedene Spannungsquellen kombiniert werden. In Abb. 4.11 sehen Sie ein beispielhaftes Szenario, bei dem zwei verschieden starke Spannungsquellen über Relais in Serie geschaltet werden können. Je nachdem, welche Relais aktiviert werden, können ein, zwei oder drei Batterien hintereinander aktiviert

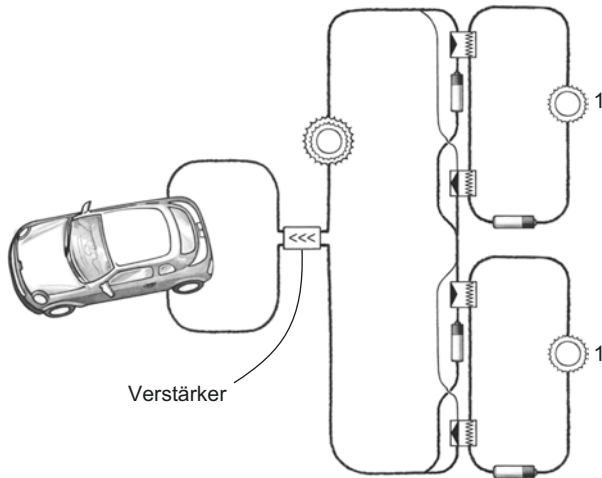


Abb. 4.11 Wie ein digitales Signal (in Form einer Dualzahl) in eine kontinuierliche Größe (z. B. eine Spannung) transformiert wird. (© Susanne Staubli, Eckart Zitzler)

sein oder es liegt überhaupt keine Spannung an. Und auch hier gilt: Nicht alle Spannungen sind möglich, nur bestimmte Spannungsstufen können erzeugt werden.

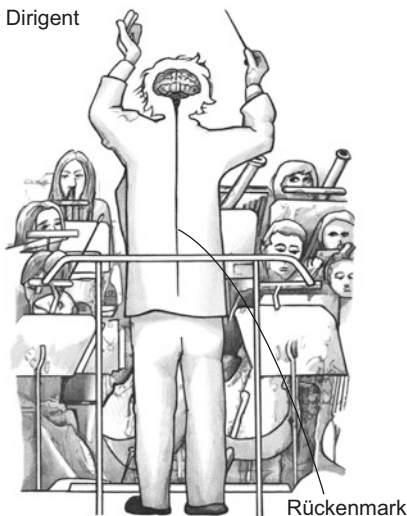
Analog-Digital- und Digital-Analog-Wandler sind die zentralen Elemente, um einen Computer mit der Außenwelt zu koppeln. Sie spielen vor allem bei eingebetteten Systemen eine große Rolle, bei denen komplexe Interaktionen mit der Umwelt bewerkstelligt werden müssen. Entsprechend groß ist das Spektrum an Ein- und Ausgabegeräten, man spricht auch von Sensoren und Aktoren. Sie bilden häufig die Möglichkeiten des Menschen nach, seien es nun Neigungssensoren, Temperatursensoren, Abstandssensoren sowie Mikrofone oder Lautsprecher und Motoren. Interessant ist, dass das Prinzip der Signalwandlung beim Menschen ganz ähnlich auftritt, denn auch hier müssen physikalische Signale in die interne Repräsentation, d. h. Nervenimpulse, umgewandelt werden bzw. Nervenimpulse müssen wiederum übersetzt werden in Bewegungen oder anderweitige Reaktionen. Sinneszellen generieren aufgrund von Umweltreizen – wir hatten exemplarisch die für das Hören zuständigen Haarzellen angeschaut – Nervenimpulse, sie digitalisieren das äußere Signal, und die Reizintensität wird über die Häufigkeit der Feuerungen codiert. Das Gegenstück zur Sinneszelle ist die Muskelzelle: Sie weist längliche Elemente in ihrem Innern auf, die sich auf ein Signal hin (durch einen Nervenimpuls übertragen) zusammenziehen können.

Ein Rückgrat für alle

Damit hätten wir den Prozessor sowie die Ein- und Ausgabegeräte. Hinzu kommen noch die Speicherkomponenten: einerseits der Hauptspeicher, der als Fächer von Speicherzellen organisiert ist, die über eindeutige Nummern – Adressen – identifiziert werden; andererseits nichtflüchtige Speicher für große Datenmengen, z. B. Festplatten, bei denen ebenfalls Speicherbereiche über Adressen ansprechbar sind. Jetzt geht es darum, diese Komponenten so miteinander zu verbinden, dass daraus ein Gesamtsystem wird. Die Rede ist vom bereits erwähnten Bus, jenem Bündel an Leitungen, das als Datenautobahn fungiert und worüber der Prozessor mit den anderen Funktionseinheiten kommuniziert. Der Bus – der Begriff ist vermutlich vom lateinischen *omnibus* („für alle“) abgeleitet – ist das Rückgrat eines Computersystems, er verbindet quasi Sinnesorgane und Muskulatur mit dem Gehirn (siehe Abb. 4.12). Der Bus selbst unterteilt sich nochmals in Daten- und Kontrollleitungen, letztere werden zur Abwicklung der Kommunikation genutzt. Zudem kann es zusätzlich zu den Datenleitungen noch separate Adressleitungen geben, über die Speicheradressen übermittelt werden können.

Orchester

Dirigent



Computer

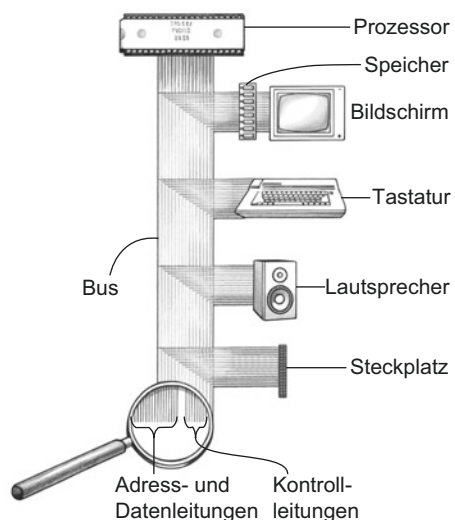


Abb. 4.12 Was beim Menschen das Rückenmark ist, ist beim Computer der Bus.
(© Susanne Staubli, Eckart Zitzler)

Wenn natürlich alle Funktionseinheiten am selben Leitungsbündel hängen, muss klar geregelt sein, wer gerade sprechen darf und wie die Kommunikation abläuft. Das funktioniert wie in einem Orchester, denn der Prozessor ist der Dirigent, der die Einsätze verteilt, von ihm geht der Datenaustausch aus. Um die Geräte anzusprechen, gibt es eindeutige Kennungen. Zu Beginn einer Transaktion sendet der Prozessor die Kennung der anzusprechenden Funktionseinheit über den Bus. Zwar empfangen alle Geräte die Kennung, doch nur das angesprochene Gerät reagiert und steigt in die Kommunikation ein. Der Datenaustausch selbst läuft nach einem strengen Muster ab, das in einem Protokoll – wie bei einem Staatsbesuch – festgelegt ist: entweder gemäß einem vorgegebenen Zeitraster, das nennt man synchron, oder durch gegenseitige Bestätigungen, was als asynchron bezeichnet wird (Anm. 4.4). Der Prozessor gibt vor, ob Daten zum Gerät geschickt oder vom Gerät empfangen werden sollen.

Anmerkung 4.4: Mit oder ohne Takt

Eine Bustransaktion – so nennt man den Vorgang, wenn ein Datenwort oder -block über den Bus vom Prozessor zu einer anderen Komponente oder umgekehrt übertragen wird – erfordert die Abstimmung der Kommunikationspartner untereinander. In Abb. 4.13 sehen Sie zwei Möglichkeiten, wie so eine Bustransaktion ablaufen kann für den Fall, dass der Prozessor Daten vom Speicher anfordert; es sind jeweils die Signale auf verschiedenen Busleitungen angedeutet. Der Prozessor teilt dem Speicher jeweils über eine spezielle Kontrollleitung mit, dass er eine Anfrage stellt, während er die Adresse der Speicherzelle auf die Adressleitungen legt. Der Speicher stellt die Daten auf den Datenleitungen des Busses bereit und kann den Auftrag über eine weitere Kontrollleitung quittieren.

Bei einer synchronen Übertragung stimmen sich beide Funktionseinheiten über einen gemeinsamen Taktgeber ab und die Kommunikation ist durchgetaktet. Das könnte so aussehen, dass der Speicher genau drei Zeitsignale nach dem Empfang der Adresse die Daten auf den Bus legt und die Bustransaktion

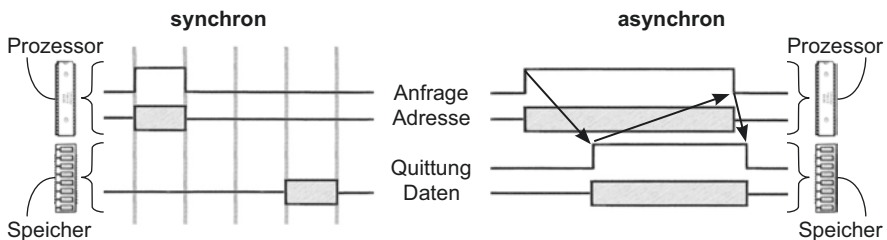


Abb. 4.13 Synchroner und asynchroner Bustransaktion. (© Susanne Staubli, Eckart Zitzler)

nach dem nächsten Zeitsignal abgeschlossen ist. In diesem Fall muss der Speicher die Bereitstellung der Daten nicht über eine separate Kontrollleitung anzeigen. Bei der asynchronen Übertragung gibt es kein starres Zeitraster. Stattdessen müssen sich die beiden Kommunikationspartner gegenseitig jeden einzelnen Transaktionsschritt quittieren und so versichern, dass sie sich verstanden haben – das nennt man dann Handshaking-Protokoll. Der Prozessor stellt die Anfrage, der Speicher signalisiert die Verfügbarkeit der Daten, der Prozessor zeigt durch das Zurücknehmen der Anfrage, dass er die Daten empfangen hat, und zum Schluss quittiert der Speicher mit der Zurücknahme seines Signals, dass die Transaktion abgeschlossen ist. Dieses Protokoll ist zwar aufwändiger, doch sehr nützlich, wenn zwei Geräte miteinander kommunizieren, die unterschiedlich schnell sind, oder wenn längere Distanzen zu überwinden sind und die Einhaltung eines gemeinsamen Zeitrasters schwierig ist.

In der Regel liegt jedem Bussystem ein anderes Protokoll zugrunde, das sich unterscheiden kann in den Hardware-Spezifikationen wie Spannungspegel oder Übertragungsgeschwindigkeit und im konkreten Ablauf der Kommunikation. Schnelle Busse haben zumeist viele kurze Leitungen und arbeiten synchron, während langsame Busse für gewöhnlich asynchron funktionieren und nur wenige Leitungen umfassen, die dafür aber wesentlich länger sein dürfen. Bussysteme sind standardisiert, sodass verschiedene Hersteller Geräte bauen, die dann problemlos an einen entsprechenden Bus angeschlossen werden können. So bleibt ein Computersystem erweiterbar. Diese Erweiterbarkeit ist der große Vorteil eines Busses, gleichzeitig ist damit aber auch ein Nachteil verbunden: Weil alle Geräte die gleichen Kommunikationsleitungen verwenden, bedarf es eines großen Koordinationsaufwands. Wären die Geräte direkt miteinander verbunden, ginge die Kommunikation wesentlich schneller. Deswegen wird der Bus häufig auch mit einem Flaschenhals verglichen, einem Engpass. Bsp. 4.2 illustriert dieses Phänomen und zeigt auf, was alles passieren muss, damit nach einem Tastendruck der entsprechende Buchstabe auf dem Bildschirm erscheint.

Beispiel 4.2: Vom Tastendruck zur Bildschirmanzeige

In Abb. 4.14 sind exemplarisch die Schritte skizziert, die bei einem der ersten Desktop-Computer notwendig waren, um einen Tastendruck zu registrieren und den entsprechenden Buchstaben auf dem Bildschirm darzustellen. Im ersten Schritt drückt die Benutzerin die Taste A, dabei wird ein Stromkreis geschlossen und dieses Signal von einem speziellen Baustein, der die Tastatur mit dem Bus verbindet, entgegengenommen. Dieser Tastaturbaustein ermittelt den zur Taste passenden achtstelligen Binärcode und speichert diesen lokal, bis er abgeholt wird. Der Prozessor überprüft regelmäßig, ob eine Taste

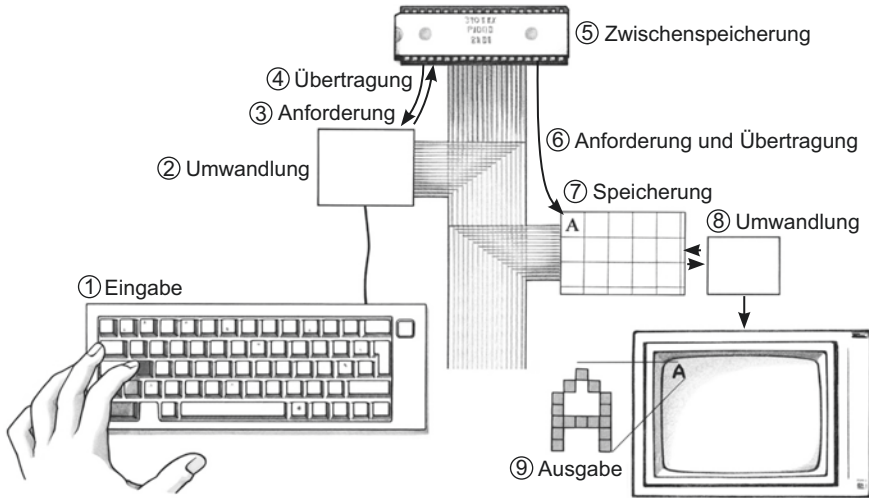


Abb. 4.14 Schritte vom Tastendruck bis zur Bildschirmanzeige am Beispiel eines frühen Desktop-Computers. (© Susanne Staubli, Eckart Zitzler)

gedrückt wurde. Daher legt er die Kennung des Tastaturbausteins auf den Bus und signalisiert über die Kontrollleitungen eine Anfrage (Schritt 3). Im nächsten Taktzyklus reagiert der Tastaturbaustein darauf, legt den Binärcode des von der Benutzerin gewählten Buchstabens auf den Bus, und der Prozessor liest ihn unmittelbar ein. Jetzt befindet sich der Buchstabencode im Prozessor. Für die Ausgabe schreibt der Prozessor den Buchstaben in eine Region im Speicher, die die Bildschirmausgabe enthält (Schritte 6 und 7). Ein Videowandler liest diese Speicherregion permanent aus – er hat eine direkte Verbindung zum Speicher – und wandelt den Buchstabencode schließlich in ein Punkteraster um, das auf dem Bildschirm dargestellt wird.

Leitungsbündel, die als Datenautobahnen fungieren, gibt es natürlich genauso beim Menschen. Nerven sind nichts anderes als Bündel von Axonen zur Informationsübertragung. Einen riesigen Bus stellt das Rückenmark dar, daneben gibt es noch zwölf Nervenstränge, die Hirnnerven, die das Gehirn mit dem Rest des Körpers verbinden. Über sie werden Sinneswahrnehmungen entgegengenommen und Bewegungen gesteuert. Im Gegensatz zu den Hirnnerven ist das Rückenmark nicht einfach nur ein Leitungsbündel, sondern es enthält auch selbst Nervenzellen und übernimmt Steuerungsfunktionen. Wenn wir die Busse unseres Nervensystems dem Bus eines Computers gegenüberstellen, gibt es jedoch einen entscheidenden Unterschied: Bei einem Computer-Bus werden die Datenleitungen mehrfach benutzt, und das in doppelter Hinsicht. Einmal kann der Datentransfer

in beide Richtungen ablaufen, also beispielsweise vom Prozessor zum Speicher und genauso anders herum; des Weiteren werden die einzelnen Leitungen eines Busses von allen angeschlossenen Geräten gemeinsam benutzt und geteilt. Nervenzellen dagegen kennen nur eine Richtung der Informationsübertragung, nämlich von den Dendriten zum Axon. Auch die Mehrfachnutzung einer Leitung gibt es im Nervensystem nicht; es existieren nur spezifische Wege, die beispielsweise ein Sinnesorgan mit bestimmten Regionen im Gehirn verbinden. So weiß das Gehirn immer, von welcher Körperstelle der Reiz ausgeht, und so können wir ihn zuordnen. Insofern hat ein Bus fast mehr mit unserem Gefäßsystem und dem Blutkreislauf zu tun. Denn auch Letzterer erfüllt die Funktion eines Busses, da durch ihn Botenstoffe, die Hormone, im Körper verteilt werden; für verschiedene Organe und Gewebe gibt es unterschiedliche Hormone, sodass tatsächlich nur die spezifischen Körperteile angesprochen werden.

Ein System, um den Betrieb zu meistern

Jetzt haben wir die Hardware-Komponenten eines Computersystems beisammen. Damit sind die Voraussetzungen geschaffen, das Ganze in Betrieb zu nehmen, d. h. ein Programm ausführen. Zumindest theoretisch, denn praktisch gilt es noch ein Problem zu lösen: Wie kommt das Programm eigentlich in den Hauptspeicher? Natürlich, wir können einen Teil des Hauptspeichers fix so verdrahten, dass dort der Programmcode abrufbar ist. Doch wenn wir an einen Wechsel des Programms denken, ist das erst die halbe Miete, es bedarf noch eines Tricks. Dieser besteht darin, dass das erste Programm, das beim Starten des Computers ausgeführt wird, die Aufgabe übernimmt, anderweitige Programme zu laden. Bei den ersten Computern hieß das, einen Lochkartenleser anzusteuern und die auf Karton festgehaltenen Programmdateien in den Hauptspeicher zu kopieren. Später wurden dann Programme von Massenspeichern wie Festplatten geladen und hinzu kam die Möglichkeit, Programme gezielt auszuwählen. Das sind die Anfänge des Betriebssystems, eines Programmsystems, das einen Computer erst nutzbar macht und seinen Betrieb sicherstellt.

Eine zentrale Aufgabe des Betriebssystems besteht darin, mit der Benutzerin zu kommunizieren und ihr die Möglichkeit zu geben, Programme einzulesen, zu starten und zu beenden. Wenn wir für den Moment davon ausgehen, dass der Prozessor immer nur ein Programm ausführen kann, dann heißt das letztlich: Das Betriebssystem muss ein Programm von einem externen Speichermedium in den Hauptspeicher

laden, dann zu dem entsprechenden Programm verzweigen und zum Schluss wieder die Kontrolle übernehmen, indem das ausgeführte Programm erneut zum Betriebssystem verzweigt. Damit verbunden ist die Möglichkeit, Programme (und anderweitige Daten wie Texte, Bilder usw.) als Dateien auf einem Massenspeicher strukturiert abzulegen – über ein Dateisystem, das vom Betriebssystem zur Verfügung gestellt und verwaltet wird. Dem Betriebssystem kommt jedoch nicht nur die Aufgabe zu, die Hardware für den Menschen nutzbar zu machen, sondern auch für Programme. Dafür stellt es Programmteile zur Verfügung, mit denen der Prozessor die verschiedenen Komponenten des Computersystems auf einfache Art und Weise ansprechen kann. Diese Programmfragmente können als Unterprogramme von anderen Programmen genutzt werden. Bsp. 4.3 erklärt, wie das im Detail funktioniert.

Beispiel 4.3: Wie ein Unterprogrammaufruf funktioniert

Um zu studieren, wie ein Programm ein bestehendes Programmfragment nutzen kann, betrachten wir ein einfaches Szenario, bei dem ein Computersystem in einem Auto eingesetzt wird (Abb. 4.15). Das Betriebssystem verfügt über ein Programmfragment, mit dem Distanzsensoren angesprochen und die entsprechenden Messwerte ermittelt werden können. Will nun ein Programm, z. B. zum Einparken, dieses Programmfragment als Unterprogramm nutzen, legt es zuerst die Nummer des auszulesenden Distanzsensors in einen vordefinierten Speicherbereich ab und verzweigt dann mittels eines speziellen Befehls (`call`) an den Anfang des Programmfragments; dabei merkt sich der Prozessor die Adresse des nachfolgenden Befehls. Das Unterprogramm wird ausgeführt und legt zum Schluss den Messwert des gewählten Sensors ebenfalls in dem vordefinierten Speicherbereich ab; mit dem Befehl `return` übergibt es die Kontrolle zurück an das aufrufende Programm und der Prozessor setzt die Programmausführung an der gemerkten Adresse fort. Das aufrufende Programm liest schließlich das Ergebnis des Unterprogrammaufrufs aus dem gemeinsamen Datenbereich aus.

Das Betriebssystem lässt sich vergleichen mit dem vegetativen Nervensystem des Menschen, das vor allem das Rückenmark und den Hirnstamm umfasst und grundlegende Körperfunktionen – Herzschlag, Atmung, Verdauung u. a. – kontrolliert und steuert. Es stellt sicher, dass das innere Körpermilieu stimmt; schließlich müssen die Systeme, die für die Versorgung und die Entsorgung zuständig sind, koordiniert und angepasst werden, wenn sich die Bedingungen ändern. Das vegetative Nervensystem arbeitet autonom und entzieht sich (größtenteils) unserem Willen. Zwar können wir unseren Herzschlag über Entspannungsübungen indirekt beeinflussen, aber direkt

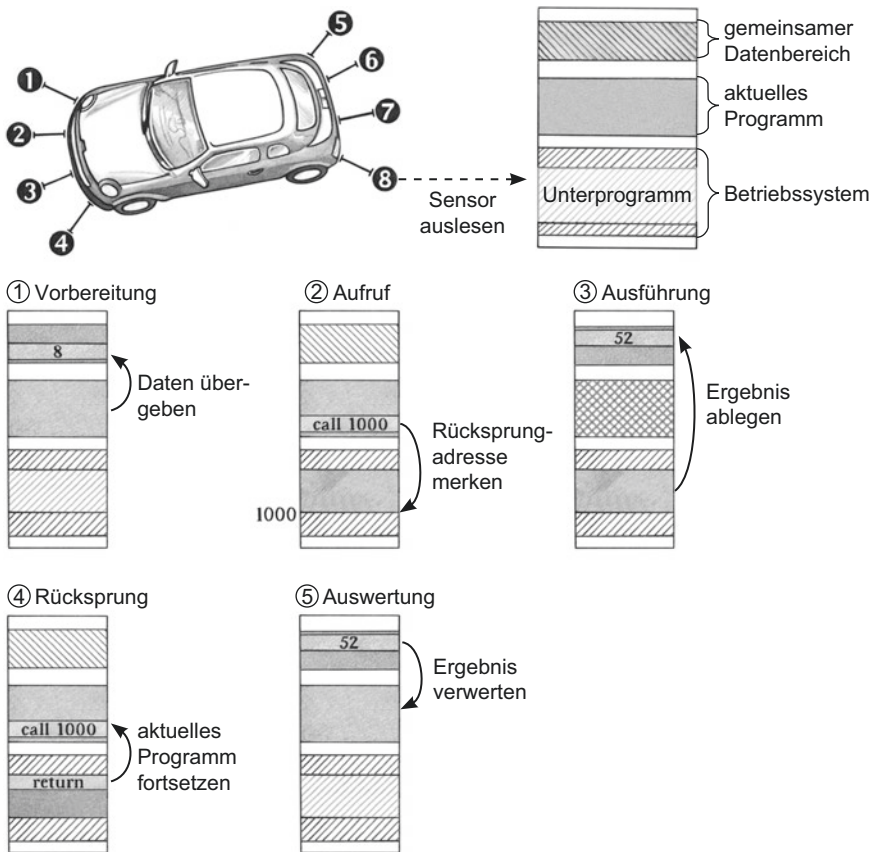


Abb. 4.15 Schematischer Ablauf eines Unterprogrammaufrufs. (© Susanne Staubli, Eckart Zitzler)

kontrollieren können wir ihn nicht. Das vegetative Nervensystem besteht selbst aus zwei Gegenspielern: dem Sympathikus und dem Parasympathikus. Ersterer ist für die Aktivierung zuständig und mobilisiert Energien, Letzterer reduziert die Körperaktivitäten und somit den Energieverbrauch – Antreiben versus Ausruhen.

Schneller sein: jeder Augenblick zählt

Das Grundgerüst eines Computers steht nun, doch in dieser reinen Form wurde es eigentlich nur bei den ersten Computersystemen eingesetzt. Schon bald begann man die Von-Neumann-Architektur zu optimieren

und an ihr herumzufeuern – schließlich kann ein Computer nicht schnell genug sein. Im Folgenden kann ich Ihnen nur beispielhaft einen Eindruck davon vermitteln, wie Computer-Tuning funktioniert. Beginnen wir beim Bus, denn er stellt häufig das Nadelöhr und den limitierenden Faktor eines Computersystems dar.

Die Relativität der Zeit

Jeder Datenaustausch zwischen zwei Funktionseinheiten eines Computers läuft über den Bus. Aus diesem Grund versucht man, den Bus so schnell wie möglich zu betreiben, muss dabei aber immer mit physikalischen, technologischen und ökonomischen Grenzen umgehen. Generell gilt: Je mehr Leitungen vorhanden sind und je kürzer diese sind, desto schneller ist auch der Bus. Doch kurze Verbindungen lassen sich nur auf der Hauptplatine, z. B. zwischen Prozessor und Speicher realisieren. Was ist, wenn wir externe Geräte anschließen wollen, die weit weg platziert werden, möglicherweise ein Windmesser auf dem Hausdach? Bei einem langen Leitungsbündel vermindert sich die Übertragungsgeschwindigkeit radikal, weil die Signale verzögert werden und das bei der zeitlichen Synchronisierung berücksichtigt werden muss. Kurz und schnell oder lang und langsam, das ist das Dilemma.

Die Lösung ist naheliegend, aber in der konkreten Realisierung recht kompliziert: Wir verwenden einfach mehrere Busse mit verschiedenen Geschwindigkeiten. Dort, wo hohe Geschwindigkeiten gefragt sind, z. B. zwischen Prozessor und Speicher, kommen schnelle synchrone Busse mit vielen parallelen Datenleitungen zum Einsatz. Dort, wo viele Geräte über längere Distanzen angeschlossen werden sollen, werden asynchrone Busse mit wenigen Leitungen verwendet. Damit der Prozessor trotzdem noch mit jedem Gerät kommunizieren kann, müssen die Busse aufwändig mit speziellen Geräten, quasi Dolmetschern, miteinander verbunden werden. Bsp. 4.4 zeigt auf, wie so ein Szenario aussehen kann. Letztlich handelt es sich um eine Hierarchie von Bussen, in der der schnellste die Basis bildet und langsamere Busse durch Brückengeräte in jeweils einen schnelleren Bus eingehängt werden. Tatsächlich gibt es je nach Anwendungszweck und Anbieter ganz verschiedene Bussysteme.

Beispiel 4.4: Eine Hierarchie von Bussen

In Abb. 4.16 sehen Sie die Systemarchitektur eines Desktop-Computers, der den rund 20 Jahre jüngeren Nachfolger des in Abb. 4.5 gezeigten Modells verkörpert. Der Prozessor ist mittels eines schnellen Busses mit einem Baustein verbunden, der die Verbindung zu allen weiteren Geräten übernimmt. Von diesem Baustein,

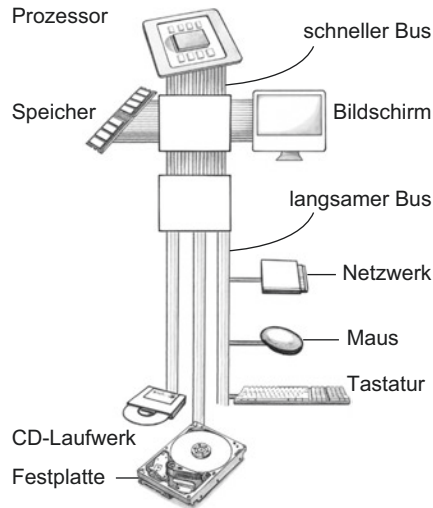


Abb. 4.16 Ein hierarchisches Bussystem. (© Susanne Staubli, Eckart Zitzler)

Northbridge genannt, gehen drei schnelle Busse ab, nämlich zum Speicher, zum Grafikprozessor und zu einem weiteren Baustein, der Southbridge, an den weitere langsamere Busse angeschlossen sind. Insgesamt gehen drei Busse mit verschiedenen Geschwindigkeiten von der Southbridge ab, sowohl interne für Massenspeicher als auch externe, an die sich mittels Kabel und Stecker externe Geräte an den Computer anschließen lassen. Sie sehen: Hier wurde viel Aufwand betrieben, um das Bussystem möglichst effizient zu gestalten.

Der Vorteil der hierarchischen Buskopplung liegt darin begründet, dass der Prozessor mit jeder anderen Funktionseinheit so schnell kommunizieren kann, wie es diese erlaubt. Allerdings setzt das eine perfekte Abstimmung zwischen den verschiedenen schnellen Bussystemen voraus, denn Signale müssen ja möglicherweise Busse passieren, bevor sie am Endgerät ankommen. Den Dolmetscher-Geräten kommt also eine wichtige Aufgabe zu: die Kopplung verschiedener Zeitsysteme. Werden z. B. synchrone Busse verbunden, müssen die entsprechenden Taktsignale aufeinander abgestimmt werden; dazu werden sie in der Regel von einem zentralen Taktgeber abgeleitet.

Auch im Menschen gibt es so einen zentralen Taktgeber, der im Gehirn sitzt – es ist eine Nervenansammlung im Gehirn mit ungefähr 20.000 Neuronen – und sich suprachiasmatischer Kern nennt. Dieses Hirnareal ist an die Sehnerven gekoppelt und kann somit erfassen, wie stark der Lichteinfall derzeit ist. Auf dieser Basis werden alle Uhren im Körper synchronisiert. Denn jede Zelle besitzt quasi eine Uhr, die einen Rhythmus von 22–28 h hat

und über eine Art Ebbe und Flut von im Wechselspiel stehenden Proteinen betrieben wird. Der Mensch besitzt folglich mehr als eine Billion Uhren in sich und alle sind aufeinander abgestimmt über den suprachiasmatischen Kern. Der Körper arbeitet also ebenfalls mit mehreren Rhythmen, von denen der Tag-und-Nacht-Rhythmus der augenscheinlichste ist.

Vom aktiven zum passiven Warten

Eine andere Möglichkeit, das Bussystem zu optimieren, besteht darin, die Anzahl der Bustransaktionen zu minimieren und so den Prozessor zu entlasten. Da jede Bustransaktion vom Prozessor ausgeht, muss dieser auch sicherstellen, dass die Daten rechtzeitig von den Funktionseinheiten abgeholt werden. Ob beispielsweise eine Taste gedrückt wurde, erfährt er nur, indem er beim Tastaturbaustein nachfragt (vgl. Bsp. 4.2). Dieses aktive Warten auf Eingaben durch Nachfragen nennt sich Polling. Da der zeitliche Abstand zwischen einem Tastendruck und dem nächsten in den Zeitdimensionen des Prozessors einer Ewigkeit gleichkommt, gehen dafür unnötig Rechenkapazitäten verloren, die der Prozessor besser für die aktuelle Berechnung nutzen könnte. Denken Sie nur an ein Action-Computerspiel, bei dem permanent aufwändige Animationen durchgeführt werden, während die Spielerin in sehr unregelmäßigen Abständen Kommandos gibt. Aus diesem Grund hat man sich eine Variante ausgedacht, bei der Ein- und Ausgabegeräte aktiv auf den Prozessor zugehen und ihm signalisieren können: „Hallo, ich hab da was!“ In der Fachsprache heißt diese Variante Interrupt oder manchmal auch Exception; es handelt sich um die Möglichkeit, den Prozessor zu unterbrechen.

Wie funktioniert das? Eine Unterbrechung muss primär in der Prozessor-Hardware behandelt werden (der Prozessor aus Kap. 3 ist dafür nicht ausgerichtet, ließe sich aber einfach dahingehend erweitern). Ein Gerät meldet über spezielle Kontrollleitungen an den Prozessor, dass es seine Aufmerksamkeit benötigt. Der Prozessor muss nun die Abarbeitung des aktuellen Programms unterbrechen und ein anderes Programm ausführen, das die Unterbrechung behandelt und die entsprechenden Schritte ausführt. Im Fall des Tastaturbausteins muss der Code des eingegebenen Zeichens vom Prozessor über den Bus vom Tastaturbaustein angefordert und eingelesen werden. Und sobald die Unterbrechung behandelt wurde, muss der Prozessor wieder an der gleichen Stelle fortfahren, wo er unterbrochen wurde. Letztlich passiert hier nichts anderes als bei einem Unterprogrammaufruf: Das aktuelle Programm wird für einen Moment verlassen und anschließend wieder an der vorherigen Stelle fortgesetzt. Bsp. 4.5 zeigt Ihnen, wie das im Detail funktioniert.

Beispiel 4.5: Wie Unterbrechungen behandelt werden

Abb. 4.17 visualisiert auf der rechten Seite die Schritte, die beim Eintreffen eines Interrupts abgearbeitet werden, am Beispiel eines Tastendrucks. Zunächst wird der laufende Befehl noch vollständig ausgeführt. Dann wird der aktuelle Wert des Befehlszählers zusammen mit dem Grund der Unterbrechung, codiert als Dualzahl, gesichert (Schritt 4). Anschließend muss der Prozessor ermitteln, welches Programm für diese Art von Unterbrechung zuständig ist. Dafür gibt es an einer fixierten Adresse im Speicher eine Tabelle, in der für jede Art von Interrupt die Anfangsadresse eines Programms abgelegt wird. Der Prozessor schaut also in der Tabelle nach, ermittelt die Programmadresse und verzweigt anschließend in die Unterbrechungsbehandlungsroutine, die für die Tastatur zuständig ist. Diese holt nun die als Dualzahl codierte Kennung der gedrückten Taste über eine Bustransaktion vom Tastaturbaustein ab und sichert sie im Speicher, damit sie später verwendet werden kann. Dabei muss sichergestellt sein, dass die Inhalte der verwendeten Register zwischengespeichert werden, schließlich soll das unterbrochene Programm später weiterlaufen, als wäre nichts gewesen. Entweder kann das vom Prozessor automatisch bewerkstelligt werden, oder die Unterbrechungsbehandlungsroutine gewährleistet dies. Zum Schluss sind die Registerinhalte wiederhergestellt und der Prozessor setzt die Ausführung des unterbrochenen Programms fort.

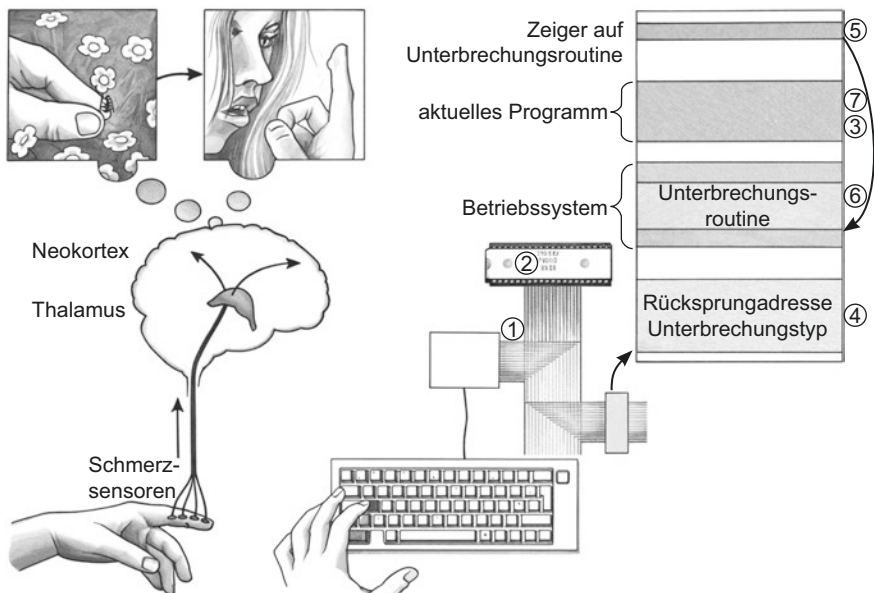


Abb. 4.17 Wie Unterbrechungen behandelt werden – beim Menschen (links) und beim Computer (rechts). (© Susanne Staubli, Eckart Zitzler)

Natürlich können auch beim Computer mehrere Geräte gleichzeitig einen Interrupt anfordern, und auch dafür gibt es eine Lösung: Die Anfragen sind priorisiert, manche Geräte werden anderen gegenüber bevorzugt. Und Interrupts lassen sich ausschalten – maskieren nennt man das. Wenn z. B. gerade eine Unterbrechung behandelt wird, dann sollte sie nicht noch einmal durch die gleiche Unterbrechung unterbrochen werden. Zudem werden Unterbrechungen auch verwendet, um unvorhergesehene Situationen zu behandeln – etwa wenn der Prozessor beispielsweise die nächste Instruktion holt und dann feststellen muss, dass es für die Dualzahl, die er decodieren will, keinen entsprechenden Befehl gibt, oder wenn er eine Division durchführt und der Divisor eine Null ist. Es gibt also diverse Gründe, und jeder davon muss ggf. anders behandelt werden.

Beim Menschen können wir in dem Zusammenhang von Aufmerksamkeit sprechen. Wir nehmen ständig Sinneseindrücke auf, unsere Sinneszellen feuern permanent und versorgen unser Gehirn mit Informationen über die Außen- und die Innenwelt. Das meiste davon blenden wir jedoch aus. Pflücken wir eine Blume, werden wir vermutlich in dem Moment nicht bewusst wahrnehmen, dass unsere Füße den Boden berühren. Sticht uns hingegen die Wespe, die wir beim Ergreifen des Halms übersehen haben, dann gehört der Wunde augenblicklich unsere volle Aufmerksamkeit. Unser aktueller Fokus kann also von dringlichen Signalen unterbrochen werden, woraufhin sich unsere Aufmerksamkeit – kurz oder länger – verschiebt. Unterbrechungen werden von unserem Zwischenhirn vorverarbeitet (siehe Abb. 4.17 links). Es nimmt die diversen Sinnesreize entgegen und entscheidet dann situativ, was davon überhaupt in welcher Intensität an das Großhirn weitergeleitet wird und in unsere Aufmerksamkeit gelangt, sprich uns bewusst wird. Denn einerseits können Unterbrechungen unterdrückt werden, beispielsweise wenn wir in eine Sache vertieft sind und gar nicht mitbekommen, dass uns jemand gerufen hat. Andererseits können auch mehrere Signale gleichzeitig eintreffen. Das Zwischenhirn entscheidet in diesem Fall, was wichtiger ist; beispielsweise übertönt ein Schmerzsignal ein Antippsignal. Das ist eine sehr geschickte Vorrichtung, denn unsere gesamte Umgebung aufmerksam zu beobachten, ist uns nur für eine beschränkte Zeit möglich.

Die Last auf mehrere Schultern verteilen

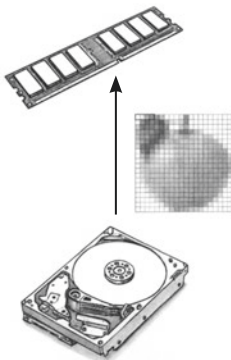
Eine weitere Möglichkeit, den Prozessor zu entlasten, besteht darin, ihm Aufgaben abzunehmen. Die Idee ist, dass er sich nicht um jede

Bustransaktion selber kümmern muss, sondern bestimmte Datentransfers ohne sein Zutun abgewickelt werden können. Einigen Geräten wird also erlaubt, direkt über den Bus mit dem Speicher zu kommunizieren – der Prozessor kann in der Zwischenzeit das aktuelle Programm abarbeiten. Das macht vor allem dann Sinn, wenn große Daten zu übertragen sind, z. B. wenn ein Foto vom Massenspeicher zur Bearbeitung in den Hauptspeicher zu laden ist. Diese Situation ist in Abb. 4.18 skizziert: Während der Prozessor fortwährend über den Bus Programminstruktionen aus dem Speicher lädt, werden die Bilddaten von der Festplatte in den Hauptspeicher transferiert. Damit handeln wir uns natürlich schon wieder das nächste Problem ein. Denn was ist, wenn der Prozessor und die Festplatte gleichzeitig auf den Speicher zugreifen und somit den Bus benutzen wollen? Auch hierfür gibt es Lösungen, Anm. 4.5 erläutert die Hintergründe. Übrigens nennt man das Prinzip in der Fachsprache DMA, was für *direct memory access*, also direkten Speicherzugriff steht.

Anmerkung 4.5: Busmeister werden

In der Praxis wird das Konzept des direkten Speicherzugriffs mittels eines speziellen Geräts, einem DMA-Controller, bewerkstelligt, das dann die Aufgabe des Prozessors übernimmt und die Daten von einem Gerät zum Speicher oder umgekehrt transferiert. Der Prozessor instruiert den DMA-Controller, von welchem Gerät Daten vom oder in den Speicher transferiert werden sollen, und er spezifiziert die Datenmenge und die Adresse, wo die Daten abgelegt werden sollen. Anschließend wird der DMA-Controller tätig und handelt als Mittler zwischen Gerät und Speicher; der Prozessor wird erst wieder per Interrupt informiert, wenn der Datentransfer abgeschlossen ist.

Datentransfer



Programmausführung

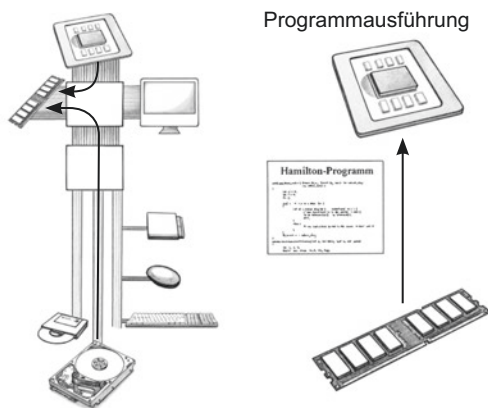


Abb. 4.18 Wie zwei Bustransaktionen gleichzeitig ablaufen können. (© Susanne Staubli, Eckart Zitzler)

Damit sich Prozessor und DMA-Controller – es kann sogar mehrere DMA-Controller geben – nicht ins Gehege kommen, bedarf es der Absprache. Eine Möglichkeit besteht darin, dass die Funktionseinheiten, die den Bus bedienen dürfen, über eine Leitung in einer Kette angeordnet sind, einer sogenannten *daisy chain*, zu Deutsch Gänseblümchenkette. Pro Bustransaktion wird eine Erlaubnis in die daisy chain gegeben. Die erste Einheit in der Kette erhält zuerst die Erlaubnis, den Bus benutzen zu dürfen, ein Gerät gibt die Erlaubnis an das nachfolgende Gerät in der Kette weiter, insofern es den Bus gerade nicht benötigt. Nur die Funktionseinheit mit der aktuellen Erlaubnis, darf über den Bus verfügen, die anderen verhalten sich still und warten, bis sie an der Reihe sind.

Spitzfindige werden vielleicht bemerken, dass es ein Problem gibt, wenn der Prozessor ständig Instruktionen aus dem Hauptspeicher holen muss und gleichzeitig ein DMA-Controller den Bus benutzt. Um das Problem zu entschärfen, sind Prozessoren mit einer Art Zwischenspeicher ausgestattet, sodass immer gleich mehrere Befehle vom Hauptspeicher in den Prozessor transferiert werden können und so der Bus entlastet wird.

Die Idee, den Prozessor bei der Kommunikation zu umgehen, ist übrigens auch in der Natur etwas sehr Nützliches. Wenn Sie schon einmal versehentlich Ihre Hand in eine Flamme oder auf eine heiße Herdplatte gehalten haben, so werden Sie bemerkt haben, dass Ihre Hand sich schon wegzog, noch bevor Sie überhaupt bewusst realisierten, was passiert war. Sie haben nämlich reflexartig reagiert. Sie kennen so etwas auch vom Knie, wenn der Arzt mit einem kleinen Hämmerchen unterhalb der Kniescheibe Ihr Bein stimuliert und der Unterschenkel entsprechend ausschlägt – auch ein Reflex (Abb. 4.19). Solche Reflexe werden unmittelbar im Rückenmark ausgelöst. Die Schmerzsensoren in der Hand senden ihre Schmerzsignale an das Rückenmark und dort wird die Rückzugsbewegung direkt ausgelöst, indem Nervenimpulse an die entsprechenden Muskeln gesendet werden. Das Gehirn ist gar nicht involviert in diesen Prozess, auch wenn das Schmerzsignal selbst an das Gehirn weitergeleitet und dort dann zu Bewusstsein gebracht wird. Wir hatten vorhin

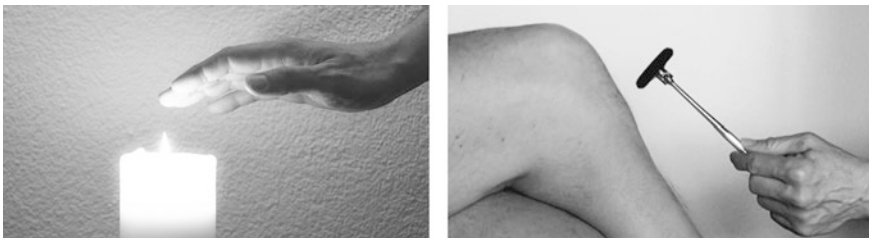


Abb. 4.19 Rückzieh- und Kniereflex. (© Judith Büsser)

schon gesehen: Im Rückenmark sitzen auch Nervenzellen, es dient nicht nur als Datenautobahn; diese verarbeiten die Schmerzsignale und können Bewegungen initiieren.

Es gibt noch verschiedene weitere Möglichkeiten, dem Prozessor Verarbeitungsaufgaben abzunehmen. Beispielsweise können Grafikprozessoren – im Englischen als *graphical processing unit* (GPU) bezeichnet, analog zu *central processing unit* (CPU) – den Prozessor bei der Bildschirmausgabe erheblich entlasten. Sie können beispielsweise 3D-Bilder oder Schattenwürfe berechnen und sich auf solche Aufgaben spezialisieren. Grafikprozessoren erhalten vom Prozessor quasi eine Beschreibung der Bildschirmausgabe und berechnen daraus das gerasterte Gesamtbild, das für jeden Bildpunkt auf dem Bildschirm den Helligkeitswert angibt. Gerade im Bereich der Computerspiele sind hoch performante Grafikprozessoren für aufwändige grafische Darstellungen sehr wichtig. Landschaften aus verschiedenen Blickwinkeln auf dem Bildschirm anzeigen lassen, ist z. B. so eine daten- und rechenintensive Aufgabe, die den Prozessor stark in Anspruch nehmen kann, je nach Detaillierungsgrad der Darstellung. Was liegt also näher, als gewisse Aufgaben an andere Komponenten zu delegieren? Die Natur macht es uns ja vor (siehe Anm. 4.6).

Anmerkung 4.6: Optische Täuschungen

Beim Auge ist das Problem der Datenflut elegant gelöst. Unser Sehwerkzeug besitzt in seinem Inneren ca. 12 Mio. Sinneszellen für das Erkennen von hell und dunkel sowie 6 Mio. Sinneszellen für das farbliche Sehen. Insgesamt müssten also 18 Mio. Nervenfasern zu den Bilder verarbeitenden Regionen im Gehirn führen, effektiv bestehen die Sehnerven jedoch nur aus je 1 Mio. Nervenfasern. Was passiert da? Im Auge wird bereits eine Vorverarbeitung vorgenommen, indem die Signale von mehreren Sinneszellen gebündelt und mittels Bipolar- und Ganglienzellen, spezialisierten Nervenzellen im Auge, zu verdichteten Signalen umgewandelt werden. Dadurch verringert sich natürlich die Auflösung, doch es passiert etwas Entscheidendes dabei: Kontraste werden hervorgehoben – etwas, was wir im Laufe unsere Entwicklungsgeschichte immer wieder gebraucht haben, z. B. um die Umrisse eines Raubtiers zu erkennen. Und gleichzeitig wird so die Datenflut enorm reduziert, ansonsten wäre unser Gehirn ständig damit beschäftigt, unsere visuellen Reize zu verarbeiten. Diese Kontrastverstärkung kommt übrigens bei optischen Täuschungen zum Ausdruck. Testen Sie es einmal und schauen auf Abb. 4.20 links. Die inneren Rechtecke sind alle gleich hell, doch je nach Umgebung werden sie stärker hervorgehoben, sodass das rechte z. B. heller wirkt als das linke. Auf der rechten Seite von Abb. 4.20 sehen Sie ein weiteres Beispiel für eine optische Täuschung, bei der Rechtecke vor dem Hintergrund der Kreise leicht verbogen erscheinen, obwohl ihre Seiten perfekt gradlinig sind.

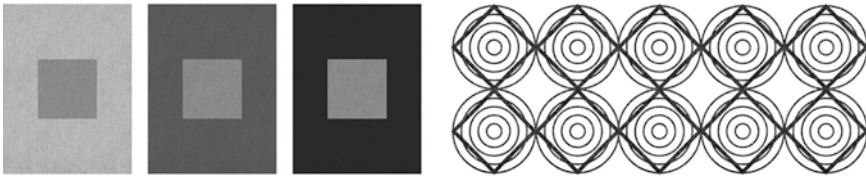


Abb. 4.20 Beispiele für optische Täuschungen. (© Susanne Staubli, Eckart Zitzler)

Grafikprozessoren werden häufig auch für andere Anwendungen zweckentfremdet, z. B. für das Hochleistungsrechnen. Als im Jahr 2016 zum ersten Mal ein Computerprogramm im Brettspiel Go einen menschlichen Profispieler schlug, kamen Grafikprozessoren zum Einsatz, um das Spielprogramm zu trainieren. Solche aufwändigen Berechnungen lassen sich vor allem dann massiv beschleunigen, wenn viele Grafikprozessoren miteinander kombiniert werden – im Go-Beispiel waren es 50. Die gleiche Frage stellt sich natürlich auch bei den gewöhnlichen Prozessoren: Warum nicht einfach mehrere von ihnen einbauen, um die Geschwindigkeit des Computersystems zu erhöhen?

Mehr ist nicht immer mehr

In der Regel kommen in Computern mehrere Prozessoren parallel zum Einsatz, wie ich das bereits in Kap. 3 angedeutet hatte. Häufig sind mehrere Zentraleinheiten, also mehrere Kopien einer (riesigen) Prozessorschaltung auf einem Chip untergebracht, und dieser Chip präsentiert sich nach außen wie ein Prozessor mit einem Busanschluss, an den die internen Prozessorschaltungen angeschlossen sind. Man nennt jede Prozessorschaltung auf dem Chip auch Core, also Kern, und einen Prozessor mit mehreren Kernen Multicore-Prozessor. Diese Bezeichnung ist übrigens analog zum Gehirn, dort spricht man auch von Kernen, wenn man eine bestimmte Ansammlung von Nervenzellkörpern bezeichnen will, die innerhalb des zentralen Nervensystems liegt. Darüber hinaus gibt es auch die Möglichkeit, effektiv mehrere vollwertige Prozessoren in einem Computersystem zu verwenden, wobei jeder von ihnen an den Systembus angeschlossen ist. Und selbstverständlich lassen sich beide Varianten auch kombinieren, indem man mehrere Multicore-Prozessoren verbaut.

Die Frage ist allerdings, wie sich die zusätzliche Rechenleistung produktiv nutzen lässt. Ein Programm wird nur dann schneller, wenn der

Berechnungsvorgang auch auf mehrere Prozessoren verteilt, also parallelisiert werden kann. Wenn zwei Prozessorkerne zur Verfügung stehen, heißt das folglich nicht automatisch, dass ein Programm doppelt so schnell abgearbeitet wird wie auf einem Ein-Prozessor-System. Über die Aufteilung der Verarbeitungsaufgaben auf mehrere Prozesse muss man sich bereits beim Entwurf des Programms Gedanken machen. Bei manchen Anwendungen ist das einfacher als bei anderen, nicht alles lässt sich parallelisieren. Bsp. 4.6 zeigt, wie sich Parallelverarbeitung gewinnbringend beim Schachspielen einsetzen lässt.

Beispiel 4.6: Schachzüge berechnen mit mehreren Prozessoren

Betrachten wir ein Schachprogramm, das die Aufgabe hat, aus einer gegebenen Spielsituation heraus einen möglichst vielversprechenden Spielzug für eine Partei zu berechnen. Das prinzipielle Vorgehen ist dabei ähnlich wie bei menschlichen Spielern. Es werden verschiedene mögliche Spielzüge betrachtet und für die wird antizipiert, was der Gegner wohl tun wird, welche Spielzüge sich daraufhin ergeben usw. Mit anderen Worten: Es werden Spielzüge durchprobiert, allerdings können aufgrund der schier unendlichen Anzahl an Möglichkeiten immer nur wenige Schritte „gedanklich“ durchgespielt werden. Die Spielzüge werden schließlich bewertet und der unmittelbare Zug, der zur vielversprechendsten Spielsituation geführt hat, wird vorgeschlagen.

Abb. 4.21 illustriert nun, wie sich so ein Schachprogramm einfach parallelisieren lässt, und zwar auf der Basis zweier Prozessorkerne. Der erste Prozessorkern ermittelt zunächst eine bestimmte Anzahl von Zügen, die näher untersucht werden sollen. Anschließend werden diese Züge gleichmäßig auf die zwei Prozessorkerne verteilt. Jeder Prozessorkern spinnt jeden Zug weiter in die Zukunft und bewertet ihn. Anschließend sammelt der erste Prozessorkern die Ergebnisse, wertet sie aus und wählt auf dieser Basis den vielversprechendsten Zug. In diesem Szenario lassen sich sogar wesentlich mehr Prozessorkerne einsetzen, womit die Zugbewertung weiter beschleunigt werden kann.

Der Parallelisierung sind allerdings Grenzen gesetzt, selbst bei Anwendungen, die sich gut auf mehrere Prozesse verteilen lassen. Intuitiv würde man ja erwarten, dass mit zwei Prozessoren die Berechnungen nur halb so lange brauchen, bei vier Prozessoren nur ein Viertel der ursprünglichen Zeit in Anspruch nehmen usw. Doch die Beschleunigung ist viel geringer, und das ist bekannt als Amdahlsches Gesetz, benannt nach dem Computer-Ingenieur Gene Amdahl, der seine Überlegungen Ende der 1960er-Jahre vorstellte. Er definierte den Speedup, also die Beschleunigung, als den Faktor, um den die parallelisierte Programmausführung schneller ist als die Ein-Prozessor-Variante. Und er zeigte auf, dass der maximal

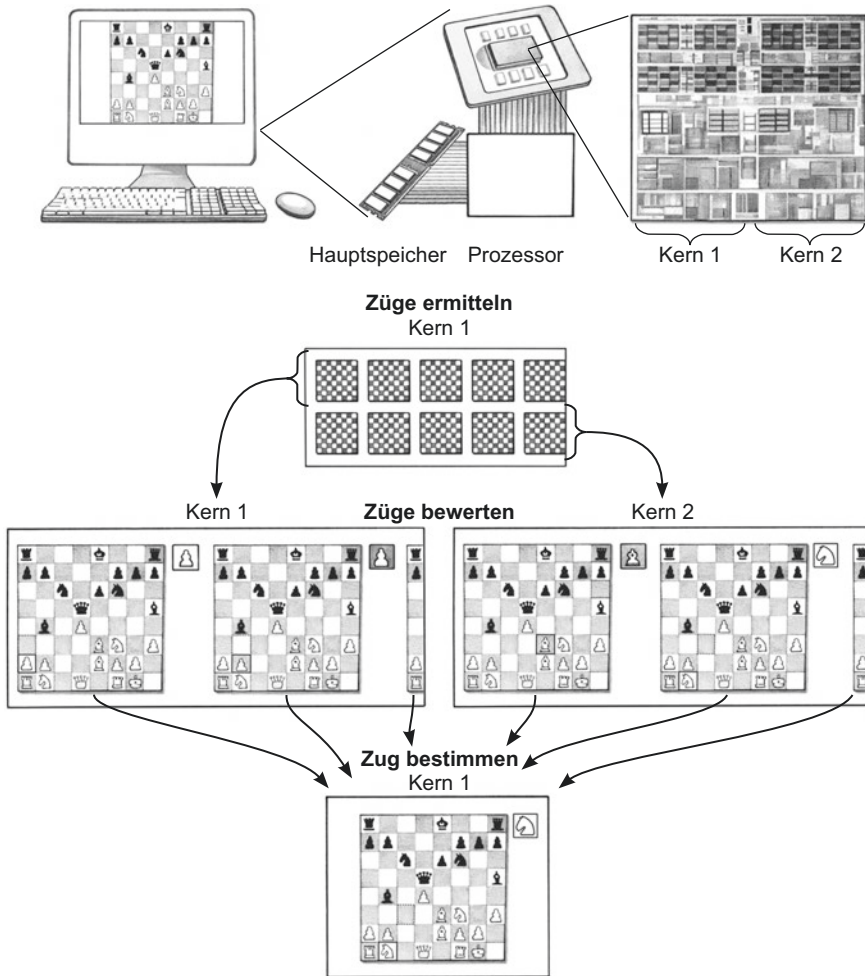


Abb. 4.21 Wie die Berechnung eines Schachzugs auf mehrere Prozessorkerne aufgeteilt werden kann. (© Susanne Staubli, Eckart Zitzler)

erreichbare Speedup nicht von der Prozessoranzahl abhängt, sondern von dem Anteil am Programm, der sich nicht parallelisieren lässt. Wenn dieser Anteil z. B. die Hälfte der Ausführungszeit auf einem einzelnen Prozessor ausmacht, dann wird die Gesamtausführungszeit, egal wie viele Prozessoren wir einsetzen, niemals diese Zeit unterschreiten können. Bsp. 4.7 demonstriert für die Schachspiel-Anwendung, wie die Anzahl der Prozessoren und der Speedup zusammenhängen.

Beispiel 4.7: Je mehr Prozessoren, desto schneller?

Betrachten wir noch einmal das Szenario in Abb. 4.21 und gehen davon aus, dass im ersten Schritt 1000 mögliche Züge ermittelt und diese im zweiten Schritt gleichmäßig auf die zur Verfügung stehenden Prozessorkerne zur weiteren Bewertung aufgeteilt werden. Des Weiteren nehmen wir an, dass die Programmteile zur Zugverteilung, zur Bewertung eines einzelnen Zugs und zur Auswahl des besten Zugs für sich genommen alle gleich viel Zeit beanspruchen, sagen wir 1 ms. Der nicht parallelisierbare Anteil des Programms beansprucht demnach 2 ms der Prozessorzeit.

Steht nur ein Prozessorkern zur Verfügung, so braucht das Programm insgesamt 1002 ms, bei zwei Prozessoren sind es 502 ms, bei zehn Prozessoren 102 ms, bei 200 Prozessoren 7 ms und bei 1000 Prozessoren 3 ms. Damit werden Speedups von ungefähr 2, 10, 143 und 334 erzielt. Wir sehen also, dass der Speedup mit der Anzahl der Prozessoren nicht proportional zunimmt, sondern viel langsamer wächst und sich dem Maximum von 334 annähert (mehr als 1000 Prozessoren können gleichzeitig nicht ausgenutzt werden).

Häufig ist es der Koordinations- und Kommunikationsaufwand, der limitierend für die Parallelverarbeitung wirkt, und zwar aus ganz praktischen Gründen. Sobald wir eine große Anzahl an Prozessoren verwenden wollen, bekommen wir ein Platzproblem. Einige Prozessoren müssen zwangsläufig weit weg von anderen platziert werden, womit auch die Kommunikationswege länger und verlangsamt werden. Skalierbarkeit ist hier das Stichwort, also die Frage, inwiefern wir durch die Erhöhung von Rechenressourcen auch einen proportionalen Zuwachs in der Rechenleistung erzielen. Und klar ist: Skalierbarkeit hat ihre Grenzen. Dieses Platzproblem spiegelt sich auch im Aufbau unseres Gehirns wider. Im evolutionär jüngsten Teil, der Großhirnrinde, sind die Körper der Nervenzellen außen in der Hirnmasse platziert; die Oberfläche ist extrem gefurcht, was die Gesamtfläche wiederum erhöht. Und im Innern sind es die Verdrahtungen, die knapp die Hälfte des Platzes ausmachen. Im Durchschnitt hat jeder Kubikmillimeter Großhirnrinde 10.000 Zellkörper und 1 Mrd. Synapsen; eine einzige Nervenzelle in der Großhirnrinde kann mit bis zu 10.000 anderen Nervenzellen gleichzeitig kommunizieren. Der Kommunikation bzw. Vernetzung kommt also eine tragende Rolle bei der Parallelverarbeitung zu – und das Gehirn arbeitet hochgradig parallel. Eine Zunahme an Rechenkapazitäten muss also stets auch den Koordinationsaufwand mitberücksichtigen.

Allgegenwärtig sein: mehrere Dinge gleichzeitig erledigen

Bislang sind wir davon ausgegangen, dass unser Computer genau ein Programm ausführen kann und dass mehrere Programme nur hintereinander abgearbeitet werden können. Natürlich, das ist Computersteinzeit. Damals war es tatsächlich so, dass ein Computer im Batch-Betrieb genutzt wurde, d. h., er arbeitete einen Stapel von Programmen eins nach dem anderen ab. Das änderte sich, als man vielen Benutzern den Zugang zu einem Computer ermöglichen wollte – noch vor der Zeit der Personal Computer. Man kam auf die Idee, nicht nur eine Tastatur und einen Bildschirm an eine Rechenmaschine anzuschließen, sondern mehrere, sodass viele Benutzer gleichzeitig mit dem Computer arbeiten konnten. Das Ganze setzte allerdings voraus, dass der Computer auch in der Lage war, mehrere Programme quasi gleichzeitig auszuführen und die Rechenzeit und sonstigen Ressourcen wie Speicherplatz unter mehreren Benutzern aufzuteilen. Heute ist es selbstverständlich, dass ein Computer viele Programme gleichzeitig ausführen kann.

Die Umsetzung scheint naheliegend: Wir erweitern die Hardware und verwenden kurzerhand mehrere Prozessoren oder Prozessorkerne. Dann kann jeder Prozessor ein Programm ausführen und die Programme können wirklich gleichzeitig ausgeführt werden. Das wird auch so gemacht, ist jedoch nur der halbe Weg zur Lösung des Problems. Denn die Anzahl der Prozessoren würde die maximale Anzahl von parallel laufenden Programmen begrenzen, bei acht Prozessoren wären es nur acht Programme. Das genügt für mich, können Sie einwenden. Fakt ist aber, dass auf einem Computer sehr viel mehr Programme gleichzeitig laufen, von vielen bekommen wir nur gar nichts mit. Und das ist bereits bei einem Computer möglich, der nur über einen einzigen Prozessor verfügt, wie wir gleich sehen werden. Der Weg führt über eine Software-Lösung: Wir erweitern das Betriebssystem, sodass es die Ausführung vieler Programme parallel ermöglicht.

Der Prozess oder die Illusion der Gleichzeitigkeit

Ein Prozessor, so haben wir in Kap. 3 gesehen, macht nichts anderes, als Befehle aus dem Hauptspeicher zu holen und die entsprechenden Aktionen umzusetzen; zu jedem Zeitpunkt wird genau ein Befehl abgearbeitet. Damit er trotzdem mehrere Anwendungen gleichzeitig bearbeiten kann – ich schreibe beispielsweise am Computer diesen Text und nutze ihn parallel, um

Musik zu hören –, bedarf es eines Tricks. Dieser besteht darin, zwischen den Programmen hin- und herzuschalten. Der Prozessor führt für eine gewisse Zeit das Textverarbeitungsprogramm aus und anschließend für eine gewisse Zeit das Musikabspielprogramm, immer im stetigen Wechsel. Erfolgen die Wechsel in genügend kurzen Zeitabständen, so merken wir die kurzen Unterbrechungen jedes Programms gar nicht. Genauso wie wir eine schnell präsentierte Abfolge von Bildern als bewegte Einheit wahrnehmen und nicht mehr als Kette von Einzelbildern – auf diesem Prinzip beruhten ja die ersten Filmprojektoren. Ähnlich ist es beim Hören (siehe Abb. 4.22): Wir können nur ungefähr 20 Signale pro Sekunde als eigenständige Geräuscheinheiten wahrnehmen, ansonsten erscheint uns das Ganze wie ein zusammenhängender Ton – oder zwei abwechselnde Töne erklingen auf einmal wie ein Akkord. Und da Prozessoren wesentlich schneller arbeiten, als unsere Sinnesorgane wie Augen und Ohren Veränderungen wahrnehmen können, funktioniert das Ganze.

Die Frage ist natürlich, wie konkret zwischen zwei Programmen gewechselt werden kann. Denn der Prozessor liest ja nur stumpf den nächsten Befehl ein, der unter der Speicheradresse abgelegt ist, auf die der Programmzähler verweist. Keine Rede von umschalten. Nun, wir haben ja bereits eine Variante kennengelernt, wie im Hintergrund Aufgaben erledigt werden können, ohne dass die Ausführung eines Programms davon beeinflusst wird. Die Rede ist von den Interrupts. Interrupts veranlassen den Prozessor, die Ausführung des aktuellen Programms zu unterbrechen, ein anderes Programm – die Unterbrechungsbehandlungsroutine – auszuführen und dann wieder beim aktuellen Programm fortzufahren. Und genau diese Idee nutzen wir auch, um zwischen Programmen umzuschalten. Ein Zeitgeber fungiert als Wecker und erzeugt nach einer bestimmten Periode, z. B. alle 10 ms, ein Signal, das eine Unterbrechung auslöst. Der Prozessor verzweigt daraufhin in eine Unterbrechungsbehandlungsroutine, die nichts anderes macht, als den Befehlszähler und die anderen Register

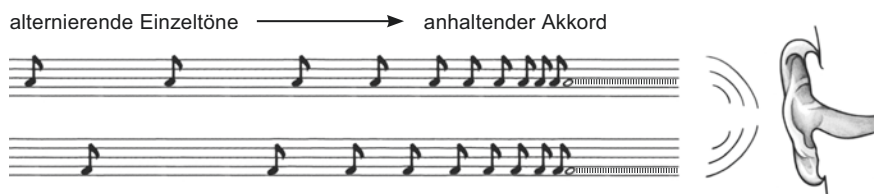


Abb. 4.22 Sich abwechselnde Töne werden bei kurzen Zeitabständen als Einheit wahrgenommen. (© Susanne Staubli, Eckart Zitzler)

so zu manipulieren, dass nach der Unterbrechung das andere Programm an der vorherigen Stelle fortgesetzt wird. Wie das im Einzelnen funktioniert, erläutert Bsp. 4.8.

Beispiel 4.8: Umschaltung zwischen Programmen

Bleiben wir beim Beispielszenario, in dem gleichzeitig ein Programm zur Textverarbeitung und ein Programm zum Abspielen von Musik laufen. In Abb. 4.23 sind drei Situationen abgebildet, in denen sich der Computer befinden kann, und zwar sehen Sie jeweils einen Ausschnitt des Hauptspeichers. Nehmen wir einmal an, der Prozessor befände sich in der Situation links, d. h., er führt aktuell das Textverarbeitungsprogramm aus. Dieses liegt an einer bestimmten Stelle im Speicher, der Programmzähler verweist auf den aktuellen Befehl. Nun erreicht das Wecksignal den Prozessor. Daraufhin wird zur Unterbrechungsbehandlungsroutine im Betriebssystem verzweigt; der Computer befindet sich in der zweiten Situation, die in der Mitte von Abb. 4.23 dargestellt ist. Dort werden nun der Programmzähler und die Inhalte der anderen Register so manipuliert, dass nach Beendigung der Unterbrechungsbehandlungsroutine das Musikabspielprogramm fortgesetzt wird beim aktuellen Befehl. Schließlich befindet sich der Prozessor für eine gewisse Zeit in der dritten Situation rechts, bis dann erneut ein Weck-Interrupt auftritt und wieder zum Textverarbeitungsprogramm umgeschaltet wird.

Selbstverständlich ergibt sich hier wiederum die gleiche Problematik, die ich bei den Interrupts angedeutet hatte. Wird ein Programm in seiner Ausführung unterbrochen, muss sich das Betriebssystem merken, an welcher Stelle im Programm die Ausführung gestoppt wurde; zudem müssen die Inhalte der Register zwischengespeichert werden. In der Fachsprache nennt man all die Daten, die beim Stopp einer Programmausführung gesichert werden müssen, den Kontext des Programms. Der Kontext stellt quasi die private Umgebung eines Programms dar und umfasst noch weitere Aspekte, beispielsweise aktuell benutzte Dateien. Und ein in Ausführung befindliches Programm – unterbrochen oder nicht – zusammen mit seinem Kontext nennt man einen Prozess. Auf jedem Computer läuft eine Vielzahl an Prozessen, und das Betriebssystem speichert all die relevanten Informationen zu den Prozessen in einer Tabelle, die bei jedem Prozesswechsel aktualisiert wird. Ja, selbst das Betriebssystem besteht aus vielen Prozessen, das macht die Architektur der Software viel einfacher.

Wenn wir das Ganze auf den Menschen projizieren wollen, dann könnten wir sagen, der Prozessor verlagert permanent seine Aufmerksamkeit. Aufmerksamkeit heißt, wir können unsere geistigen Ressourcen auf bestimmte Wahrnehmungen oder Gedanken oder andere Bewusstseinsinhalte

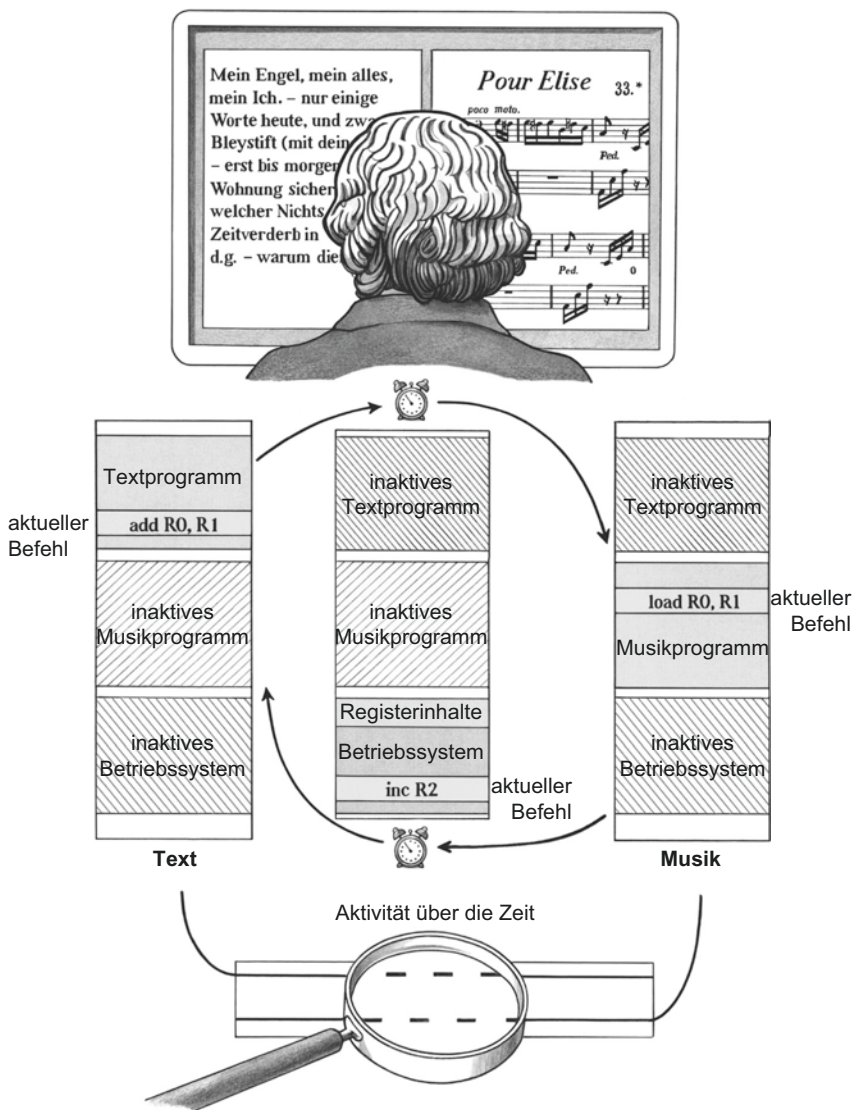


Abb. 4.23 Illustration, wie die Umschaltung zwischen zwei Programmen funktioniert. (© Susanne Staubli, Eckart Zitzler)

fokussieren und gleichzeitig vieles andere ausblenden. Und wir können unsere Aufmerksamkeit wechseln, z. B. von einer nach innen gerichteten zu einer nach außen gerichteten. Sie unterhalten sich, Ihre Aufmerksamkeit ist beim Gegenüber, beim Sehen, beim Zuhören. Dann schweift Ihre Aufmerksamkeit ab, Sie hängen einem Gedanken nach, der durch eine

Assoziation ausgelöst wurde. Und schließlich spüren Sie ein kleines Zwicken im Magen und Ihre Aufmerksamkeit richtet sich auf diese Wahrnehmung. Das geht also möglicherweise sehr schnell, aber wie schnell wir wechseln können, hängt davon ab, wie viel Konzentration, wie viele Ressourcen unsere Aufmerksamkeit für etwas Bestimmtes benötigt. In eine anspruchsvolle Aufgabe müssen wir uns vertiefen, wir brauchen eine gewisse Zeit, um reinzukommen; sind wir erst so richtig drin, erleben wir vielleicht sogar den „Flow“. Es fällt uns schwer, dann zu einer anderen Aufgabe zu wechseln, die ebenfalls hohe Konzentration erfordert. Offenbar müssen wir jedes Mal wieder den Kontext aufrufen, und das ist aufwändig. Sind die Unterbrechungen zu häufig, kann es sein, dass wir den ganzen Tag eigentlich nichts Gescheites auf die Reihe bekommen – wer kann davon nicht ein Lied singen?

Teilen ganz praktisch

Jetzt haben wir das Problem der quasi gleichzeitigen Ausführung mehrerer Programme gelöst, doch es stellt sich gleich das nächste: Wie verwalten wir die Ressourcen des Computersystems, also Speicher, Rechenzeit usw.? Laufen viele Prozesse parallel, müssen sie sich zwangsläufig die zur Verfügung stehenden Ressourcen teilen. Logisch, dass da Konflikte vorprogrammiert sind. Es ist die Aufgabe des Betriebssystems, hier das Chaos zu verhindern und ein geregeltes Miteinander zu gewährleisten. Manchmal kann es notwendig sein, einem Prozess den exklusiven Zugriff zu einer Ressource zu gewähren, z. B. zu einer drehbaren Kamera, mit der der Raum aktiv abgesucht werden kann. Hier gibt es dann entsprechende Methoden, um die Geräte anzufordern und freizugeben. Bei anderen Ressourcen muss ein enormer Verwaltungsaufwand betrieben werden, beispielsweise beim Speicher. Ein Prozess kann nicht einfach den Hauptspeicher in Beschlag nehmen, wie er möchte. Er muss vielmehr Speicherbereiche vom Betriebssystem anfordern und nicht benötigte wieder freigeben. Das Betriebssystem wiederum muss den Überblick haben, welche Regionen im Hauptspeicher frei sind und welcher Prozess welche Speicherbereiche alloziert hat. Je mehr Prozesse Speicheranforderungen und -freigaben vornehmen, desto zerstückelter wird der Speicher mit der Zeit – ein Phänomen, das man Fragmentierung nennt. Noch komplizierter wird es bei der zentralen Ressource Prozessorzeit. Das Betriebssystem muss permanent darüber entscheiden, welcher Prozess als Nächstes ausgeführt und wie viel Rechenzeit ihm zugestanden wird. Das Ganze nennt sich Scheduling, und hierfür gibt es meist vordefinierte Strategien, wie in Bsp. 4.9 erläutert wird.

Beispiel 4.9: Mit Prozessen jonglieren

Die einfachste Variante, die Prozessorzeit zwischen mehreren Prozessen aufzuteilen, beruht auf Gleichbehandlung: Jedem Prozess wird die gleiche Ausführungszeit zugestanden, die Abstände zwischen den Unterbrechungen sind immer gleich und die Prozesse kommen gemäß einer fixierten Reihenfolge zum Zug. Häufig ist diese Vorgehensweise ungünstig, weil sie unnötig Rechenzeit vergeudet. Vielleicht wartet ein Prozess auf den Tastendruck des Benutzers oder ein anderer Prozess ist blockiert, weil der Zugang zum Drucker noch nicht freigegeben ist? In solchen Situation bringt es nichts, dem jeweiligen Prozess Rechenzeit zuzugestehen, er wartet ja eh nur. Es gibt auch Situationen, bei denen gewisse Aufgaben dringlicher sind als andere. In einem Flugzeug erfordert die Wetterlage die Priorisierung der Prozesse, die für die stabile Fluglage zuständig sind; andere, für die Fluglage nicht relevante Prozesse können in diesem Moment in den Hintergrund treten. Generell ist diese Frage bei eingebetteten Systemen, die in Echtzeit reagieren müssen, viel drängender als bei unserem Computer zu Hause – dennoch, die Problematik ist dieselbe. Die Anforderungen an das Scheduling können sehr unterschiedlich sein, und entsprechend gibt es verschiedene Verfahren. Wie aus dem Leben gegriffen, erscheinen z. B. der ALAP-Algorithmus und der ASAP-Algorithmus, die eingesetzt werden, wenn Prozesse mit Fristen zur Erledigung versehen sind. ALAP steht für *as late as possible* und heißt, eine Aufgabe wird erst zum spätestmöglichen Zeitpunkt angepackt, das Gegenteil ist *as soon as possible*, also so früh wie möglich.

Scheduling heißt Jonglieren mit den zu bewältigenden Aufgaben, und das kennen wir ja selber bestens, das ist unser tägliches Programm. Gelegentlich können wir jeweils situativ entscheiden, was als Nächstes zu tun ist, doch anspruchsvolle Aufgaben, die aus vielen Teilaufgaben bestehen, bedürfen des vorausschauenden Ressourcenmanagements. Wir können nicht beliebig viel gleichzeitig erledigen, selbst wenn wir für viele Tätigkeiten durch Üben eine Routine entwickeln können. Neben dem Jonglieren ist Kochen ein gutes Beispiel (Abb. 4.24). Hier sind viele Tätigkeiten parallel auszuführen,



Abb. 4.24 Mehrere Aktivitäten unter einen Hut bekommen: beim Jonglieren, beim Kochen und bei der Bahn. (© Judith Büsser)

und wir bedienen uns häufig eines vorgegebenen Plans, Rezept genannt, um den Ablauf hinzubekommen. Wenn etwas anbrennt, müssen wir situativ die Aufmerksamkeit anpassen und auch die Ablaufplanung. Bei großen Systemen wie z. B. der Bahn braucht es große Ablaufpläne (Abb. 4.24 rechts) – und auch die müssen bei Unvorhergesehenem angepasst werden. Zudem ist es gar nicht so einfach, solche Fahrpläne zu erstellen, es müssen dabei äußerst komplexe Optimierungsaufgaben gelöst werden.

Zusammenarbeit will gelernt sein

Herausforderungen ergeben sich allerdings nicht nur dann, wenn sich Prozesse um die vorhandenen Ressourcen „streiten“, sondern auch wenn sie an gemeinsamen Aufgaben arbeiten und sich koordinieren müssen. Wenn wir z. B. an das Computersystem einer Bank denken, dann greifen da viele Prozesse ineinander und auf die gleichen Daten, sprich Konten, zu. Der Einfachheit halber nehmen wir an, es gäbe nur einen zentralen Computer, der die Konten verwaltet. Die Kontodaten, insbesondere der Kontostand, seien im Hauptspeicher abgelegt, und auf dem Computer liefen mehrere Prozesse: Ein Prozess registriert Auszahlungen am Automaten und aktualisiert den Kontostand entsprechend, ein anderer Prozess verarbeitet Einzahlungen. Dann kann es bei unglücklichen Umständen passieren, dass wie von Zauberhand Geld von Ihrem Konto verschwindet – unangenehm! Wenn nämlich die beiden Prozesse gleichzeitig auf den Kontostand zugreifen und ihn verändern wollen, kann es sein, dass einer der beiden auf einmal mit veralteten Daten operiert. Bsp. 4.10 erläutert die Situation im Detail.

Beispiel 4.10: Wenn Prozesse sich ins Gehege kommen

In Abb. 4.25 sehen Sie das Szenario grafisch dargestellt. In der Mitte ist der Hauptspeicher des Bankencomputers dargestellt mit dem Betriebssystem, dem Einzahlprozess, dem Auszahlungsprozess und den Kontodaten (für den Moment gehen wir von einem einzigen Konto aus). Links sehen Sie den Ablauf beim Einzahlungsprozess, rechts denjenigen beim Auszahlungsprozess. Nehmen wir nun an, beide Prozesse würden gleichzeitig auf das Konto zugreifen, weil eine Einzahlung von 2000 EUR eingeht und parallel eine Auszahlung über 100 EUR getätigt werden soll. Jetzt kommt es darauf an, wer zuerst dran ist. Angenommen der Auszahlungsprozess liest den Kontostand von 5000 EUR zuerst, wird jedoch unterbrochen, bevor er den Kontostand aktualisieren kann. Nun ist der Einzahlungsprozess an der Reihe, er liest den Kontostand und setzt ihn gemäß Einzahlung auf 7000 EUR hoch. Dann wird wieder umgeschaltet und der Auszahlungsprozess schließt seinen virtuellen Geldtransfer

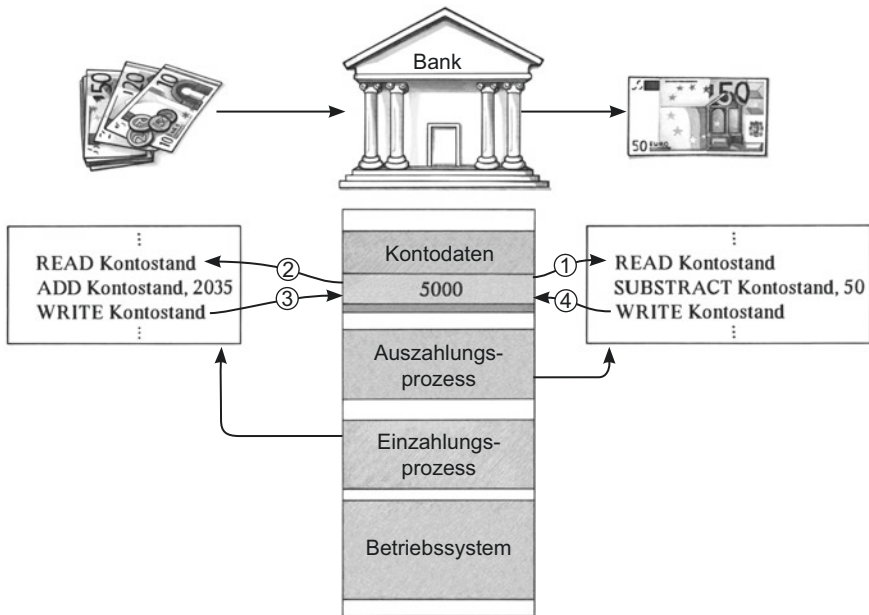


Abb. 4.25 Wenn zwei Prozesse auf die gleichen Daten zugreifen, kann es zu Inkonsistenzen kommen. (© Susanne Staubli, Eckart Zitzler)

ab: Es bleibt noch der Kontostand zu aktualisieren, nämlich auf 4900 EUR (er hatte ja den Stand von 5000 EUR gelesen). Danach lautet der Kontostand 4900 EUR und die Einzahlung ist ausgelöscht worden. Das konnte nur passieren, weil der Auszahlungsprozess zwischen dem Lesen und dem Aktualisieren des Kontostandes unterbrochen wurde.

Um diese Probleme zu umgehen, wurden verschiedene Methoden ausgeheckt. Die Grundidee ist letztlich immer dieselbe: Prozesse können in exklusive Phasen eintreten, in denen sich immer nur ein Prozess befinden kann. Sie fordern vom Betriebssystem Eintritt in diese Phase an, und entweder wird die gewährt, wenn kein Prozess bereits in der Phase ist, oder der Prozess wird so lange blockiert, d. h. schlafen gelegt, bis die Phase wieder freigegeben ist. Eine Phase ist wie ein Gerät: Sie erhält einen Namen, den alle beteiligten Prozesse kennen, und immer nur ein Prozess kann diese Phase reservieren. Bsp. 4.11 erläutert das Prinzip.

Beispiel 4.11: Realisierung des gegenseitigen Ausschlusses

In Abb. 4.26 ist das Bankenszenario noch einmal dargestellt, allerdings benutzen die beide Prozesse diesmal zwei Betriebssystemaufrufe, LOCK und UNLOCK, um den Zugriff auf die Kontodaten zu reservieren bzw. freizugeben. Diese Aufrufe funktionieren im Prinzip wie Unterprogrammaufrufe. Beide Prozesse versuchen zunächst über den LOCK-Aufruf, die Phase des Kontozugriffs für sich zu reservieren. Ist die Reservierung erfolgreich, kann der Prozess weiter ausgeführt werden, im anderen Fall wird der Prozess temporär deaktiviert, d. h., er bekommt so lange keine Prozessorzeit mehr zugewiesen, bis die Phase wieder freigegeben und die Reservierung durchgeführt werden kann. Nach dem LOCK-Aufruf wird in beiden Prozessen der Kontostand manipuliert und zum Schluss wird die Phase wieder mittels des UNLOCK-Aufrufs freigegeben. In Abb. 4.26 sehen Sie den Ablauf visualisiert, wenn erst der Auszahlungsprozess die Phase reserviert, dann den Kontostand liest und direkt im Anschluss der Einzahlungsprozess das Gleiche versucht. Der Einzahlungsprozess wird vom Betriebssystem deaktiviert, bis der Auszahlungsprozess den UNLOCK-Aufruf tätigt. Anschließend führt der Einzahlungsprozess seine Transaktion durch und gibt wiederum die Phase frei.

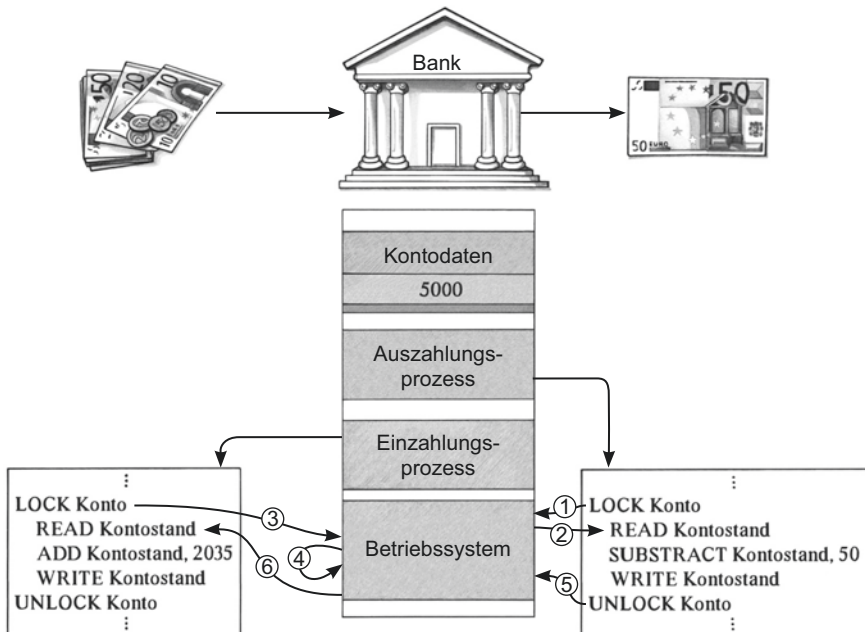


Abb. 4.26 Wie das Bankenproblem gelöst werden kann (© Susanne Staubli, Eckart Zitzler)

Leider gilt auch hier: Mit jedem Problem, das wir in den Griff bekommen, entsteht ein neues. Es kann nämlich die Situation auftreten, dass sich zwei Prozesse gegenseitig blockieren. Beispielsweise könnten zwei Prozesse gleichzeitig versuchen, eine Überweisung von einem Konto auf ein anderes vorzunehmen. Der eine Prozess fordert zunächst den exklusiven Zugriff auf das erste Konto an, der andere Prozess reserviert hingegen zuerst das zweite Konto. Und jetzt haben wir das Dilemma: Wenn nun einer der Prozesse den Zugriff auf das jeweils andere Konto erhalten möchte, wird er deaktiviert, weil ja der andere Prozess das andere Konto reserviert hält. Beide Prozesse stecken folglich fest, weil der andere im Besitz der angeforderten Ressource ist. Das nennt man einen Deadlock. Ein Deadlock ist das, was wir vom Autofahren kennen. Wenn an einer Kreuzung mit Rechts-vor-links-Vorfahrtsregelung gleichzeitig an jeder Zufahrt ein Auto ankommt, dann braucht es eine zusätzliche Verständigung per Gesten o. Ä., um die Blockadesituation aufzulösen. Bei Computersystemen versucht man, diese von vorneherein zu vermeiden. Bei obigem Beispiel könnte die Lösung so aussehen, dass als Phase der Exklusivzugriff auf beide Kontos definiert wird. Aber so einfach sind die Situationen in der Regel nicht vorzusehen. Deswegen spielt man die Prozesskommunikation häufig in Computersimulationen durch, um potenzielle Deadlocks zu identifizieren.

In unserem Alltagsleben würden wir so einen Deadlock wohl als Teufelskreis beschreiben. Ja, auch unsere Aufmerksamkeit kann sich in so einem Teufelskreis befinden. Vielleicht müssen wir eine wichtige Präsentation halten, gleichzeitig hindert uns eine diffuse Angst daran, vor Leuten zu stehen. Um die Angst zu verlieren, müssten wir gute Erfahrungen machen, dass es nicht so schlimm ist und vielleicht sogar Spaß machen kann. Wir sollten die Angst in neuen Situationen relativieren können, aber genau diese Angst hindert uns daran. Ja, was nun: Erfahrungen machen oder Angst verlieren? Das Ganze blockiert uns. Und dann braucht es häufig ein Zurücktreten, den Blick von außen, damit wir einen ganz anderen Schritt machen und den Konflikt auflösen können.

One Program To Rule Them All

Zum Schluss werfen wir noch einmal einen Blick auf das Betriebssystem. Mit Einführung der Prozesse hat es eine große Anzahl neuer Aufgaben bekommen. Anfangs war es noch primär Dienstleister und seine Hauptaufgabe bestand darin, die Hardware verfügbar zu machen. Mittlerweile ist es Auftraggeber und Herrscher, denn das Betriebssystem

ist an sich das Programm, welches alle anderen Abläufe und Programme steuert. Dies ist der Grund, warum das Betriebssystem mit der Zeit immer wichtiger wurde: Es ist das Software-Pendant zum Prozessor auf der Hardware-Seite. Die Aufgaben, die es heutzutage wahrnimmt, sind äußerst vielschichtig. Seine Kernaufgaben umfassen die Initialisierung der Hardware nach dem Einschalten, den einfachen Zugriff auf die Hardware und deren Abkapselung, die Verwaltung der Hardware-Ressourcen wie Prozessorzeit, Speicher, Netzwerk usw. – dazu gehört insbesondere die Funktionalität, mehrere Programme gleichzeitig laufen zu lassen – sowie die Interaktion mit der Benutzerin und eine entsprechende Bedienungsoberfläche.

Betriebssysteme haben sich zusammen mit der Hardware weiterentwickelt, immer wieder sind neue Funktionalitäten hinzugekommen, und mittlerweile handelt es sich um riesige Programmsammlungen mit Millionen Zeilen von Code, die eine enorme Komplexität aufweisen, an denen eine beträchtliche Anzahl von Entwicklern arbeitet, und die aus diesem Grund auch immer wieder Fehler aufweisen und kontinuierlich verbessert werden müssen. Um der Komplexität Herr zu werden, sind Betriebssysteme in überschaubare, hierarchisch organisierte Einheiten aufgeteilt. Bsp. 4.12 gibt Ihnen einen Einblick in den Aufbau des Betriebssystems Linux.

Beispiel 4.12: Linux sein Linux

Linux ist ein Derivat des Betriebssystems UNIX, das ursprünglich von dem Finnen Linus Torvalds Anfang der 1990er entwickelt wurde. Der Name Linux leitet sich von Linus ab, spielt aber eben auch auf UNIX an. Viele Programmierer beteiligten sich schließlich an der Entwicklung dieses Betriebssystems, und es gab bald verschiedene Firmen, die Linux-Distributionen vertrieben und dabei auf demselben Linux-Programmfundament aufsetzten. Linux ist wie UNIX sehr umfangreich, besitzt diverse Komponenten und ist schichtenartig aufgebaut (siehe Abb. 4.27). Auf unterster Ebene befinden sich einerseits die Programmteile, die die einzelnen Hardware-Komponenten ansprechen, sogenannte Geräte-Treiber; andererseits sind dort auch das Prozessmanagement (Scheduling und Synchronisation), das Speichermanagement, das Dateisystem-Management und das Netzwerk-Management angesiedelt. Über dieser Kernel genannten Schicht befindet sich eine Programmbibliothek, die Standardroutinen zur Ein- und Ausgabe zur Verfügung stellt. Dann gibt es darüber noch zusätzliche Programme, die als eigenständige Prozesse laufen, z. B. die Benutzerverwaltung, die grafische Bedienungsoberfläche und Weiteres. Und ganz oben laufen schließlich die Anwendungsprogramme, z. B. das Musik- oder das Textverarbeitungsprogramm.

Da Betriebssysteme nicht mehr einzelne Programme sind, sondern über viele Programme mit vielen Prozessen realisiert werden, ergibt sich eine weitere

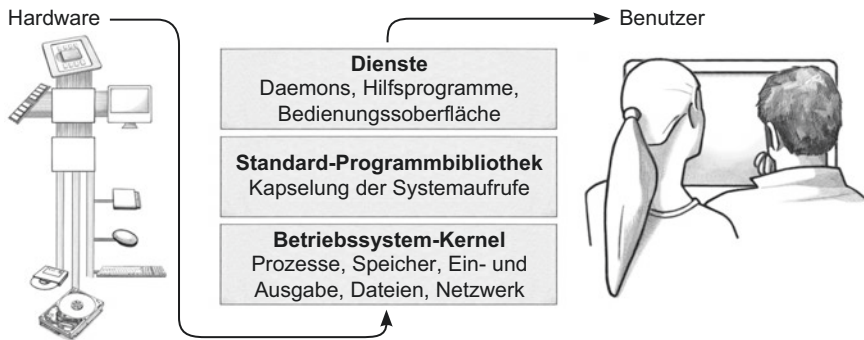


Abb. 4.27 Schematischer Aufbau des Linux-Betriebssystems. (© Susanne Staubli, Eckart Zitzler)

Schwierigkeit. Ein Betriebssystem muss erst einmal warmlaufen, wenn der Computer angeschaltet wird. Den ganzen Vorgang nennt man im Englischen *booting*, im Deutschen sagt man dazu Hochfahren oder Booten. Während dieses Vorgangs ist der Computer mit sich selbst beschäftigt und noch nicht ansprechbar. Er handelt sich hoch in die Betriebsbereitschaft, schrittweise werden alle Komponenten aktiviert, die dafür notwendig sind. Das erinnert etwas an die bekannte Lügengeschichte des Barons von Münchhausen, der berichtete, sich und sein Pferd am eigenen Schopf aus dem Sumpf gezogen zu haben. Im Englischen gibt es dafür ein anderes Bild: die Idee, sich selbst an den Stiefelriemen (*bootstraps*) über einen Zaun zu ziehen – der Ausdruck *booting* ist entsprechend nur eine Abkürzung für *bootstrapping*. Am Schopf oder an den Stiefelriemen, beides geht natürlich nicht – und es trifft ja auch beim Computer nicht zu, dass er ohne Bezugspunkt hochklettern könnte. Einen kleinen Anker müssen wir ihm auswerfen, ein Startprogramm, von dem ausgehend das gesamte System hochgefahren wird. Bsp. 4.13 erklärt die einzelnen Schritte dieses Vorgangs im Detail.

Beispiel 4.13: Sich am eigenen Schopf aus dem Sumpf ziehen

Was geschieht also genau beim Einschalten eines Computers? Abb. 4.28 illustriert den Vorgang. Nun, zu Beginn passiert das, was ich bereits erläutert hatte: Ein in der Hardware festgeschriebenes Startprogramm wird ausgeführt, dessen Aufgabe darin besteht, einerseits das Betriebssystem in den Hauptspeicher zu laden sowie anschließend zur Ausführung zu bringen und andererseits minimale Funktionen für die Ansteuerung von Hardware-Komponenten zur Verfügung zu stellen. Sobald das Startprogramm die Kontrolle an das Betriebssystem übergibt, baut sich letzteres selbst auf. Es muss nun seinerseits alle erforderlichen Datenstrukturen anlegen und seine Komponenten in

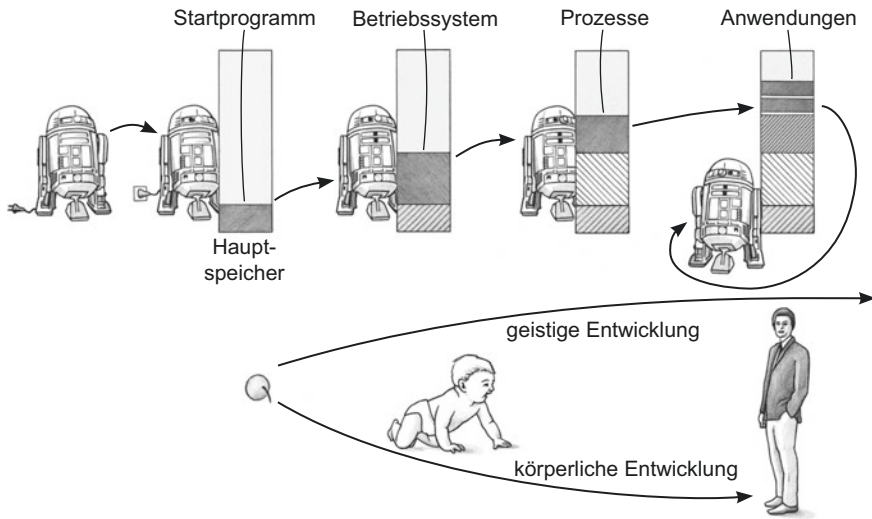


Abb. 4.28 Illustration des Boot-Vorgangs. (© Susanne Staubli, Eckart Zitzler)

Form von Prozessen starten. Da gibt es z. B. Prozesse, die die Kommunikation mit der Benutzerin regeln und die Bedienungs Oberfläche zur Verfügung stellen; da gibt es Daemons, im Hintergrund ablaufende Programme, die für die Druckeransteuerung oder die Netzwerkanbindung zuständig sind; und all diese Aktivitäten laufen gleichzeitig ab und werden nun geordnet gestartet. Das Betriebssystem braucht sozusagen Zeit, um die Arbeitsfläche, den Hauptspeicher, einzurichten und zu ordnen. Erst dann kommt eigentlich die Benutzerin ins Spiel, erst dann werden Eingaben über Tastatur, Mikrofon usw. entgegengenommen und auf dem Bildschirm oder anderen Ausgabegeräten dargestellt.

Ein Betriebssystem lässt sich im weitesten Sinne vergleichen mit den Fähigkeiten und dem Wissen, die wir uns im Laufe unserer Kindheit erwerben, um uns selbstständig in der heutigen Welt zurechtzufinden. Wir lernen gehen, sprechen, schwimmen, nehmen in der Schule unser Kulturgut auf, üben uns im Sozialverhalten usw. Das Gelernte steht uns später als Erwachsener zur Verfügung und ist vorhanden, ohne dass wir weiter darüber nachdenken müssten. All diese Ressourcen und Kompetenzen sind die Basis, auf der wir unser Leben gestalten. Und was ist dann Booten beim Menschen? Das Ganze nennt sich Ontogenese, die Entwicklung eines Organismus. Und dieser Prozess ist Booten im Großen, so viele faszinierende Veränderungen laufen dort ab, wenn sich aus einer befruchteten Eizelle während vieler Jahre ein erwachsener Mensch herausbildet. Wenn der Mensch

geboren wird, entwickelt sich seine „Hardware“ ja noch, viele Verknüpfungen im Gehirn entwickeln sich im Tun, z. B. das Gehen, anderes wiederum ist schon vorgegeben und funktioniert bereits sehr früh, etwa was das vegetative Nervensystem betrifft, wie der Herzschlag oder die Atmung. Dann, mit der Zeit, lernen wir immer mehr, bis wir schließlich selbstständig sind. Allerdings ist die Entwicklung unseres „Betriebssystems“ nie abgeschlossen, es wird laufend ergänzt und erweitert mit jeder neuen Fähigkeit, die wir erwerben.

Fazit

Wir haben gesehen, dass das System Computer in seinen Grundzügen diverse Aspekte der menschlichen Anatomie widerspiegelt. Da gibt es eine Steuerungszentrale mit Prozessor und Hauptspeicher, die den Kern der Informationsverarbeitung darstellen – quasi das Gehirn. Diese Zentrale ist über mehrere Leitungsbündel, Busse, mit den anderen Komponenten des Computers verbunden, die das Tor zur Umwelt darstellen: mit den Ein- und Ausgabegeräten. Das ist durchaus vergleichbar zum Menschen, wo Sinnes- und Muskelzellen mittels Nervensträngen und dem Rückenmark mit dem Gehirn verbunden sind. Und genauso, wie wir Menschen als ein unbeschriebenes Blatt auf die Welt kommen und zuerst Grundwissen und Grundfähigkeiten erwerben müssen, um uns eigenständig in dieser Welt zurechtfinden zu können, benötigt ein Computer ein erstes Programm – das Betriebssystem –, um voll funktionstüchtig zu sein. Und es ist erstaunlich zu sehen, wie viele weitere Analogien entstehen, wobei wir uns natürlich zugleich der engen Grenzen dieser Gegenüberstellung bewusst sein müssen. Zum Beispiel spielt beim Menschen die Wahrnehmung seiner Innenwelt eine enorm wichtige Rolle, während Computersysteme per se erst einmal vor allem nach außen gerichtet sind. Je komplexer Computersysteme jedoch werden – und das ist die zwangsläufige Entwicklung –, umso wichtiger werden die Möglichkeiten, die „Befindlichkeit“ des „Computerkörpers“ zu erfassen. Tatsächlich gibt es heute schon Service-Komponenten in großen Computern, bei denen die Administratoren, Ärzten vergleichbar, Messungen vornehmen und auf dieser Basis Diagnosen erstellen und Maßnahmen ergreifen können. Und auch andere Aspekte, die beim Menschen zentral sind, z. B. der sparsame Umgang mit Energie und die Energiegewinnung, spielen bei Computersystemen eine zunehmend wichtigere Rolle.

Zum Nach- und Weiterlesen

- Damasio, A. (2011). *Selbst ist der Mensch: Körper, Geist und die Entstehung des menschlichen Bewusstseins*. München: Siedler.
- Frings, S., & Müller, F. (2014). *Biologie der Sinne: Vom Molekül zur Wahrnehmung*. Berlin: Springer.
- Grier, D. A. (2007). *When computers were human*. Princeton: Princeton University Press.
- Neumann, J. von. (1991). *Die Rechenmaschine und das Gehirn* (6. Aufl.). München: Oldenbourg.
- Patterson, D., & Hennessy, J. L. (2011). *Rechnerorganisation und Rechnerentwurf: Die Hardware/Software-Schnittstelle* (4. Aufl.). München: Oldenbourg.
- Tanenbaum, A. S., & Bos, H. (2016). *Moderne Betriebssysteme* (4. Aufl.). München: Pearson Studium.
- Shetterly, M. L. (2017). *Hidden Figures: Unerkannte Heldinnen*. New York: HarperCollins.

Bildquellen

Abb. 4.3: Harris & Ewing Collection, Library of Congress, <https://www.loc.gov/item/hec2009007278/> (links, abgerufen am 26.05.2017); NASA, <https://www.dfrc.nasa.gov/Gallery/Photo/Places/HTML/E49-54.html> (rechts, abgerufen am 26.05.2017)

5

Speichern und Erinnern

Überblick

In diesem Kapitel dreht sich alles um die Kerneigenschaft eines Computers, die am Anfang seiner Entwicklung stand: die Speicherfähigkeit. Dabei geht es nicht nur um die Frage, wie sich Informationen überhaupt festhalten lassen, sondern auch um die geschickte Nutzung der vorhandenen Speichergefäße. Zunächst werden wir prominente Speichertechnologien beleuchten, die die herkömmliche, in einem Prozessor eingesetzte Technologie der rückgekoppelten Stromkreise auf verschiedenste Art und Weise ergänzen. Mit ihnen lassen sich robuste, transportable, große und erschwingliche Speicherkomponenten bauen, die ihren Inhalt auch dann behalten, wenn der Strom abgestellt wird. Anschließend werden wir uns anschauen, wie sich solche Aufbewahrungsgefäße für Daten effizient nutzen lassen. Indem unterschiedliche Speicherarten hierarchisch miteinander kombiniert und Daten geschickt strukturiert werden, können auch große Datenmengen schnell abgelegt und abgerufen werden. Insbesondere die Frage der Datenstrukturierung ist in der Informatik ein zentrales Thema, welches bei der Entwicklung von Programmen eine wichtige Rolle spielt.

Komplexe Verarbeitungsvorgänge setzen sich in der Regel aus vielen kleinen, einfachen Schritten zusammen, wie wir das bei der Turing-Maschine oder den digitalen Schaltungen gesehen haben. Dabei fallen Zwischenergebnisse an, die geeignet festgehalten und für die spätere Weiterverwertung zur Verfügung gestellt werden müssen. Der Speicher ist folglich essenziell für die Informationsverarbeitung, gleichzeitig ist es eine Krux mit ihm: Blitzschnell soll er sein, maximales Fassungsvermögen aufweisen bei minimalem Platzbedarf, seinen Inhalt auch langfristig sicher aufbewahren und gleichzeitig mit geringem Aufwand herstellbar sein. Wir hätten gerne den Fünfer

(das Fünfrappen-Stück) und das Weggli (das Milchbrötchen, das einmal fünf Rappen kostete), wie es im Schweizerdeutschen so schön heißt. Dass es diesen Super-Speicher so nicht gibt, liegt auf der Hand. Um dem Ideal trotzdem möglichst nahezukommen, hat man sich ausgeklügelte Strategien überlegt, sowohl auf der Hardware- als auch auf der Software-Ebene. Das ist der Grund, warum ich dem Speicher ein eigenes Kapitel widme.

Im Folgenden schauen wir uns an, welche Antworten die Informatik bislang gefunden hat, um mit den konkurrierenden Ansprüchen umzugehen. Zunächst einmal werfen wir einen Blick auf verschiedene Speichertechnologien, anknüpfend an die Diskussionen in Kap. 2 und 3, wo wir aus rückgekoppelten Stromkreisen Speicherkomponenten wie Register und Hauptspeicher konstruiert hatten. Im Fokus steht dabei der Bau und Einsatz von Massenspeichern, die große Datenmengen dauerhaft speichern können; insbesondere werden wir sehen, wie sich diese geschickt in ein Computersystem integrieren lassen. Anschließend werden wir uns mit den Möglichkeiten der Datenstrukturierung auseinandersetzen. Dabei geht es um die Art und Weise, wie wir Daten im Speicher ablegen. Letztlich betrachten wir dasselbe Grundproblem, nämlich den inhärenten Konflikt zwischen Speicherkapazität und Zugriffsgeschwindigkeit, aus zwei Perspektiven: einmal aus dem Blickwinkel der Hardware, einmal aus dem der Software. In beiden Fällen geht es um eine geschickte Organisation des Speichers.

Vom Leben eines Bücherwurms

Doch bevor wir in die technischen Untiefen hinabsteigen, lassen Sie uns ein Jahrhundert zurückspringen in eine Zeit, als noch gebundene, bedruckte Papierbündel – Bücher genannt – das primäre Medium waren, um Wissen festzuhalten. Denn die Informatik hat das Rad nicht neu erfunden und viele der Lösungsansätze, die wir später diskutieren werden, basieren auf Prinzipien, die der Mensch seit jeher einsetzt. Werfen Sie z. B. einen Blick auf die in Abb. 5.1 dargestellte Situation, in der Sie einen Bücherwurm bei der Arbeit sehen, vor sich und um sich herum eine riesige Wissensbibliothek in Form von Büchern. Ob es sich dabei um einen Wissenschaftler, Juristen oder Buchautor handelt, sei Ihrer Fantasie freigestellt (tatsächlich ist die Abbildung durch das Arbeitszimmer des Schweizer Schriftstellers Franz Hohler inspiriert).

Augenscheinlich kommen in der skizzierten Szene ganz unterschiedliche Aufbewahrungsgefäße für Bücher zum Einsatz. Jene Bücher, mit denen sich der Bücherwurm gerade beschäftigt, liegen aufgeklappt vor ihm. Weil



Abb. 5.1 Der Arbeitsplatz eines Bücherwurms. (© Susanne Staubli, Eckart Zitzler)

der Platz auf dem Schreibtisch beschränkt ist, sind ein paar im Gebrauch befindliche Bücher kompakt am Rand übereinandergestapelt. Einige wichtige Nachschlagewerke, die immer wieder zurate gezogen werden müssen, liegen in fächerartig angeordneten Ablagekörben, die sich ebenfalls auf dem Schreibtisch befinden. Der Großteil der Bücher ist jedoch in einer breiten Regalwand untergebracht, deren oberer Teil nur über eine Leiter erreicht werden kann und der entsprechend selten genutzte Exemplare enthält. Und wir können das Ganze noch weiterspinnen: Vielleicht befinden sich zusätzliche Bücher im Keller, während gewisse Raritäten in der Stadtbibliothek aufbewahrt werden.

Dass es überhaupt verschiedener Aufbewahrungsgefäße bedarf, ist einer grundsätzlichen Problematik im Zusammenhang mit Speichern geschuldet: Entweder passt viel hinein oder der Inhalt ist schnell zugreifbar, mehr vom einen bedingt weniger vom anderen und umgekehrt. Unser Bücherwurm hat das Dilemma geschickt gelöst, indem er die verschiedenen Aufbewahrungsgefäße miteinander kombiniert. Der Schreibtisch stellt einen kleinen schnellen Speicher dar, das Regal einen großen langsamen und der Keller bzw. die Bibliothek einen noch größeren und noch schwerer zugänglichen. Das Ganze funktioniert, weil der Bücherwurm aktuell immer nur mit

einer kleinen Anzahl von Büchern beschäftigt ist. Er muss vermutlich nur gelegentlich ein Buch aus dem Regal holen und dafür ein anderes ins Regal zurückstellen. Und auch der Gang zur Stadtbücherei (oder in den Keller) wird eher die Ausnahme sein, dafür bringt er von dort dann immer gleich mehrere Bücher mit.

Zudem können die Aufbewahrungsgefäße unterschiedlich genutzt werden. Die einzelnen Ablagekörbe auf dem Schreibtisch können beschriftet sein und somit den Zugriff wesentlich beschleunigen. Und wenn der Bücherwurm sich die Mühe macht, die Bücher im Regal in Kategorien und darüber hinaus sortiert anzuordnen, sind sie viel einfacher bzw. schneller zu finden als ohne Vorsortierung. Indem er die Wissensmenge strukturiert, erleichtert er sich den Zugriff darauf, auch wenn die Strukturierung mit zusätzlichem Aufwand verbunden ist. Das gilt übrigens auch für die Bücher selbst, auch sie weisen eine klare Struktur auf, die sich je nach Kulturkreis unterscheiden kann. Da sind Papierblätter, die zusammengeheftet und somit in eine fixe Reihenfolge gebracht werden. Diese Blätter werden nummeriert, beispielsweise von vorne nach hinten, und auf den Blättern werden Buchstaben und weitere Symbole aufgebracht, wiederum gemäß einem vorgegebenen Schema, z. B. von links nach rechts und von oben nach unten. Häufig gibt es noch ein Inhaltsverzeichnis, Unterteilungen in Kapitel und Abschnitte, möglicherweise ein Stichwortverzeichnis. All das dient dem Zurechtfinden im Datenwust.

Und wenn wir noch weiter eintauchen in die Wissensdarstellung und das Zeichensystem unserer Sprache untersuchen, dann finden wir auch dort unzählige Strukturen. Sprache verbindet Informationen miteinander, und das auf höchst erstaunliche und vielschichtige Art und Weise. Wir bilden Worte – also elementare Informationen im Sinne von Begriffen – und setzen daraus Sätze zusammen, die komplexere Informationen abbilden. Wie wir das machen, ist festgelegt durch eine Grammatik. Sie definiert die entsprechenden Regeln, gibt den Sätzen Struktur und ermöglicht uns, aus Sprachdaten die richtigen Informationen zu rekonstruieren (Abb. 5.2). In diesem Fall erleichtert die Datenstrukturierung nicht nur den Zugriff auf die Informationen, sondern beinhaltet auch Zusatzinformationen, die Auswirkungen auf die Bedeutung eines Satzgebildes haben.

Wir sehen: Das Fünfer-und-Weggli-Problem lässt sich nicht wegzaubern, aber weitreichend entschärfen. Die Lösungswege, die wir soeben im Zusammenhang mit Büchern studiert haben, lassen sich auf Computersysteme übertragen. Auch dort werden verschiedene Speicherkomponenten miteinander kombiniert, werden Daten je nach Anwendungszweck geeignet strukturiert. Das wollen wir uns jetzt im Detail

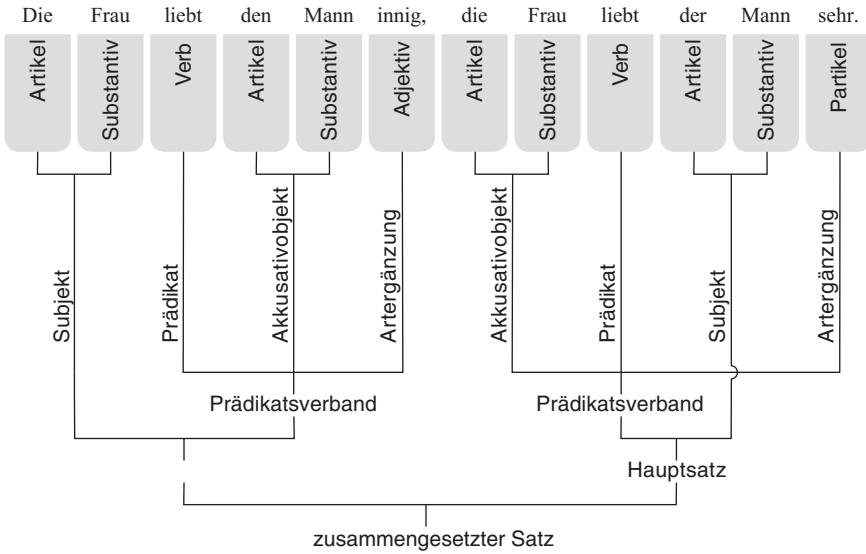


Abb. 5.2 Die Struktur eines Satzes und die Beziehung der Wörter untereinander werden durch die Grammatik bestimmt. (© Susanne Staubli, Eckart Zitzler)

anschauen, zuerst auf der Ebene der Hardware. Später wenden wir uns dann den Möglichkeiten zu, die bei der Programmierung eingesetzt werden.

Wie man Informationen haltbar macht

Wie sich ein elektronischer Speicher bauen lässt, wissen wir bereits: über die Rückkopplung von Stromkreisen (vgl. Kap. 2). Allerdings handelt es sich dabei um einen flüchtigen Speicher, d. h., er verliert seinen Inhalt, sobald der Strom abgestellt wird. Wollen wir eine dauerhafte Speicherung sicherstellen, müssen wir auf andere Technologien zurückgreifen. Dasselbe gilt, wenn Daten transportiert oder in großen Mengen aufgenommen werden sollen. Aus diesem Grund wird der Hauptspeicher eines Computers zumeist mit einem oder mehreren Zusatzspeichern ergänzt und zu einem Speichersystem kombiniert.

Digitale Gravuren

Die meisten nichtflüchtigen Zusatzspeicher beruhen auf einer von drei Speichertechnologien. Magnetische Verfahren nutzen eine Eigenschaft

bestimmter Materialien aus: Die enthaltenen Atome lassen sich in zwei Richtungen magnetisieren und somit als Zeichenträger umfunktionieren. Dieser Ansatz der Datenspeicherung hat die Computertechnik von Anfang an begleitet. Erst waren es große Magnetbänder – diese sind immer noch beliebte Speichermedien, wenn es um die Archivierung von Daten geht –, später kamen rotierende Magnetscheiben hinzu, die uns heute noch als Festplatten begegnen. Optische Speichertechnologien hingegen verwenden Laserlicht, um die Oberflächenstruktur eines Materials zu verändern und die resultierenden Markierungen wieder auszulesen. Sie betraten wesentlich später die Bühne, ebenfalls in Form von Scheiben. Die Compact Disc (CD) machte den Anfang und verbreitete sich in den 1990ern, anschließend folgten diverse Weiterentwicklungen wie z. B. die DVD. Elektronischen Speichertechnologien schließlich liegt die Idee zugrunde, Schalter zu verwenden, die ihren Zustand – geöffnet oder geschlossen – unabhängig von der Stromversorgung beibehalten. Ein weit verbreiteter Vertreter dieser Gattung ist der Flash-Speicher. Er setzt sich aus vielen Transistoren zusammen und konserviert deren Schaltungszustand mittels elektrischer Ladungen. Diese Art der Speicherung ist die jüngste und mit Abstand schnellste der drei Technologien. Wer etwas tiefer in die Technik einsteigen möchte, findet in Anm. 5.1 weitere Hintergrundinformationen.

Anmerkung 5.1: Technologien für die dauerhafte Speicherung

Abb. 5.3 illustriert die Arbeitsweise der drei erwähnten Speicherverfahren. Bei magnetischen Technologien bildet ein magnetisierbares Material die Grundlage. Dieses ist in kleine Regionen unterteilt, die wiederum aus mehreren magnetischen Domänen bestehen – das sind Bereiche, in denen die Magnetisierung der einzelnen Atome in die gleiche Richtung zeigt. Eine Region fungiert als elementarer Zeichenträger, und ihre magnetische Ausrichtung steht entweder für eine 0 oder eine 1. Das Schreiben und Lesen kann z. B. über einen Elektromagneten erfolgen, eine Spule, die um ein u-förmiges Eisenstück gewickelt ist. Fließt durch die Spule ein Strom, so wird zwischen den Enden des Eisenstücks ein Magnetfeld induziert (die Richtung des Stroms bestimmt die Richtung des Magnetfelds) und die dort befindlichen physikalischen Domänen werden entsprechend ausgerichtet. Beim Lesen wird das umgekehrte Prinzip verwendet; immer wenn die Magnetisierungsrichtung wechselt, entsteht eine messbare Spannung in der Spule, je nach Art des Wechsels eine negative oder positive. Letztlich werden also nicht die einzelnen Regionen, sondern die Übergänge zwischen verschiedenen gepolten Regionen detektiert.

Optische Speicherverfahren nutzen ein reflektierendes Material zur Datenspeicherung sowie Laserlicht, um darauf Markierungen abzutasten und ggf. auch einzuprägen. Eine Methode des Schreibens besteht darin, die Oberfläche des Datenträgers mit einem Laser punktuell zu erhitzen und dann gezielt abzukühlen. Je nach Abkühlzeit wird das Material an dieser Stelle schlecht oder gut reflektierend; diese zwei Zustände stehen für 0 und 1. Beim

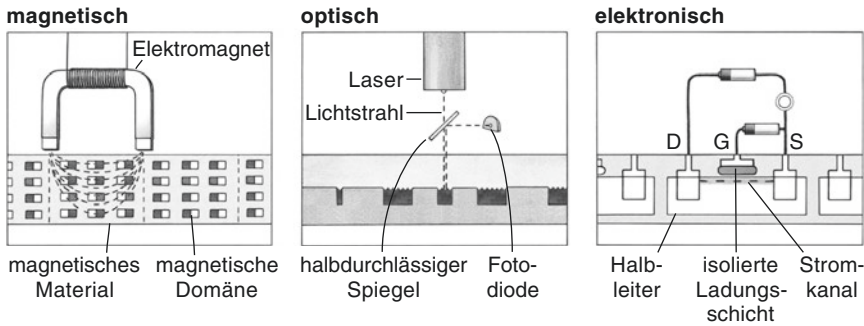


Abb. 5.3 Die drei zentralen Technologien, auf denen Computerspeicher beruhen (in historischer Reihenfolge): (a) magnetische Speicherung, (b) optische Speicherung und (c) elektronische Speicherung. (© Susanne Staubli, Eckart Zitzler)

Lesen wiederum tastet ein Laser die Oberfläche ab und das reflektierte Licht wird über einen halb durchlässigen Spiegel zu einer Fotodiode weitergeleitet. Das Laserlicht wird je nach Markierung unterschiedlich reflektiert, was dann über die Fotodiode erkannt wird.

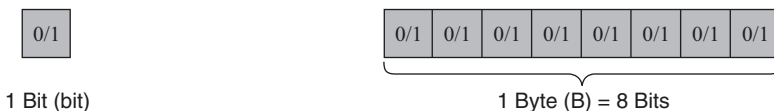
Der Flash-Speicher als Vertreter der elektronischen Speichertechnologien nutzt spezielle Transistoren als Speicherelemente. Jeder dieser Transistoren realisiert einen einfachen stromgesteuerten Schalter, über den ein Strom zwischen den äußeren Anschlüssen (S und D in Abb. 5.3) ermöglicht oder blockiert werden kann. Zudem besitzen diese Transistoren in der Mitte ein kleines, isoliertes Metallplättchen, welches durch Anlegen einer hohen positiven oder negativen Spannung an den Anschlüssen G und S mit einer Ladung versehen bzw. entladen werden kann – dabei kommen quantenmechanische Effekte zum Tragen. Das Verhalten des Transistors hängt letztlich davon ab, ob das Metallplättchen eine Ladung enthält oder nicht. In einem Fall ist der Transistor leitend, d. h., es etabliert sich ein Stromkanal im Halbleiter, im anderen Fall isolierend, d. h., zwischen den äußeren Anschlüssen kann kein Strom fließen. So lassen sich wiederum die zwei Zeichen 0 und 1 codieren.

Die drei grundlegenden Speichertechnologien können ganz unterschiedlich eingesetzt werden, und so gibt es eine Vielfalt an Speicherbausteinen und -geräten, die wiederum dem Fünfer-Weggli-Dilemma geschuldet ist. Welcher Datenspeicher für welchen Zweck am besten geeignet ist, hängt von vielen Faktoren ab. Im Zentrum steht immer die Speicherkapazität. Sie wird traditionellerweise in Bytes bemessen (siehe Anm. 5.2) und kann eigentlich nie groß genug sein. Ihr gegenüber stehen die Zugriffsgeschwindigkeit und die Herstellungskosten, die sich selbst diametral zueinander verhalten. Weitere Aspekte können Dauerhaftigkeit, Transportabilität, Robustheit usw.

sein. Eine magnetische Technologie wie die Festplatte beispielsweise bietet viel Speicherplatz bei moderaten Kosten, ist in der Regel jedoch langsam, weil mechanische Vorgänge involviert sind. Optische Datenträger wie CD und DVD erlebten einen großen Hype, als es darum ging, preiswerte transportable Speichermedien zur Verfügung zu stellen; mit dem Aufkommen hoher Bandbreiten für Internetzugänge wurden sie jedoch wieder zurückgedrängt. Flash-Speicher scheinen das Ideal zu bieten mit hohen Speicherdichten und Zugriffsgeschwindigkeiten – sie kommen ja ohne bewegliche Komponenten aus –, doch dafür sind die Herstellungskosten (noch) relativ hoch. Sie ersetzen zunehmend Festplatten, jedoch nicht die flüchtigen elektronischen Speicher, da die Schreibvorgänge wesentlich länger dauern.

Anmerkung 5.2: Bits und Bytes

Traditionellerweise ist die kleinste Einheit, mit der Speicher operieren, das Byte, das sind 8 Bits – weil eben die ersten Prozessoren intern mit 8-Bit-Dualzahlen arbeiteten. Später nahm man 16-Bit- oder 32-Bit-Wörter, doch Byte ist nach wie vor die Einheit, um die Kapazität eines Speichers zu klassifizieren. Die Präfixe Kilo, Mega, Giga, Tera usw. dienen dazu auszudrücken, ob man ein Tausend, eine Million, eine Milliarde, eine Billion oder mehr Bytes meint (siehe Abb. 5.4). Hier gab es zu der Zeit, als die Computer allmählich die Haushalte eroberten, noch etwas Verwirrung. Eigentlich ist die Kapazität eines Arbeitsspeichers ja durch die Anzahl der Adressleitungen bestimmt, weil die Adressen über Dualzahlen dargestellt werden. Beispielsweise ermöglichen 10 Leitungen einen Speicher mit 1024 Bytes, 20 Leitungen einen Speicher mit 1.048.576 Bytes usw.



Vorsätze mit Zehnerpotenzen		Vorsätze mit Zweierpotenzen	
Name (Symbol)	Bedeutung	Bedeutung	Name (Symbol)
Kilo (K)	$10^3 = 1.000$	$2^{10} = 1.024$	Kibi (Ki)
Mega (M)	$10^6 = 1.000.000$	$2^{20} = 1.048.576$	Mebi (Mi)
Giga (G)	$10^9 = 1.000.000.000$	$2^{30} = 1.073.741.824$	Gibi (Gi)
Tera (T)	$10^{12} = 1.000.000.000.000$	$2^{40} = 1.099.511.627.776$	Tebi (Ti)
Peta (P)	$10^{15} = \dots$	$2^{50} = \dots$	Pebi (Pi)
...	...	...	...

Abb. 5.4 Bezeichnungen für die Speichergröße. (© Susanne Staubli, Eckart Zitzler)

Man meinte also $2 \text{ hoch } 10$ bzw. $2 \text{ hoch } 20$ und schrieb damals auch Kilo- bzw. Mega-Byte. Aus diesem Grund wurde 1996 vorgeschlagen, in letzterem Fall andere Wörter zu benutzen, also Kibibyte und Mebibyte anstatt Kilo- und Megabyte, wie in Abb. 5.4 gezeigt ist. Die letzteren Bezeichnungen werden aber selten verwendet.

Übrigens werden Speicher häufig noch gemäß ihrer Flexibilität klassifiziert. Man spricht von ROM (*read only memory*), wenn man nichtveränderliche Speicherkomponenten meint. Beispielsweise nannte man nicht veränderbare CDs auch CD-ROM und einmalig beschreibbare CDs bezeichnete man als WORM (*write once read many*). Flash-Speicher fallen in die Kategorie der EEPROM: Das sind nichtflüchtige elektronische Speicher, deren Inhalt auf elektrische Art und Weise gelöscht und neu geschrieben werden kann (EEPROM steht für *electrically erasable programmable read-only memory*). Das Gegenteil, also flüchtiger Speicher, der seinen Inhalt beim Ausschalten des Stroms verliert, wird häufig als RAM bezeichnet, obwohl das eigentlich nicht korrekt ist. RAM steht für *random access memory* und bedeutet, dass alle Speicherzellen einzeln angesprochen werden können und die Zeit zum Lesen bzw. Schreiben eines Datums für alle Zellen gleich ist. Das war als Unterscheidung zu Magnetbändern und Festplatten gedacht, bei denen die Daten hintereinander oder blockweise eingelesen werden müssen und die Zugriffszeit stark variieren kann. Und da Hauptspeicher immer über RAM-Bausteine realisiert und zumeist flüchtig sind, hat sich diese Fehlinterpretation eingeschlichen.

Speicher organisieren

Wir haben jetzt gesehen, wie sich digitale Markierungen technologisch realisieren lassen. Damit daraus aber auch tatsächlich ein nutzbarer Massenspeicher wird – einer, der große Datenmengen dauerhaft aufnehmen kann –, braucht es noch einiges mehr. Da geht es beispielsweise darum, in welcher Form das verwendete Material eingesetzt wird (als Band, Chip oder Scheibe), wie das Material strukturiert wird (auf einer CD werden die Daten entlang einer Spirale auf dem Material angeordnet, auf einer Festplatte sind es konzentrische Kreise), wie 0-1-Sequenzen effektiv abgespeichert werden (in der Regel werden die 0-1-Folgen aufgrund technischer Rahmenbedingungen in eine andere Darstellung übersetzt) und wie weitere Markierungen angebracht werden (sie entsprechen den Blättern, Seitenzahlen und Linien eines leeren Notizbuches). Tatsächlich sind

Speichergeräte äußerst komplexe Systeme und ihr Bau ist eine Wissenschaft für sich. Ich möchte Sie hier nicht mit weiteren Details langweilen. Falls Sie dennoch tiefer einsteigen wollen: Bsp. 5.1 gibt Ihnen einen Eindruck vom Aufbau einer Festplatte.

Beispiel 5.1: Wie eine Festplatte aufgebaut ist

Eine Festplatte besteht aus mehreren Scheiben z. B. aus Aluminium, die mit einer magnetischen Schicht überzogen sind, auf der Datenspuren in konzentrischen Kreisen angebracht sind (Abb. 5.5). Diese Kreise sind weiter unterteilt in Blöcke, die direkt angesprochen werden können; jeder Block umfasst eine größere Anzahl Bytes. Die Blockgröße ist ein Kompromiss zwischen der Zeit, überhaupt den Lesekopf zu positionieren, der Zeit zum Lesen eines Blocks und einer potenziellen Speicherverschwendung. Jedes Byte einzeln zu lesen, dauert bei einer Festplatte einfach zu lange, daher macht es mehr Sinn, direkt mehrere Bytes hintereinander einzulesen. Der Zugriff erfolgt also blockweise, und um den Zugriffsaufwand zu rechtfertigen, werden gleich viele Bytes eingelesen (oder geschrieben).

Zum Vergleich: Bei einem Magnetband wird eine 0-1-Sequenz vollständig hintereinander abgespeichert, es ist keine weitere Untergliederung nötig. Beim Hauptspeicher ist die Granularität der Blöcke noch viel feiner, in jeder Speicherzelle hat nur eine kleine Anzahl von Bits Platz (bei unserem Beispiel-Prozessor waren es 8 Bits, d. h., eine 0-1-Sequenz wird in Byte-große Blöcke unterteilt).

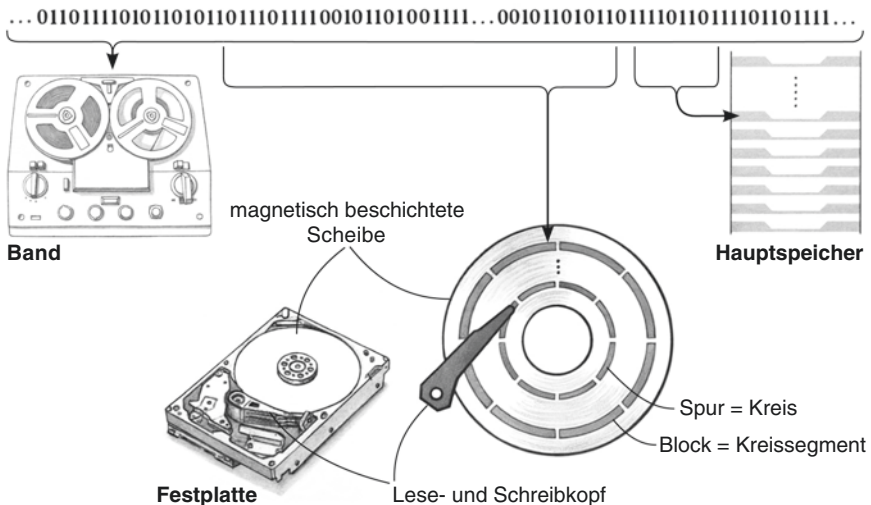


Abb. 5.5 Wie Daten auf einer Festplatte abgespeichert werden. (© Susanne Staubli, Eckart Zitzler)

Wie lässt sich nun ein Massenspeicher in ein Computersystem integrieren und mit den vorhandenen Speicherressourcen koppeln? Die einfachste Möglichkeit besteht darin, den Hauptspeicherinhalt eins zu eins auf den Massenspeicher zu kopieren, bevor der Strom abgestellt wird, und später nach dem Einschalten wieder zurückzukopieren. Diese Möglichkeit wird manchmal genutzt, um den Zustand des Computers einzufrieren und ihn quasi in einen Winterschlaf zu versetzen. Doch richtig praktisch ist sie noch nicht, denn einerseits dauert dieser Kopiervorgang sehr lange und andererseits lässt sich so die große Kapazität des Massenspeichers nicht nutzen. Stattdessen hat man sich das Konzept der Datei ausgedacht: Dabei handelt es sich schlicht um eine Sequenz von Nullen und Einsen, die mit einem Namen versehen ist. Der Begriff Datei ist in Analogie zu Kartei (gebildet aus dem Wort „Karten“) aus dem Wort „Daten“ konstruiert. Das Konzept der Datei ist so allgemein, dass sich so auch nur Ausschnitte des Hauptspeichers extern speichern lassen; zudem lassen sich auch Datensequenzen erzeugen, die die Größe des Hauptspeichers übersteigen. Die Größe einer Datei kann variieren, es gibt jedoch zumeist eine Größenbeschränkung.

Wie eine Datei auf einem Massenspeicher abgelegt wird, hängt vom verwendeten Typ ab. Bei einer Festplatte und auch einer *solid state disc* (SSD) – das ist ein auf Flash-Speicher basierender Zusatzspeicher, der wie eine Festplatte angesprochen wird – wird eine Datei je nach Länge auf mehrere, beliebig gewählte Blöcke verteilt. Bsp. 5.2 zeigt auf, wie das im Detail funktioniert. Die Speicherorganisation dahinter nennt sich Dateisystem, das Thema hatten wir in Kap. 4 bereits angesprochen. So eine Speicherorganisation muss allerdings erst einmal eingerichtet werden; beispielsweise muss eine Art Inhaltsverzeichnis mit den Namen der Dateien abgespeichert werden, es muss eine Liste der freien Blöcke geführt werden usw. Diesen Vorgang des Einrichtens nennt man übrigens Formatierung. Das heißt auch, nicht alle Speicherzellen – wenn wir den kleinsten Bereich, der ein Bit speichern kann, so einmal nennen wollen – repräsentieren in einer Datei enthaltene Nutzdaten. Das geht sogar so weit, dass man eine physikalische Festplatte in mehrere virtuelle Festplatten unterteilen kann, diese nennen sich Partitionen.

Beispiel 5.2: Dateien als Speichereinheiten

Dateien werden auf einer Festplatte als eine Folge von Blöcken abgelegt, wobei irgendwo vermerkt sein muss, welche Blöcke effektiv zu welcher Datei gehören. Eine einfache Möglichkeit besteht darin, die Blöcke miteinander zu verketten, indem jeder Block die Kennung des Nachfolgeblocs enthält. Man muss sich für eine Datei einzig den Startblock merken sowie eine Liste der

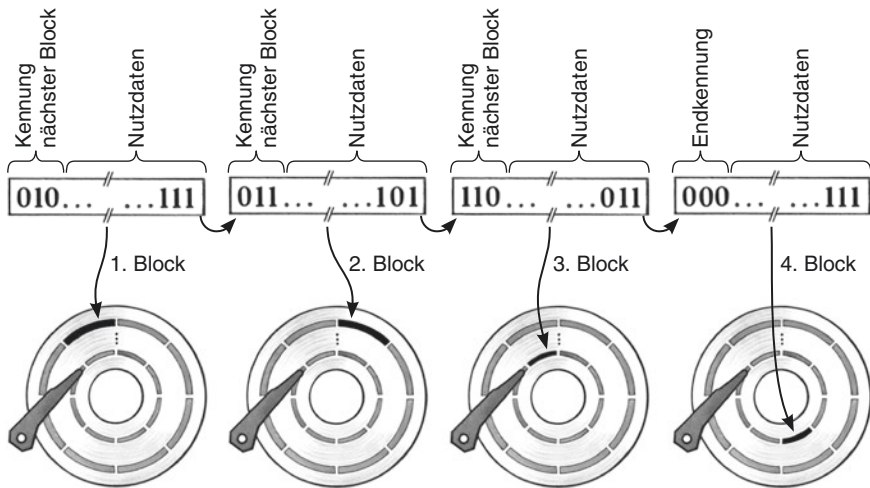


Abb. 5.6 Wie eine Datei auf einer Festplatte physikalisch repräsentiert werden kann. (© Susanne Staubli, Eckart Zitzler)

freien Blöcke (eine riesige leere Datei), aus dem man dann Blöcke herauschneiden kann, wenn sie benötigt werden. Dieses Prinzip ist in Abb. 5.6 illustriert. Des Weiteren ist auf speziell markierten Blöcken ein Inhaltsverzeichnis abgelegt, das die Namen der Dateien und den entsprechenden Anfangsblock enthält. Moderne Dateisysteme sind natürlich wesentlich ausgeklügelter, doch für unsere Zwecke soll dieses einfache Beispiel genügen.

Es gibt noch viele weitere Überlegungen, die beim Einsatz von Massenspeichern eine Rolle spielen. Beispielsweise können gelegentlich Fehler bei der Datenspeicherung auftreten, möglicherweise wegen Materialermüdung oder zerstörerischer Umwelteinflüsse. Aus diesem Grund werden häufig Maßnahmen ergriffen, um solche Fehler aufzufangen und zu erkennen. Anm. 5.3 zeigt auf, wie durch die Kombination mehrerer identischer Massenspeicher die Ausfallsicherheit erhöht werden kann: Wenn einer der Massenspeicher beschädigt ist, doch die anderen einwandfrei funktionieren, lassen sich auf diese Weise die gespeicherten Daten vollständig rekonstruieren.

Anmerkung 5.3: Fehler auffangen

Festplatten werden häufig miteinander kombiniert, um fehlerrobuste Speichersysteme – gerade für Backups – zu bauen. In Abb. 5.7 ist eine mögliche Variante eines Prinzips zu sehen, das sich RAID nennt (RAID steht für

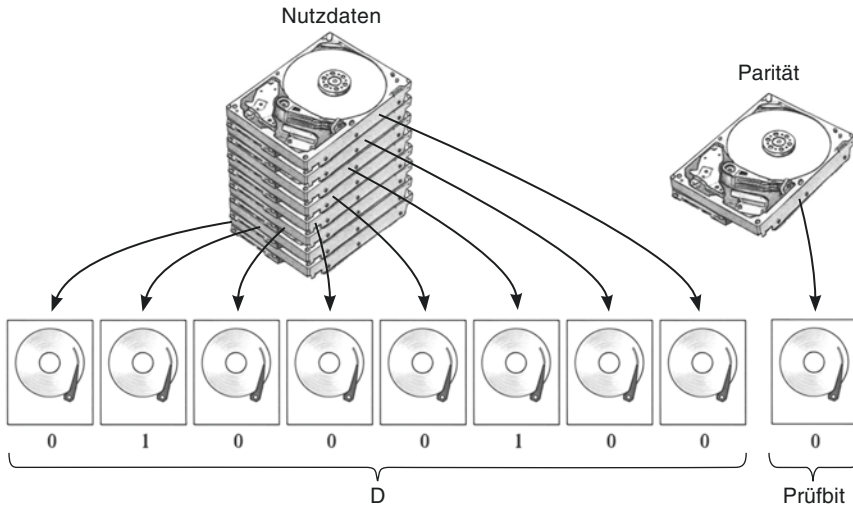


Abb. 5.7 Wie mehrere Festplatten benutzt werden können, um den Ausfall einer zu kompensieren. (© Susanne Staubli, Eckart Zitzler)

redundant array of independent discs, also eine Kopplung mehrerer unabhängiger Festplatten). Soll ein Byte gespeichert werden, so werden die einzelnen Bits auf die acht Festplatten verteilt: Das erste Bit kommt auf die erste Festplatte, das zweite Bit auf die zweite Festplatte usw. Schließlich gibt es noch eine neunte Festplatte; diese speichert ein Prüfbit, das Zusatzinformationen über das gespeicherte Byte enthält. Das Prüfbit ist gleich 0, wenn die Anzahl der Einsen im Byte gerade ist, und sonst eine 1. Diese Art der Zusatzinformation wird auch als Parität (Gleichheit) bezeichnet. Wozu das nützlich sein soll? Nun, wenn immer eine der Festplatten ausfällt, so lässt sich für jedes Byte das fehlende Bit einfach aus den restlichen acht rekonstruieren. Diese Fehlertoleranz haben wir uns erkaufte durch das „Verschwenden“ einer Festplatte zum Speichern der Parität.

Jetzt haben wir gesehen, wie sich Daten unter Verwendung eines Massenspeichers dauerhaft und sicher in einem Computersystem festhalten lassen. Die Permanentenspeicherung ist jedoch nur ein Aspekt, wofür Zusatzspeicher eingesetzt werden können. Eine weitere Anwendung besteht darin, die Speicherkapazität des Hauptspeichers zu erhöhen.

Die Eier legende Wollmilchsau

Nehmen wir einmal an, wir wollten viele, sehr datenintensive Programme gleichzeitig auf unserem Computer ausführen, sodass die Größe des

Hauptspeichers nicht mehr ausreicht. Natürlich könnten wir den Hauptspeicher schlicht ausbauen, aber weil hier möglichst schnelle Speicherbausteine zum Einsatz kommen, ist das recht teuer. Ein Massenspeicher wie eine Festplatte hingegen ist viel preiswerter, bietet wesentlich mehr Platz, ist dafür aber langsam. Das Beste beider Welten lässt sich geschickt kombinieren, sodass eine hohe Speicherkapazität bei moderaten Kosten zur Verfügung gestellt werden kann, ohne große Einbußen bei der Zugriffsgeschwindigkeit zu haben. Die Idee ist einfach: Wir nutzen letztlich die Festplatte als Hauptspeicher, wobei wir nur noch eine Umrechnung der Speicheradressen in entsprechende Orte auf der Festplatte einbauen müssen. Natürlich, ein Festplatten-Hauptspeicher wäre unerträglich langsam. Aus diesem Grund wird der richtige Hauptspeicher als Puffer verwendet, in dem Speicherbereiche, auf die der Prozessor gerade zugreift, zur schnellen Verwendung zur Verfügung gestellt werden. Weil der Hauptspeicher ja nur scheinbar größer ist, wird das Konzept auch virtueller Speicher genannt. Bsp. 5.3 erklärt, wie das konkret funktioniert.

Beispiel 5.3: Virtueller Speicher

Das Ziel: Wir machen kurzerhand aus der Festplatte (oder einem anderen Massenspeicher) den Hauptspeicher und aus dem Hauptspeicher einen Puffer. Dafür unterteilen wir Hauptspeicher und Festplatte in Datenblöcke, die Seiten genannt werden. Seiten sind die Austauscheinheiten zwischen Festplatte und Hauptspeicher. Das Prinzip wird in Abb. 5.8 illustriert. Die konkrete Umsetzung ist dann allerdings etwas komplizierter.

Der virtuelle Speicher stellt sich für ein Programm wie ein regulärer Hauptspeicher dar: Die einzelnen Speicherzellen werden über Adressen angesprochen. Allerdings sind diese Adressen virtuell, denn die effektiven Daten befinden sich ja in irgendwelchen Blöcken auf der Festplatte und sind irgendwo im regulären Hauptspeicher gepuffert. Aus diesem Grund muss bei jedem Speicherzugriff eine Übersetzung vorgenommen werden, von einer Adresse im virtuellen Speicher in eine konkrete Hauptspeicheradresse. Greift nun ein Programm auf eine virtuelle Adresse zu, passiert intern Folgendes: Ein spezieller Baustein, die *memory management unit* (MMU), prüft, ob die entsprechende Seite im Hauptspeicher vorhanden ist. Wenn nicht, lädt sie diese von der Festplatte in den Hauptspeicher; ggf. muss vorher Platz gemacht werden, indem eine andere Seite vom Hauptspeicher zurück auf den Massenspeicher transferiert wird. Anschließend rechnet sie die virtuelle Adresse um in eine konkrete physikalische Adresse, wo das Datum im Hauptspeicher effektiv liegt.

Und warum funktioniert das Ganze? Und lohnt sich der Aufwand? Ja, der Aufwand lohnt sich, und der Grund dafür ist das Prinzip der Lokalität. Häufig ist es nämlich so, dass der Prozessor auf einen eng eingegrenzten

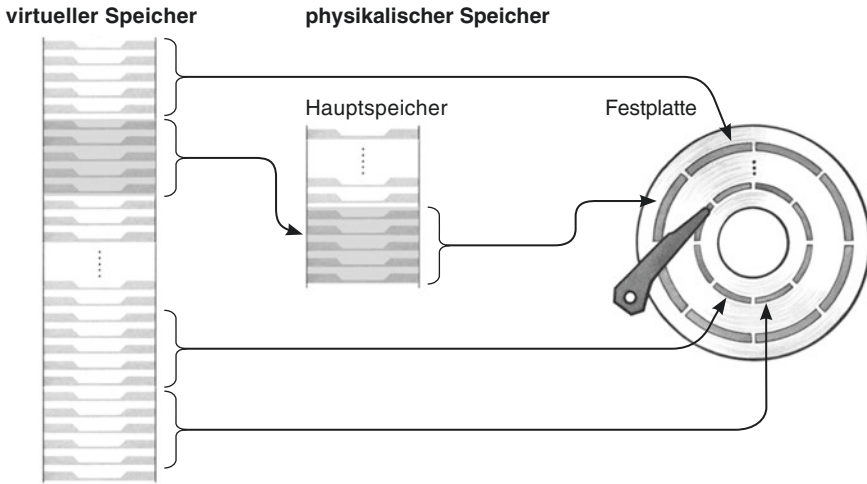


Abb. 5.8 Wie ein virtueller Speicher funktioniert. (© Susanne Staubli, Eckart Zitzler)

Speicherbereich zugreift, denn der Programmcode oder auch die Daten, die ein Programm bearbeitet, liegen ja in der Regel zusammenhängend in derselben Speicherregion. Und häufig ist es auch so, dass der Prozessor, wenn er ein Datum aus einer Speicherregion liest, in nächster Zeit auf ein benachbartes Datum zugreift. Es gibt also eine örtliche und eine zeitliche Lokalität. Der Aufwand, einen Datenblock von der Festplatte in den Hauptspeicher zu schaufeln (und auch wieder zurück), ist hoch, aber wenn der Prozessor den Datenblock mehrfach benutzt und die Austauschoperationen selten sind, funktioniert das Ganze recht gut. Die Grenzen dieses Ansatzes werden vor allem dann sichtbar, wenn man den virtuellen Speicher extrem auslastet – irgendwann macht sich der nachgelagerte, langsame Massenspeicher natürlich bemerkbar.

Die Idee lässt sich noch weiterspinnen. Indem wir vor einen langsamen Speicher einen schnelleren, kleineren Speicher einbauen, können wir die Zugriffszeit massiv reduzieren. Tatsächlich wird diese Beobachtung generell genutzt, um der Eier legenden Wollmilchsau, einem schnellen Speicher mit großer Kapazität, der Daten langfristig und robust speichert, näherzukommen. Das ist die gleiche Idee, die bereits unser Bücherwurm eingesetzt hat: Wir verwenden mehrere Speicher und kombinieren diese. Auf der einen Seite des Spektrums stehen die schnellen kleinen Speicher, auf der anderen Seite die langsamen großen Speicher und dazwischen Kompromisse beider Extreme. Was so entsteht, ist eine Speicherhierarchie, bei der die Register

das eine Ende des Spektrums darstellen, während die Massenspeicher das andere Ende repräsentieren. Wie so eine Speicherhierarchie im Gesamten aussehen kann, ist in Bsp. 5.4 erläutert.

Beispiel 5.4: Die Speicherhierarchie

Abb. 5.9 zeigt auf, wie so eine Speicherhierarchie konkret aussehen kann. Register, Hauptspeicher und Massenspeicher sowie deren Zusammenspiel kennen Sie schon, neu ist die Stufe des Cache. Ein Cache ist ein Zwischenspeicher auf dem Prozessor, der als Puffer zwischen Registern und Hauptspeicher fungiert. So ein Cache hat nur einen kleinen Prozentsatz der Kapazität des Hauptspeichers, kann aber viel mehr Daten aufnehmen als die Register. Und so wird es Sie auch nicht wundern, dass der Cache hinsichtlich der Zugriffsgeschwindigkeit genau in der Mitte liegt: Er verwendet in der Regel eine schnellere und teurere Technologie als der Hauptspeicher, kann aber aufgrund seiner Größe nicht die schnellen Zugriffszeiten von Registern bieten. Der Cache wird ähnlich eingesetzt wie der Hauptspeicher beim virtuellen Speicher: Er nimmt zusammenhängende Blöcke vom Hauptspeicher auf. Wann immer der Prozessor auf den Hauptspeicher zugreifen will, schaut er zunächst im Cache nach, ob der gewünschte Adressbereich dort vorhanden ist; falls nicht, muss er vom Hauptspeicher in den Cache geladen werden. Ein Cache kann die Zeit, die der Prozessor bei einem Hauptspeicherzugriff warten muss, erheblich reduzieren. Aus diesem Grund werden in der Regel sogar mehrere Cache-Stufen eingesetzt.

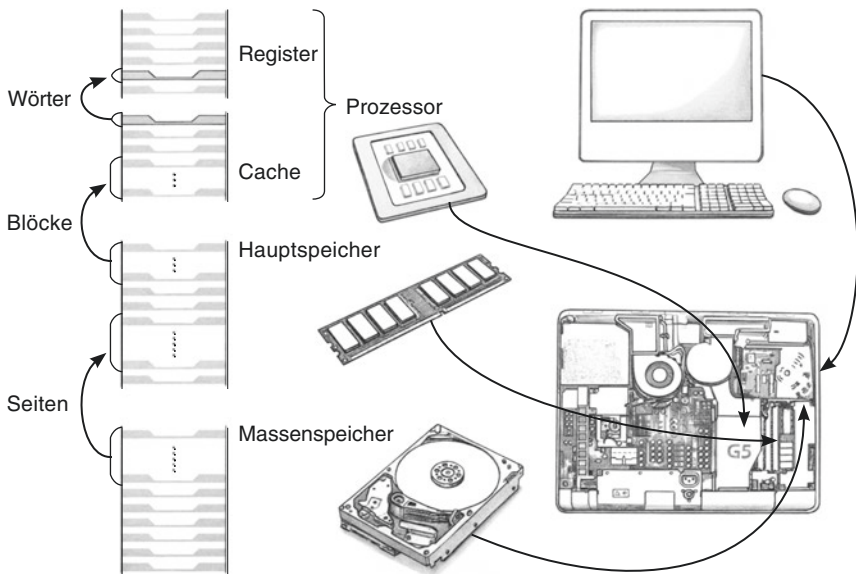


Abb. 5.9 Wie eine einfache Speicherhierarchie aussieht. (© Susanne Staubli, Eckart Zitzler)

Übrigens ist mit dem virtuellen Speicher die Speicherhierarchie noch nicht an ihrer Spitze angekommen, mit dem Internet haben sich weitere Stufen etabliert. Nicht nur, dass es große Speicherfarmen gibt, auf denen die Daten vieler Nutzer lagern. Eigentlich bilden alle Speichersysteme der Welt, die im Internet zusammengeschlossen sind, einen riesigen Speicher, der wiederum ganz anders organisiert ist als die altbekannten Speichersysteme (mehr dazu in Kap. 6). Die Tendenz, Informationen extern abzulegen und abzuspeichern, ist natürlich nichts Neues, wenn wir wieder auf unseren Bücherwurm zurückkommen und an die damalige Stadtbibliothek denken. Den Bibliotheken entsprechen heute die riesigen Speicherfarmen, wir sind also unserer Kulturtechnik treu geblieben.

Durch die Kombination verschiedener Speichertechnologien lassen sich also Vorteile kombinieren und Nachteile kompensieren. Neue Speichertechnologien wie beispielsweise Flash-Speicher können Lücken teilweise füllen, doch der Unterschied zwischen Verarbeitungsgeschwindigkeit von Prozessoren und der notwendigen Speicherkapazität wird wohl immer bestehen (siehe Anm. 5.4). Der Grund ist der, den wir bereits beim Bücherwurm gesehen haben: Der Platz ist beschränkt – nicht alles, was nah sein sollte, kann auch nah sein.

Anmerkung 5.4: Das Mooresche Gesetz

Gordon Moore, ein Mitbegründer des in der Computergeschichte eine tragende Rolle spielenden Halbleiterherstellers Intel, prognostizierte 1965, dass sich – vereinfacht formuliert – die Anzahl der Komponenten auf einem integrierten Schaltkreise bzgl. einer fixen Fläche und minimaler Produktionskosten alle ein bis zwei Jahre verdoppelt. Und weil er mit seiner Vorhersage so gut lag – bis 1975 traten seine Vorhersagen sehr präzise ein –, wurde seine Aussage bald verallgemeinert und als Mooresches Gesetz zu einem Leitsatz der Industrie. Man übertrug es beispielsweise auch auf die Dichte und Kapazität von Speicherbausteinen oder die Übertragungsgeschwindigkeit von Netzwerkverbindungen. Doch an sich handelt es sich hierbei weder um ein Gesetz noch um eine zwingende Entwicklung, vielmehr um eine Beobachtung, die später aufgrund von Markphänomenen wie eine selbsterfüllende Prophezeiung erstaunlich präzise eintraf. Moore war zwar etwas zu optimistisch, doch im Wesentlichen hat sich die Anzahl an Transistoren, die auf einem Stück Silikon gefertigt werden können, ungefähr alle zwei Jahre verdoppelt. Das lässt sich zwar nicht unmittelbar auf die Speicherkapazität übertragen und zudem kann dieses Wachstum nicht ewig so weitergehen, doch es illustriert, welcher Dynamik das Gebiet der Informationstechnologie unterliegt. Die Zunahme der Speicherkapazität ging jedoch nicht einher mit einer proportionalen Zunahme der Zugriffsgeschwindigkeit. Diese hat sich ungleich langsamer gesteigert, während Prozessoren wesentlich schneller wurden. Das wiederum hat das Problem langsamer Speicher bei schnellem Prozessor verschärft, und genau aus diesem Grund hat man sich Lösungen überlegt, um Zugriffsgeschwindigkeit und Speicherkapazität zu optimieren.

Das waren die Möglichkeiten auf der Hardware-Seite, um möglichst schnell auf viel Speicher zugreifen zu können. Dem eigentlichen Programm bleiben vieler dieser Mechanismen jedoch verborgen, es kann Register und Hauptspeicher manipulieren, doch von Caches und virtuellem Speicher bekommt es nichts mit. Aber auch die Software-Ebene spielt eine gewichtige Rolle, wenn es darum geht, mit großen Datenmengen schnell hantieren zu können.

Ordnung muss sein: den Daten ihre Struktur

Ein Programm muss zu jedem Zeitpunkt wissen, wo die einzelnen Zwischenergebnisse abgelegt und welche Speicherzellen im Hauptspeicher unbelegt sind. Aus diesem Grund überlegt sich ein Programmierer im Vorfeld Ablagestrukturen, die helfen, den zur Verfügung stehenden Speicher zu organisieren und gezielt Informationen wiederzufinden. Denn es ist nicht egal, wie die Daten im Hauptspeicher abgelegt werden; je nach Anwendung sind einige Techniken der Datenablage besser und andere weniger geeignet. Dabei geht es einerseits darum, die Daten platzsparend abzuspeichern, und andererseits darum, schnell auf Daten zugreifen zu können.

Es gibt verschiedene Konzepte, wie man Daten strukturieren kann, und bezeichnet diese – wen wundert’s – allgemein als Datenstrukturen. Datenstrukturen helfen Daten zusammenzuhalten und stellen eigentlich Zusatzinformationen dar, die etwas über die Beziehung zwischen den ursprünglichen Daten aussagen. Datenstrukturen sind sozusagen Metadaten, die ggf. ganz anders aufbewahrt werden als die Originaldaten, um die es geht. Das klingt vermutlich noch etwas nebulös, doch der Nebel wird sich später lichten, wenn wir konkrete Methoden anhand der in Bsp. 5.5 skizzierten Anwendung studieren.

Beispiel 5.5: Noah als Datensammler

Am besten lassen sich die Datenstrukturen in der Informatik, welche Ideen ihnen zugrunde liegen und wie sie konkret funktionieren, anhand eines Beispiels verstehen. Stellvertretend für diverse Anwendungsszenarien betrachten wir hier das Problem des systematischen Erfassens von Daten über Individuen bei der Einreise, Einschiffung, aber auch beim Entdecken der Lebenswelt einer Insel etc., hier in Anlehnung an die Geschichte von der Arche Noah sinnbildlich (siehe Abb. 5.10). Es geht darum, von jedem Tierpaar einige zentrale Daten wie Name, Gattung, durchschnittliche Größe, durchschnittliches Gewicht und durchschnittliche Lebenserwartung aufzunehmen und akribisch zu erfassen. Sagen wir, diese Informationen seien einem Pass gleich

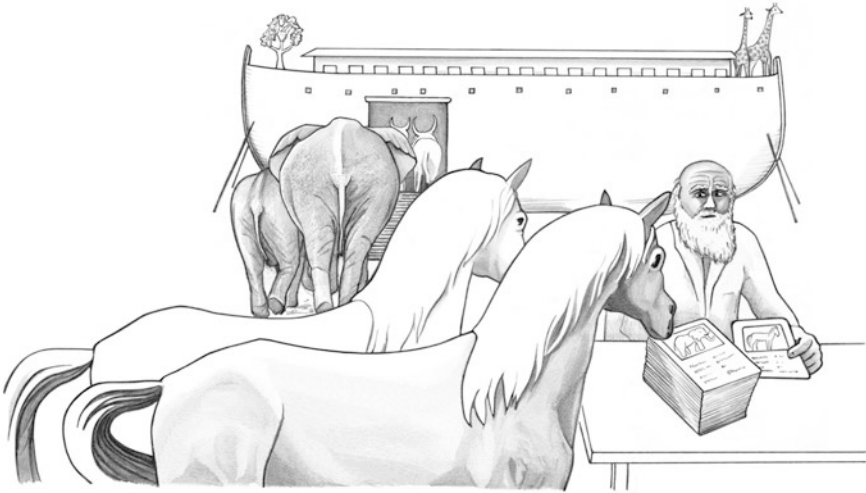


Abb. 5.10 Die Arche Noah. (© Susanne Staubli, Eckart Zitzler)

auf einer Karte schriftlich festgehalten. Jedes Tierpaar gibt beim Betreten des Schiffes eine Karte ab. Die Frage ist nun, wie ein Programm die Informationen auf den Karten im Hauptspeicher ablegen kann, sodass auf sie einfach wieder zugegriffen werden kann.

Im Folgenden werden wir mit einfachen Grundstrukturen beginnen und anschließend Erweiterungen kennenlernen, über die Flexibilität und Zugriffsgeschwindigkeit gesteigert werden können. Dazu gehen wir davon aus, dass dem Programm ein bestimmter Bereich im Hauptspeicher zur Verfügung steht, den es zur Datenspeicherung verwenden und beliebig organisieren kann.

Von Stapeln und Feldern

Wenden wir uns also Noahs Problem zu, die Angaben auf mehreren Karten im Hauptspeicher abzulegen. Dabei müssen wir zwei Fragen beantworten: Wie legen wir eine einzelne Karte im Speicher ab und wie ordnen wir mehrere Karten relativ zueinander im Speicher an? Ein naheliegender Ansatz ist: alles hintereinander. Für jeden Eintrag auf einer Karte sehen wir eine bestimmte Anzahl von Bytes vor, und diese Datenblöcke werden in der vorgegebenen Reihenfolge im Hauptspeicher abgelegt. Das ist bereits eine Datenstruktur. Jede Karte beansprucht in dieser Variante einen stets gleich

großen Speicherbereich. Und mehrere Karten legen wir nach dem gleichen Schema hintereinander im Hauptspeicher ab. Das entspräche der Situation, wenn wir die Karten einfach übereinander als Stapel anordnen würden – quasi der Bücherstapel auf dem Schreibtisch des Bücherwurms. Diese zweite, übergeordnete Datenstruktur nennt sich auch Stapel und ist in Bsp. 5.6 illustriert.

Beispiel 5.6: Daten stapeln

In Abb. 5.11 sehen Sie eine Möglichkeit, wie die Kartendaten im Hauptspeicher abgelegt werden können. Für jede Karte ist ein Speicherbereich von 1000 Bytes reserviert, die Karten sind beginnend mit der Speicheradresse 1000 übereinander angeordnet. Das heißt, die erste Karte liegt im Speicherbereich 1000 bis 1999, die zweite im Speicherbereich 2000 bis 2999, die dritte im Speicherbereich 3000 bis 3999 usw. Was die einzelnen Informationen auf einer Karte anbetrifft, so wird für jeden Eintrag eine fixe Anzahl von Bytes reserviert, die Reihenfolge der Einträge ist ebenfalls fixiert. Zuerst kommen 50 Bytes für den Namen, dann 50 Bytes für die Gattung, anschließend 2 Bytes für das Gewicht, dann 2 Bytes für die Größe und schließlich 2 Bytes für das Alter; darüber könnten noch weitere, hier nicht illustrierte Daten abgelegt werden. Die 50 Bytes codieren jeweils eine Zeichensequenz, jedes Zeichen ist wiederum als eine 8-stellige Dualzahl codiert. Die 2 Bytes enthalten jeweils eine 16-stellige Dualzahl.

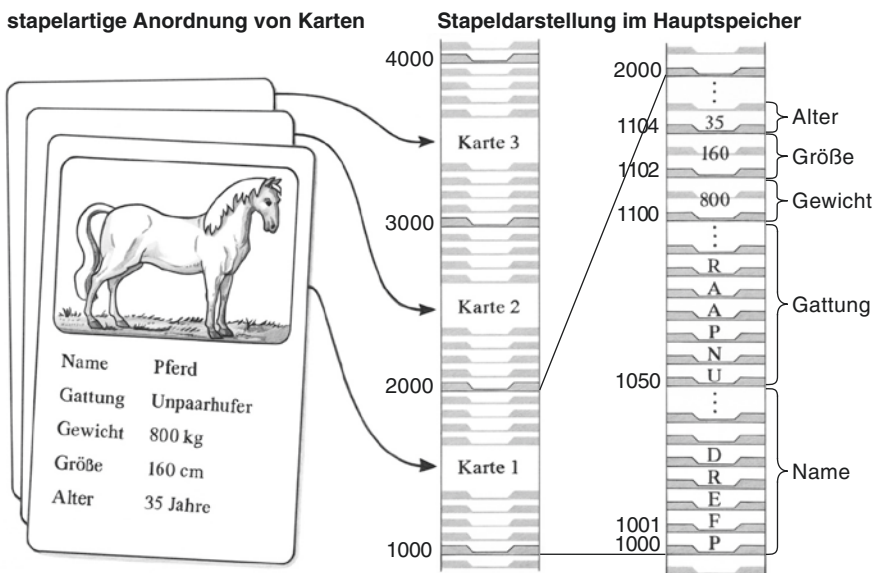


Abb. 5.11 Der Stapel als eigenständige Datenstruktur und seine Realisierung auf dem Computer. (© Susanne Staubli, Eckart Zitzler)

Die Stapel-Datenstruktur ist die denkbar einfachste und sparsamste Speichervariante, an Zusatzinformationen müssen wir uns einzig die erste freie Speicheradresse, sozusagen die Spitze des Stapels, merken. Ganz nebenbei speichert diese Datenstruktur noch die Reihenfolge ab, in der die Karten hinzugefügt wurden – schließlich liegen die Kartendaten ja übereinander im Hauptspeicher. Die räumliche Anordnung im Speicher spiegelt also eine zeitliche Anordnung wider.

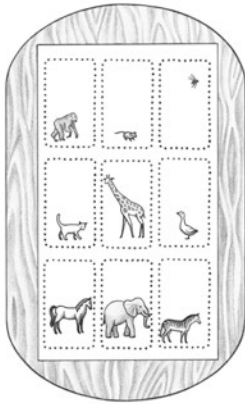
Genauso gut kann aber auch eine effektive räumliche Anordnung die Organisation der Daten im Speicher bestimmen. Nehmen wir z. B. an, das Unterdeck der Arche bestände aus mehreren gleich großen Gehegen, die schachbrettartig in einem Raster angeordnet sind. Wir möchten gerne die Daten so abspeichern, dass wir sofort eruieren können, in welchem Gehege welches Tierpaar untergebracht ist. Mit den Karten lässt sich das einfach realisieren: Wir legen die Karten in der gleichen Anordnung wie die Gehege auf den Tisch. Um die gleiche Struktur auch im Computer anlegen zu können, bedarf es eines kleinen Kniffs, weil der Hauptspeicher ja ein eindimensionales Gebilde ist und keine zweidimensionale Anordnung zulässt. Der Trick besteht darin, erst die Karten in der unteren Reihe hintereinander im Speicher abzulegen, dann die der zweiten Reihe und zum Schluss, ganz oben, die der dritten Reihe. Bsp. 5.7 erläutert die Details.

Beispiel 5.7: Daten in Reih und Glied

Wenn wir eine mehrdimensionale Anordnung von Karten im eindimensionalen Hauptspeicher darstellen wollen, so heißt das letztlich: Wir müssen eine Reihenfolge festlegen, in der die Karten abgelegt werden sollen. In dem in Abb. 5.12 skizzierten Szenario gehen wir von links nach rechts und zudem von unten nach oben. Wenn wir die Reihe und die Spalte einer Karte kennen, können wir daraus direkt den entsprechenden Speicherort ermitteln. Die Adresse einer konkreten Karte ergibt sich, wenn wir die Reihenummer (die Nummerierung beginnt wie beim Hauptspeicher bei 0) mit der Größe einer Reihe multiplizieren anschließend die Spaltennummer, multipliziert mit der Größe einer Karte (1000 Bytes), addieren und das Ergebnis zur Startadresse (1000) hinzuzählen. Die Maus liegt in Reihe 2 (dritte Reihe) und Spalte 1 (zweite Spalte), entsprechend sind die Mausdaten an der Adresse 8000 (1000 plus 2 mal 3000 plus einmal 1000) abgelegt.

Der Vorteil dieser Datenstrukturierung liegt auf der Hand: Das zugeordnete Gehege ist explizit mit repräsentiert, die Position im Speicher ist mit der Anordnung der Gehege verknüpft. Ansonsten, z. B. wenn wir eine einfache Stapel-Datenstruktur verwenden, müssten wir zu jeder Karte auch noch das Gehege in geeigneter Codierung abspeichern. Datenstrukturen können

Schiffsdeck mit Gehegen



gitterartige Anordnung von Karten

 Name Schimppanse Gattung Menschenaffe Gewicht 65 kg Größe 75 cm Alter 15 Jahre	 Name Maus Gattung Nagetier Gewicht 25 g Größe 10 cm Alter 2,5 Jahre	 Name Fruchtfliege Gattung Zweiflügler Gewicht 2 g Größe 2 mm Alter 1 Tag
 Name Katze Gattung Felis Gewicht 5 kg Größe 30 cm Alter 15 Jahre	 Name Giraffe Gattung Paarhufer Gewicht 1400 kg Größe 500 cm Alter 25 Jahre	 Name Gans Gattung Vogel Gewicht 8 kg Größe 40 cm Alter 5 Jahre
 Name Pferd Gattung Unpaarhufer Gewicht 800 kg Größe 160 cm Alter 35 Jahre	 Name Elefant Gattung Rüsseltier Gewicht 6000 kg Größe 450 cm Alter 75 Jahre	 Name Zebra Gattung Unpaarhufer Gewicht 250 kg Größe 120 cm Alter 35 Jahre

Felddarstellung
im Hauptspeicher

9000	Fruchtfliege
8000	Maus
7000	Schimppanse
6000	Gans
5000	Giraffe
4000	Katze
3000	Zebra
2000	Elefant
1000	Pferd

Abb. 5.12 Eine zweidimensionale Feld-Datenstruktur. (© Susanne Staubli, Eckart Zitzler)

also Zusatzinformationen implizit abspeichern. Doch es geht auch darum, Daten möglichst schnell wiederzufinden. Wenn wir zu einem Gehege wissen wollen, welche Tierpaar dort untergebracht ist, können wir aus der Position unmittelbar die Speicheradresse der zugehörigen Kartendaten ermitteln und direkt darauf zugreifen. Das entspricht den angeschriebenen Ablagekörben auf dem Schreibtisch des Bücherwurms. Sind die Kartendaten wie bei einem Stapel in der Reihenfolge ihres Eintreffens im Hauptspeicher abgelegt, so müssten wir alle Karten durchgehen und nach dem gespeicherten Gehege suchen, das würde viel länger dauern.

In der Fachsprache nennt man so eine gitterartige Datenstruktur ein *Feld* oder auf Englisch *array*. Ein Feld stellt nicht anderes als eine Tabelle dar, kann auch nur eine Reihe beinhalten oder aber drei und mehr Dimensionen aufweisen. Es ist definiert über seine Ausmaße, seine Lage im Hauptspeicher und die Größe der enthaltenen Elemente; diese Angaben genügen, um auf jedes Element über seine Reihen- und Spaltennummer – im zweidimensionalen Fall – zugreifen zu können.

Vielleicht kommt Ihnen in diesem Zusammenhang ein Koordinatensystem in den Sinn, wie es uns auf geografischen Karten begegnet (Abb. 5.13).



Abb. 5.13 Das Koordinatensystem teilt eine Weltkarte in Felder ein (links), so entsteht ein Raster wie bei einem Schubladenregal (rechts). (© cartarium/stock.adobe.com (links), Deyan Georgiev/stock.adobe.com (rechts))

Indem wir beispielsweise einer Wanderkarte ein Raster überstülpen, teilen wir sie in viele kleine Teilkarten auf (genau genommen ist es natürlich der Kartenhersteller). Suchen wir einen uns unbekannten Ort und kennen seine Koordinaten, so können wir ihn in kurzer Zeit auf der entsprechenden Teilkarte finden – ansonsten wird es schnell zu einem Gestocher im Heuhaufen. Hier erkennen wir also das Prinzip des Feldes wieder, der schnelle Zugriff macht es zur ersten Wahl, wann immer sich die Daten in ein Koordinatensystem einfügen lassen.

Der Baum der Sucherkenntnis

Wir haben die Feld-Datenstruktur soeben eingesetzt, um einzelne Karten gezielt zu suchen – über die Gehegeposition. Doch was ist, wenn wir nach einem anderen Kriterium suchen, z. B. nach dem Namen? Dann bliebe uns nichts anderes übrig, als den ganzen Datenbestand durchzugehen und nach dem angegebenen Namen zu suchen – das dauert lange und ist aufwändig. Es geht jedoch schneller, und zwar indem wir wiederum die Anordnung der Kartendaten im Speicher an dem Suchkriterium ausrichten: in diesem Fall so, dass die Karten in alphabetischer Reihenfolge im Speicher abgelegt werden. Dieser Trick war ja schon zu Zeiten des Bücherwurms weit verbreitet: Die Einträge in Lexika waren nach Namen sortiert, deswegen konnte man in den zumeist voluminösen Schinken überhaupt etwas finden. Es gibt sogar eine Methode, wie man am besten vorgeht und ausgehend von der Mitte schnell zum gewünschten Eintrag kommt, wie Bsp. 5.8 zeigt. Sie nennt sich binäre Suche.

Beispiel 5.8: Schnelles Suchen in sortierten Feldern

Nehmen wir mal an, wir hätten sieben Karten bereits eingesammelt und in einem eindimensionalen Feld im Hauptspeicher so abgelegt, dass die Kartenreihenfolge der alphabetischen Reihenfolge der Namen entspricht. Diese Situation ist in Abb. 5.14 angedeutet, wobei der Übersichtlichkeit halber die konkrete Speicherbelegung nicht eingezeichnet ist und die Karten nebeneinander anstatt übereinander dargestellt sind.

Jetzt wollen wir wissen, ob das Gänsepaar bereits an Bord ist. Dazu beginnen wir unsere Suche beim Element in der Mitte der Kartenreihe. Dort ist die Karte zur Giraffe gespeichert. Weil die Reihe sortiert ist und Gans alphabetisch vor Giraffe kommt, wissen wir nun, dass die Karte links von der Giraffe liegen muss. Dann nehmen wir uns die mittlere Karte der linken Hälfte vor, hier ist die Fruchtfliegen-Karte zu finden. Falls die Gans-Karte bereits erfasst ist, so muss sie rechts von der Fruchtfliege liegen. Die verbleibende Hälfte können wir wieder aufteilen in zwei Teile und uns dabei auf die rechte Unterhälfte beschränken. Hier ist nur noch eine Karte vorhanden, nämlich die Gans. Die Suche ist beendet. Der Vorteil dieses Suchverfahrens ist, dass sich sieben Elemente in der Reihe befinden, aber maximal drei Karten angeschaut werden müssen, um einen bestimmten Namen zu finden (oder festzustellen, dass er nicht vorhanden ist).

Indem wir die Karten in der Reihenfolge ihrer Namen in einem Feld ablegen und dann die binäre Suche anwenden, können wir Kartendatensätze schnell über den Namen finden. Das Gleiche lässt sich auch für andere Kriterien, z. B. die Größe, durchführen. Allerdings müssen wir die Karten vor der Suche sortieren – sie können ja in beliebiger Reihenfolge eintrudeln –, und das ist mit Aufwand verbunden (wie sich die Elemente in einem Feld sortieren lassen, erfahren Sie in Kap. 8). Das gilt insbesondere, wenn immer

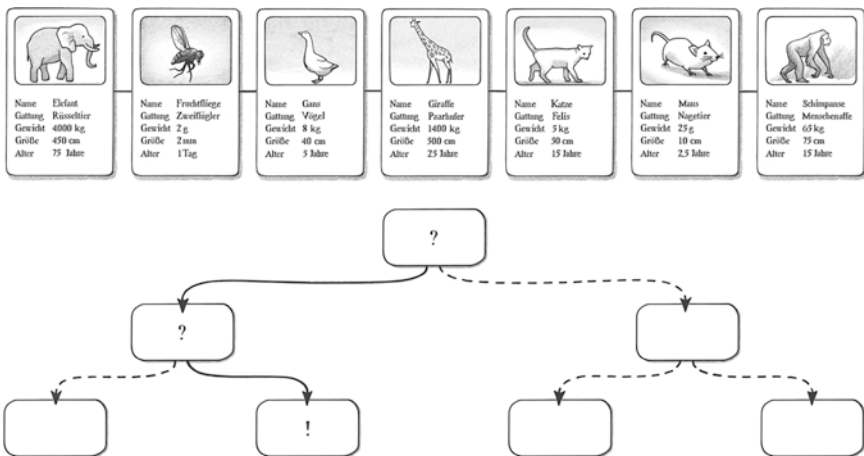


Abb. 5.14 Das Prinzip der binären Suche. (© Susanne Staubli, Eckart Zitzler)

wieder abwechselnd neue Karten hinzukommen und nach Karten gesucht wird; dann sind viele Sortiervorgänge notwendig.

Diese Überlegungen führen schließlich zur Idee, die Karten im Feld stets sortiert zu halten und neue Karten direkt an der korrekten Position einzufügen. Wir schauen, wo eine Karte im Speicher hingehören würde, und verschieben dann den Rest der Karten, sodass die neue hineinpasst. Es geht jedoch noch eleganter und ohne das Verschieben von Karten, wenn wir das gleiche Vorgehen wie bei der binären Suche auch beim Einfügen anwenden. Wie das konkret funktioniert? Wir starten mit einem leeren Feld und markieren alle Elemente als unbenutzt. Kommt eine Karte hinzu, führen wir die binäre Suche durch, bis wir auf ein unbenutztes Element stoßen, und legen die Kartendaten anschließend an dieser Speicherstelle ab.

Es gibt sogar eine Datenstruktur, die das Vorgehen der binären Suche direkt in eine räumliche Anordnung umsetzt. Dabei bildet das Kartenelement in der Mitte die Basis, links und rechts davon sind die Kartenelemente angeordnet, die in der Mitte der linken bzw. rechten Hälfte liegen. Letzteren sind dann nach dem gleichen Prinzip die restlichen Karten angehängt, wie es in Abb. 5.15 illustriert ist. An sich handelt es sich also

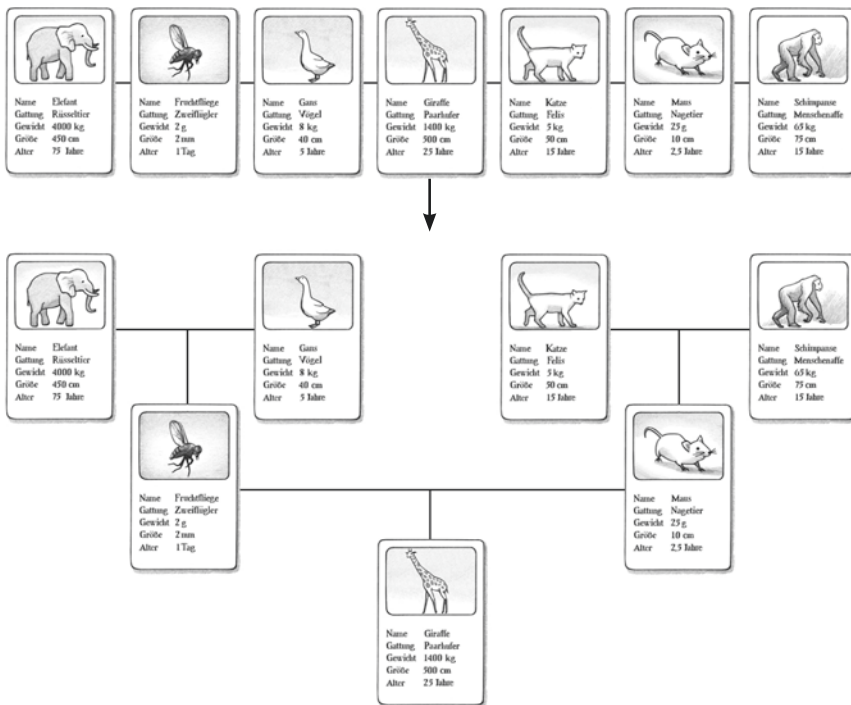


Abb. 5.15 Überführung einer sortierten Kartenreihe in eine baumartige Struktur. (© Susanne Staubli, Eckart Zitzler)

nur um eine andere Darstellung einer sortierten Kartenreihe. Diese Art der Datenstrukturierung wird als Baum bezeichnet, weil die Anordnung der Datensätze mit etwas Fantasie an einen Baum erinnert, der sich ausgehend vom Stamm immer weiter aufteilt und bis hin zu den Blättern verästelt. Bsp. 5.9 erläutert, wie sie im Hauptspeicher realisiert wird und wie darin effizient gesucht werden kann.

Beispiel 5.9: Ein Baum voller Daten

In Abb. 5.16 sehen Sie auf der linken Seite, wie verschiedene Vogelarten gemäß einer Baumstruktur angeordnet werden können; auf der rechten Seite ist die entsprechende Organisation im Hauptspeicher dargestellt. Die Übersetzung der Kartenstruktur in die Speicherstruktur ist analog zum Beispiel mit den Gehegen und der Feld-Datenstruktur. Wir gehen von links nach rechts und von unten nach oben, um die Karten im Hauptspeicher abzulegen.

Um eine bestimmte Karte anhand ihres Namens zu suchen, starten wir beim untersten Element, an das alle weiteren Elemente angehängt sind. Dieses wird auch Wurzel genannt. Wir vergleichen den gesuchten Namen mit dem Namen des Wurzelements: Sind sie identisch, sind wir am Ziel; kommt Ersterer vorher im Alphabet, so führen wir die Suche im linken Wurzelnachfolger weiter; kommt Ersterer später im Alphabet, geht es beim rechten Wurzelnachfolger weiter. Wenn wir bei einem Element in der obersten Reihe angekommen und nicht fündig geworden sind, wird die Suche abgebrochen. Wie gesagt: Das Vorgehen ist dasselbe wie bei der binären Suche.

In dem Zusammenhang stellt sich noch die Frage, wie die Speicheradressen der Nachfolgeelemente berechnet werden können. Das geht im Prinzip wie beim zweidimensionalen Feld über die Reihen- und die Spaltennummer, nur dass wir mitberücksichtigen müssen, dass die Reihen unterschiedliche Größen aufweisen. Denn im Unterschied zur Gitteranordnung verdoppelt sich die Kartenzahl bei jeder nachfolgenden Reihe; zuerst ist es eine Karte, anschließend werden es 2, 4 und 8.

Ein Baum ist ein Beispiel für eine rekursive Datenstruktur, also eine Struktur, die sich selbst enthält. Er besteht jeweils aus einem Wurzelement und einem linken sowie einem rechten Teilbaum. Die Teilbäume sind jeweils genauso aufgebaut – mit einer Wurzel und weiteren Teilbäumen. Wie in der Natur sind Bäume keinesfalls immer so perfekt ausgeglichene und symmetrische Gebilde, wie es im vorherigen Beispiel zu sehen war. Es kann durchaus sein, dass ein Element nur einen Nachfolger hat und die Äste verschieden hoch sind. Aus diesem Grund muss man sich bei dieser

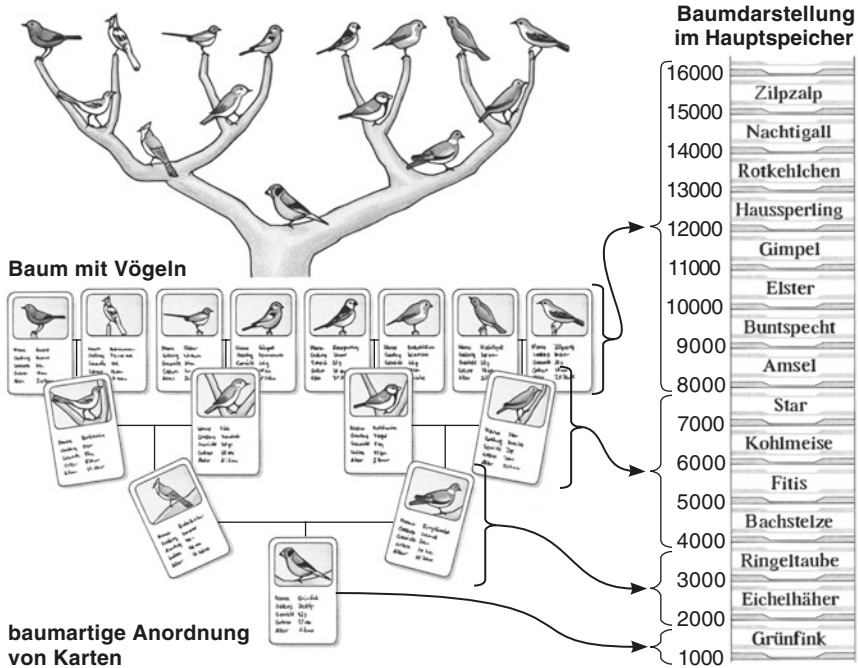


Abb. 5.16 Illustration einer Baum-Datenstruktur. (© Susanne Staubli, Eckart Zitzler)

Datenstruktur nicht nur die Position der Wurzel merken, sondern auch pro Element wissen, ob und wie viele Nachfolger vorhanden sind. Ich werde Ihnen gleich eine Methode vorstellen, wie sich dies leicht bewerkstelligen lässt; in dem Zusammenhang zeige ich Ihnen auch, wie sich Elemente hinzufügen lassen.

Aufgrund ihrer Vorteile – die Datensätze sind stets sortiert angeordnet und somit schnell durchsuchbar – sind Bäume in der Informatik ein äußerst beliebtes und vielfach verwendetes Konzept. Die Binärbäume, die sie soeben kennengelernt haben und bei denen jedes Element maximal zwei Nachfolger haben kann, sind nur eine Spielart. Insbesondere gibt es Varianten, die auf eine möglichst ausgeglichene Baumstruktur abzielen. Denn die Voraussetzung für eine schnelle Suche ist, dass der Baum eine möglichst flache Struktur aufweist. Einzelne, in die Höhe schießende Äste können den Geschwindigkeitsgewinn gegenüber einem unsortierten Feld zunichtemachen.

Solche sich verästelnde Strukturen kommen nicht nur in der Natur häufig vor (Abb. 5.17), sondern auch im Alltag. Es gibt viele grafische, den



Abb. 5.17 Ein Baum in der Natur (links) und das baumartige Gefäßsystem eines Blatts (rechts). (© Eckart Zitzler (links), Judith Büsser (rechts))

natürlichen Namensgebern nachempfundene Darstellungen – denken wir nur an Organigramme –, und die gab es schon lange vor den Computern. Ein Stammbaum ist ein Beispiel dafür: Am Anfang stehen die Urahnen, dann gibt es für jedes Kind (mit späterem Partner) einen weiteren Ast usw. Wenn jedes Paar genau zwei Kinder hätte, dann sähe der Baum aus wie das Gebilde bei der binären Suche.

Zeige mir deinen Nächsten

Bislang hatten wir uns darauf konzentriert, die Karten der Tierpaare in eine geschickte Anordnung zueinander zu bringen und die Anordnung auf den Hauptspeicher zu übertragen. Den Aspekt des Kommens und Gehens hatten wir dabei weitestgehend ausgeblendet. Häufig sind Datenstrukturen aber nicht statisch, sondern präsentieren sich als dynamische Gebilde, zu denen permanent neue Elemente hinzugefügt und aus denen bestehende entfernt werden. Für diese Zwecke sind die Ansätze, die ich Ihnen bis hierher vorgestellt hatte, nur teilweise geeignet. Soll z. B. das Wurzelement in einem Baum gelöscht werden, so muss ein anderes Element dorthin verschoben werden – 1000 Bytes sind zu kopieren, bei größeren Datensätzen kann der Aufwand wesentlich höher sein. Es gibt jedoch eine Methode, solche Verschiebungen im Hauptspeicher zu umgehen.

Ordnen wir die Karten nach einem bestimmten Schema an, dann heißt das nichts anderes, als dass wir die Karten in eine bestimmte Beziehung zueinander setzen. In Abb. 5.12 und 5.16 sind diese Beziehungen als Linien visualisiert. In einem Feld ist ein Element Teil eines Gitters, befindet sich in einer Reihe und hat – mit Ausnahme der Randelemente – zwei Nachbarn. In einem Binärbaum hat ein Element einen Vorgänger (mit Ausnahme der

Wurzel) und keinen, einen oder zwei Nachfolger. Die Idee ist nun, diese Beziehungen explizit zu machen und im Hauptspeicher mit abzuspeichern. Nur wie? Ich erkläre Ihnen das Prinzip am Beispiel des Binärbaums.

Jeder Datensatz ist über die Speicheradresse identifizierbar, an der er im Hauptspeicher abgelegt ist. Und diese Speicheradresse verwenden wir nun als Referenz. Zu jedem Element unseres Binärbaums fügen wir also zwei Einträge hinzu, nämlich die Adresse des linken und die Adresse des rechten Nachfolgers. Und wenn kein Nachfolger vorhanden ist? Dann setzen wir die Adresse 0 ein; dabei handelt es sich um eine Konvention, um auszudrücken, dass die Adresse nicht auf ein weiteres Element verweist. Bsp. 5.10 zeigt, wie das im Detail aussieht. In diesem Fall müssen wir also einzig die Speicheradresse der Wurzel kennen, und wir können den Baum durchsuchen und verändern.

Beispiel 5.10: Mit Zeigern einen Baum bauen

In Abb. 5.18 oben sehen Sie, wie ein Element eines Binärbaums erweitert wird, sodass die beiden Nachfolger explizit aufgeführt sind. Der untere Teil der Abbildung demonstriert, wie mit dieser Elementestruktur der Baum Schritt für Schritt aufgebaut wird. Was genau passiert dort?

Am Anfang ist der Baum leer. Wird nun die Karte für den Grünfink im ersten Schritt an der Speicheradresse 1000 hinzugefügt, so hat der Baum eine Wurzel. Das Wurzelement besitzt jedoch noch keine Nachfolger. Um dies auszudrücken, wird wie erwähnt eine Konvention benutzt: Die Speicheradresse 0 bedeutet, dass kein Nachfolger existiert. Nun kommt die zweite Karte für den Eichelhäher hinzu. Da sie gemäß dem Namen alphabetisch vor der Grünfink-Karte anzusiedeln ist, wird diese Karte als linker Nachfolger der Wurzel eingefügt. Die Eichelhäher-Karte wird im Hauptspeicher oberhalb der ersten Karte abgelegt (sie könnte aber auch ganz woanders liegen) und die Speicheradresse 2000 wird in das erste Wurzelement als Referenz zum linken Nachfolger hinzugefügt. Die dritte Karte (Ringeltaube) wird gemäß der alphabetischen Sortierung im rechten Teilbaum von der Wurzel abgelegt, entsprechend wird das Wurzelement aktualisiert und die Speicheradresse 3000 als Referenz zum rechten Nachfolger eingetragen. Anhand der vierten Karte können Sie noch einmal das Einfügeschema studieren. Wo wird sie angehängt? Dazu traversieren wir den Baum von der Wurzel her und verzweigen nach links, wenn der Name des Elements alphabetisch vor „Fitis“ liegt, ansonsten nach rechts. Sobald wir am Ende eines Astes angelangt sind, haben wir die Einfügeposition gefunden: Die Fitis-Karte wird als rechter Nachfolger der Eichelhäher-Karte angehängt.

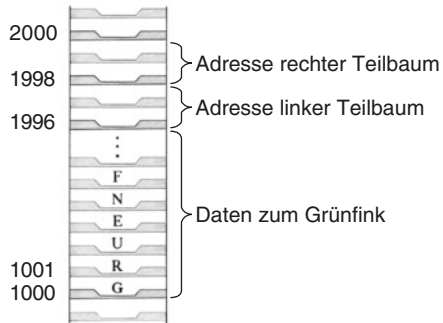
So wie wir mit dem Finger auf einen Gegenstand deuten, zeigen die Datenelemente über die Adressen auf ihre Nachfolger. Deswegen werden die Referenzen auch Zeiger genannt, im Englischen *pointer*. Zeiger

Prinzip Verzeigerung

Karte mit Verweisen



Kartendarstellung im Hauptspeicher



Beispiel Verzeigerung

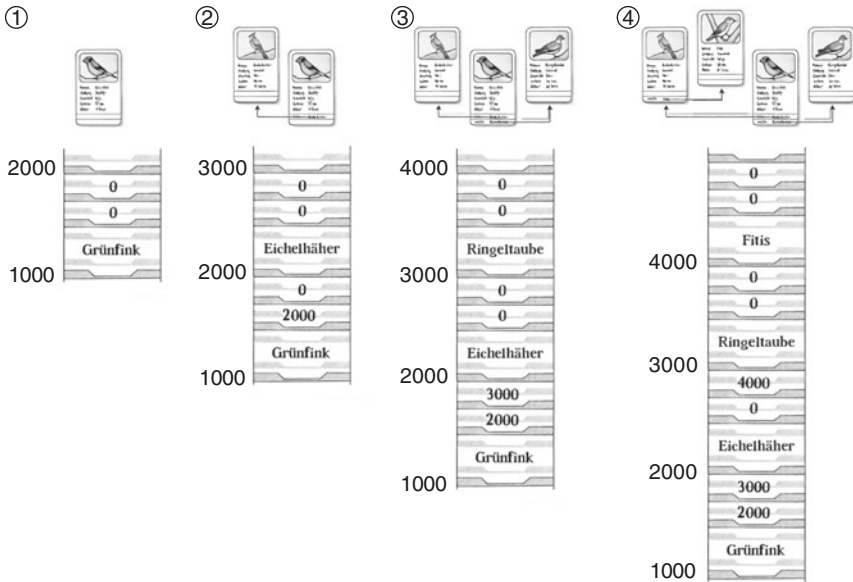


Abb. 5.18 Die Idee des Zeigers und der Verzweigung am Beispiel einer Baum-Datenstruktur. (© Susanne Staubli, Eckart Zitzler)

sind vom Prinzip her nichts anderes als Verweise, so wie Seitenzahlen und Abbildungsnummern in einem Buch. Dadurch müssen die Kartenelemente nicht hintereinander im Hauptspeicher abgelegt werden, sondern können an beliebiger Stelle stehen. Welch großer Vorteil das ist, zeigt sich dann, wenn Elemente aus einer Datenstruktur zu entfernen sind. Jetzt müssen keine Elemente mehr im Hauptspeicher verschoben werden, sondern es reicht, wenn wir die Zeiger umhängen. Wenn wir beispielsweise die Wurzel eines

Baums löschen wollen, dann lässt sich die Baumstruktur so anpassen, dass das am weitesten rechts stehende Element im linken Teilbaum herausgeschnitten und als neue Wurzel verwendet wird. Dazu müssen wir einzig drei Zeiger ändern: Beim Vorgänger ist der rechte Nachfolgerzeiger gleich 0 zu setzen und bei der neuen Wurzel werden die Nachfolgerzeiger durch diejenigen der alten Wurzel ersetzt. Fertig. Ach ja: Ist kein linker Teilbaum vorhanden, ist der rechte Nachfolger der Wurzel die neue Wurzel.

Das Zeigerkonzept ist generell einsetzbar, und mit ihm lassen sich beliebige Datenstrukturen erzeugen – oder anders gesagt: Beziehungsgeflechte abbilden. Abb. 5.19 zeigt Ihnen drei Beispiele. Eine Warteschlange können wir beispielsweise über eine Kette von über Zeiger verbundenen Elementen modellieren – und dabei kann sogar jemand entnervt die Schlange verlassen und die Datenstruktur lässt sich einfach anpassen. So eine Datenstruktur nennt sich dann Liste. Eine Liste ist wie ein spezieller Baum, bei dem jedes Element maximal einen Nachfolger hat. Die Bäume haben wir bereits diskutiert. Mit ihnen lassen sich Stammbäume darstellen und selbst Stadtpläne, wie in Bsp. 5.11 erläutert ist.

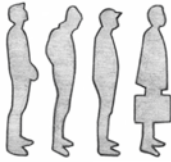
Beispiel 5.11: Wie man einen Stadtplan im Computer speichert

Auf einem Stadtplan sind sehr viele verschiedene Informationen enthalten, wir wollen uns hier nun einzig auf das Straßennetz konzentrieren und abspeichern, welche Straßen wie miteinander verbunden sind. Um das zu realisieren, wählen wir eine auf den ersten Blick komisch anmutende Perspektive. Wir konzentrieren uns auf die Kreuzungen und betrachten diese als die Elemente unserer Datenstruktur. Aus diesem Blickwinkel heraus drücken die Straßen Beziehungen zwischen Kreuzungen aus, sprich: Wir repräsentieren einen Fahrweg, der von einer Kreuzung zu einer anderen führt, als Zeiger. Ein Datenelement – eine Kreuzung – enthält also eine variable Anzahl von Zeigern – abgehenden Fahrwegen –, die es mit anderen Datenelementen – Kreuzungen – verbindet. Dabei werden Straßen, die in beiden Richtungen befahren werden können, auch doppelt verkettet.

Liste und Baum sind übrigens nur zwei Varianten desselben Grundschemas, bei dem Datenelemente beliebig miteinander verknüpft sind. Dieses allgemeine Schema wird als Graph bezeichnet, und Graphen begegnen uns im Alltag andauernd. Liniennetze für öffentliche Verkehrsmittel werden so grafisch(!) dargestellt, auch Wissensinhalte lassen sich auf diese Art und Weise strukturiert visualisieren (siehe Abb. 5.20); dabei werden die Datenelemente häufig als Kreise dargestellt und die Zeiger als Pfeile oder Linien. Mit Graphen lassen sich also beliebige Netzwerke darstellen.

Objekte und ihre Anordnung in der Wirklichkeit

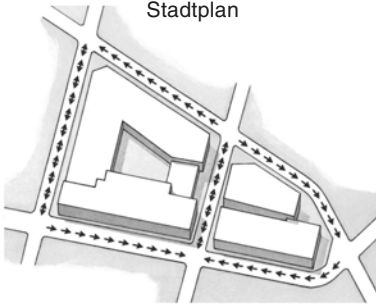
Menschenschlange



Familienstammbaum



Stadtplan

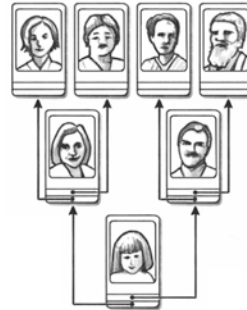


Darstellungsarten mittels Karten und Verweisen

Liste



Baum



Graph

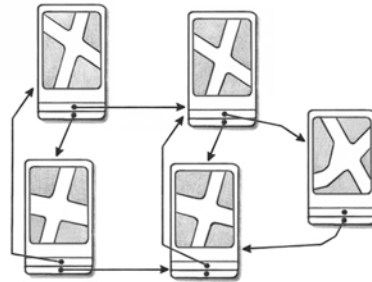


Abb. 5.19 Beispiele für Graphenstrukturen: Listen, Bäume und beliebige Netzwerkstrukturen. (© Susanne Staubli, Eckart Zitzler)

Zeiger bieten also die Möglichkeit, beliebige Datenstrukturen zu erzeugen und dynamisch anzupassen. Damit sind sie ein mächtiges Instrument zur Strukturierung von Daten, auch wenn sie in vielen Programmiersprachen so gar nicht mehr ersichtlich sind. Häufig wird ein umfangreicher Satz an dynamischen, d. h. leicht veränderbaren Datenstrukturen zur Verfügung gestellt, die natürlich auf dem Zeigerkonzept beruhen, doch um deren detaillierte Implementierung sich die Programmiererin nicht kümmern muss.

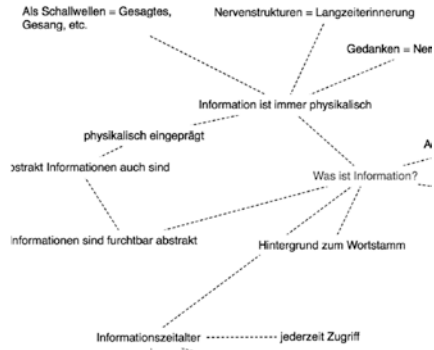


Abb. 5.20 Graphenbeispiele: ein Liniennetzplan für Bus und Straßenbahn (links) und eine Ideensammlung in Form eines semantischen Netzes (rechts). (© BERNMOBIL (links), Eckart Zitzler (rechts))

Welche Struktur ist die richtige?

Mit Stapeln, Feldern, Listen, Bäumen und allgemeinen Graphen haben Sie zentrale Datenstrukturen in der Informatik kennengelernt, aber natürlich gibt es noch viel mehr. Und genauso wie bei den Speichertechnologien können wir uns fragen, warum es denn so viele verschiedene braucht. Die Antwort ist letztlich auch die gleiche: Alle haben ihre Vor- und Nachteile, es hängt von der konkreten Anwendungssituation ab, welche Datenstruktur am besten geeignet ist. Geht es darum, die Daten möglichst schnell abzufragen, weil diese so schnell hineinkommen, oder ist vor allem wichtig, später etwas schnell darin zu finden? Oder brauchen wir eine flexible Datenstruktur, bei der sowohl das Hinzufügen als auch das Löschen schnell abläuft, und die gleichzeitig moderate Suchzeiten gewährleistet? Oder brauchen wir eine besonders platzsparende Datenstruktur? Zugriffszeit und Platzbedarf sind also die wesentlichen Faktoren, um die es hierbei geht.

Bei der Zugriffszeit ist die Frage, wie lange gewisse Operationen wie Einfügen, Löschen und Suchen dauern. Da es sich ja um kleine Programme handelt, die diese Operationen realisieren, werden diese Programme auch genauso wie normale Algorithmen bewertet: Wie viele elementare Prozessorbefehle werden benötigt in Abhängigkeit von der Anzahl der Elemente, die aktuell in der Datenstruktur gespeichert sind? Dabei interessiert weniger die konkrete Zahl, sondern es geht um die Größenordnung: Ist die Anzahl der Befehle proportional zur Größe der Datenstruktur oder wächst sie exponentiell? Und wie sieht das im besten Fall, im durchschnittlichen Fall und im schlechtesten Fall aus? Diese Themen werden wir ausführlicher in

Kap. 8 behandeln, trotzdem gebe ich Ihnen in Bsp. 5.12 schon einmal einen ersten Eindruck.

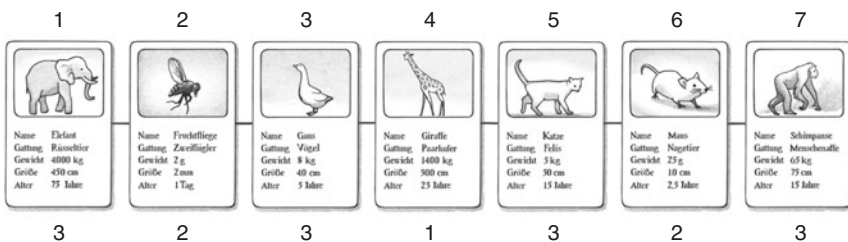
Beispiel 5.12: Datenstrukturen bewerten

Um zu illustrieren, wie Datenstrukturen verglichen werden können, lassen Sie uns das Suchen in einem unsortierten und einem sortierten Feld gegenüberstellen. Wie lange dauert es nun, um ein Element zu finden? Dabei soll uns hier nicht interessieren, welcher Prozessor zum Einsatz kommt und wie viel Zeit effektiv verstreicht, sondern wir nehmen eine sehr grobe Abschätzung vor. Wir betrachten einzig die Anzahl der Karten, die angeschaut werden müssen, bevor die gewünschte Karte gefunden wurde.

In Abb. 5.21 sehen Sie das bereits bekannte Beispiel mit den sieben Karten. Für jede einzelne Karte ermitteln wir nun, wie viele Karten wir anschauen müssen, wenn wir nach dem entsprechenden Namen suchen – einmal für eine lineare Suche, bei der von vorne nach hinten nach dem gewünschten Datensatz gesucht wird, und einmal für eine binäre Suche. Die entsprechenden Zahlen sind an den Karten vermerkt, oben für die lineare Suche und unten für die binäre Suche. Wie Sie sehen können, müssen bei der binären Suche wesentlich weniger Karten untersucht werden als bei der linearen Suche: im Schnitt nur zweieinhalb Karten gegenüber vier Karten.

Beim Platzbedarf geht es im Wesentlichen um die Frage, wie viel zusätzlicher Speicherplatz benötigt wird, um das Gerüst der Datenstruktur zu speichern. Nutzen wir z. B. Zeiger, so erfordert das pro Datenelement ja auch Speicherplatz, um die Adressen der verbundenen Datenelemente zu speichern. Verwenden wir hingegen ein zweidimensionales Feld, z. B. um eine Schachspielsituation zu speichern, dann müssen wir von vorneherein den entsprechenden Speicherplatz reservieren, auch wenn ein Großteil vielleicht nicht benutzt wird. Repräsentiert im Schach-Beispiel jedes Feldelement eine

lineare Suche: 28 Kartenzugriffe



binäre Suche: 17 Kartenzugriffe

Abb. 5.21 Vergleich zweier Datenstrukturen und der Zugriffszeiten. (© Susanne Staubli, Eckart Zitzler)

Position auf dem Spielfeld, dann ist mindestens die Hälfte der Elemente unbenutzt, denn es gibt nur 32 Spielfiguren.

Es ist immer ein Abwägen zwischen Nutzen und Aufwand, sowohl auf die Zeit als auch auf den benötigten Platz bezogen. An sich muss jede Datenstruktur auf die jeweilige Anwendungssituation zugeschnitten werden, häufig kombiniert man auch verschiedene Datenstrukturen, um einen ausgewogenen Kompromiss zu erzielen (siehe Bsp. 5.13).

Beispiel 5.13: Datenstrukturen kombinieren

In Abb. 5.22 sehen Sie eine kombinierte Datenstruktur, die ein zweidimensionales Feld mit mehreren Listen verknüpft. Sie imitiert einen Karteikasten zu Zeiten des Bücherwurms, bei dem die Karten gemäß den ersten zwei Anfangsbuchstaben in Gruppen abgelegt sind. Ziel dabei ist, Karten schnell über den Namen finden zu können.

Das Feld enthält für jedes mögliche Buchstabenpaar ein Feldelement (die Umrechnung von Buchstaben in Reihen und Spalten ist einfach, das A bekommt eine 0, das Z die 25 zugewiesen). Jedes Feldelement wiederum enthält einen Zeiger auf eine Liste. In dieser Liste sind in beliebiger Reihenfolge alle Karten abgelegt, deren Name mit dem entsprechenden Buchstabenpaar beginnt. Um eine Karte über ihren Namen zu suchen, rechnen wir zuerst die zwei Anfangsbuchstaben in eine Feldposition um, holen uns aus dem entsprechenden Feldelement die Startadresse der Liste und gehen anschließend die Liste von vorne nach hinten durch, um die gewünschte Karte zu finden.

Diese hybride Datenstruktur vergeudet auf der einen Seite etwas Platz, denn nicht alle Buchstabenkombinationen kommen als Wortanfänge für die Namen vor (z. B. Zz). Doch auf der anderen Seite ist der Zugriff sehr schnell: Wenn nur wenige Elemente enthalten und die Listen kurz sind, ist man wesentlich schneller als mit einer reinen Liste. Und verwendet man Bäume anstatt Listen, wird man noch schneller – schneller, als wenn wir nur einen Baum verwenden würden. Der hier vorgestellte Ansatz ist ein Beispiel für eine Technik, die sich Hashing nennt. Dabei wird ein Datensatz über eine Hash-Funktion eindeutig in eine Zahl umgerechnet (die ersten zwei Anfangsbuchstaben des Namens lassen sich auch als eine Dualzahl darstellen). Über diese Zahl wird dann auf ein eindimensionales Feld zugegriffen, von dem auf eine weitere Datenstruktur verwiesen wird, in der die effektiven Datensätze abgelegt sind.

In der Realität können Datenstrukturen äußerst komplexe Gebilde und ihr Entwurf eine Kunst sein. Oft kommen mehrere Datenstrukturen gleichzeitig zum Einsatz – für denselben Datenpool; so lassen sich die Zugriffszeiten minimieren, auch wenn verschiedenste Operationen auf dem Datenbestand erfolgen. Das ist vor allem der Fall bei Datenbanken: Das sind spezielle Informatiksysteme, um große Datenmengen konsistent zu speichern und zugreifbar zu machen. Datenstrukturen sind zudem eng mit Algorithmen

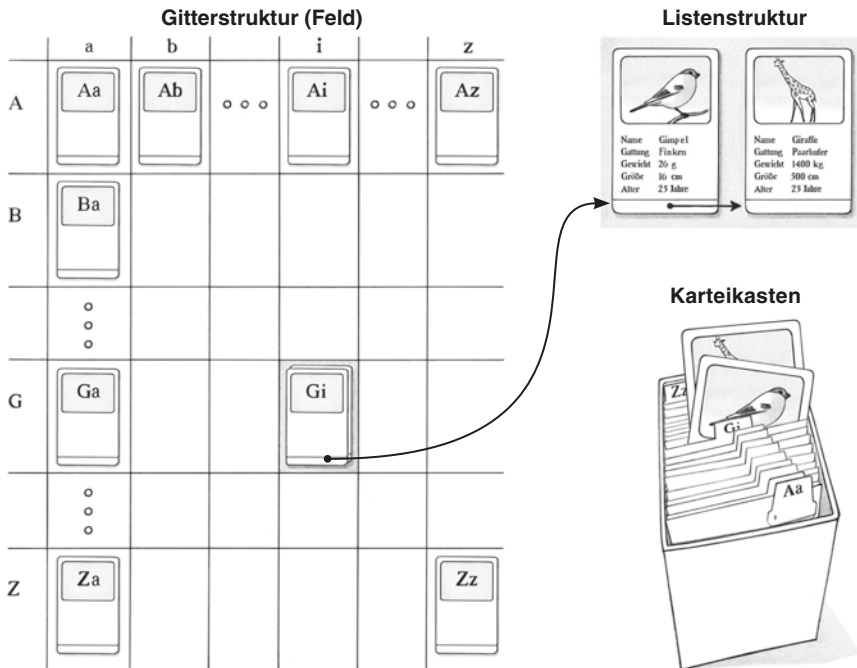


Abb. 5.22 Beispiel einer hybriden Datenstruktur bestehend aus Feld und Liste.
(© Susanne Staubli, Eckart Zitzler)

verknüpft, denn jedes Programm muss ja mit Daten jonglieren und diese Daten geeignet zwischenspeichern. Deswegen werden beide Begriffe häufig im gleichen Atemzug genannt: Algorithmen und Datenstrukturen.

Das Leben und die Einprägungen

Es ist schon erstaunlich, welche ausgeklügelten Systeme wir entwickelt haben, um Informationen festhalten und abrufen zu können. Dabei haben die Menge an Informationen und die Geschwindigkeit, mit der wir auf sie zugreifen können, enorm zugenommen. Die Computertechnologie ist quasi unsere neueste Errungenschaft. Doch ihr ist viel vorausgegangen: die Sprache, die Schrift, der Buchdruck und unzählige weitere Entdeckungen, Erkenntnisse und Erfindungen. Es scheint damit zusammenzuhängen, dass wir ungemein neugierige und vor allem auch soziale Tiere sind: Wir wollen die Welt um uns verstehen, wir wollen uns verstehen und wir wollen dieses Wissen mitteilen und weitergeben.

Ja, und wir bauen nicht nur informationsverarbeitende Systeme, wir sind selbst welche und stellen riesige, wandelnde Informationsspeicher dar. Man könnte fast meinen, im Leben drehe sich alles um eines, nämlich darum, Informationen zu sammeln. Nicht von ungefähr hat ein bekannter Gedächtnisforscher eines seiner Bücher so betitelt: „Wir sind Erinnerung“. Wenden wir also unseren Blick am Ende dieses Kapitels von den künstlichen zu den natürlichen Speichern und schauen uns an, wie Informationen uns prägen. Dieser Perspektivenwechsel ist auch für die Informatik interessant und inspirierend.

Informationen einfangen

Wir Menschen beinhalten verschiedenste Speichertechnologien, die in einem vielschichtigen Wechselspiel zueinander stehen. Alles begann mit der Entstehung eines Moleküls vor über 3,5 Mrd. Jahren, der DNA (vgl. Kap. 1). Man nimmt an, dass sich verschiedenste Moleküle in der Ursuppe befanden, doch nur die DNA war aufgrund ihrer besonderen Beschaffenheit in der Lage, Informationen als Folge von A-C-G-T-Ketten langfristig zu speichern und – vor allem – sich zu replizieren, also fortzupflanzen. Die DNA ist ein äußerst robustes Molekül, nahezu unverwundlich, was man schon daran erkennt, dass wir die DNA von Dinosauriern in Fossilien extrahieren können. Ähnliche Eigenschaften wie die DNA weist das RNA-Molekül auf, es ist jedoch insofern einfacher, als dass es nicht die Doppelhelix-Struktur und somit nicht die Eigenschaft der Replizierbarkeit aufweist. RNA ist unter gewissen Bedingungen weniger robust, dafür aber reaktiver, also einfacher beeinflussbar durch Enzyme. Eine Hypothese lautet, dass sich DNA aus RNA entwickelt hat. Wie wir in Kap. 1 gesehen haben, kann DNA die Baupläne von Proteinen speichern, und somit ergab sich ein faszinierendes Zusammenspiel von DNA, RNA und Proteinen. Dies war die Grundvoraussetzung für die Bildung von ersten Zellen. DNA bildete darin den zentralen Langzeitspeicher, während RNA als Hilfsppeicher fungierte. Und auch Proteine speichern Informationen, denn sie können sich an der DNA festkrallen und somit die Abschrift steuern; ihre Präsenz ist Information. Allerdings sind Proteine anfällig, sie können z. B. bei Hitze zerfallen und ihre Form verändern. Proteine sind demnach Kurzzeitspeicher, dabei aber sehr flexibel, vor allem schnell veränderlich.

Den nächsten Entwicklungsschritt stellen mehrzellige Organismen dar. Sie bildeten sich, als sich bei Zellteilungen die Tochterzellen nicht komplett abspalteten, sondern wie siamesische Zwillinge verhängt waren – ausgelöst

durch eine Mutation im Erbgut. Die ersten Tiere, die so vor vielen hundert Millionen Jahren entstanden, waren Schwämme und Quallen (Abb. 5.23). Mit der Zeit entstanden immer größere Zellverbände und schließlich begannen sich Gruppen von Zellen auf bestimmte Aufgaben zu spezialisieren: auf die Fortbewegung, auf das Fressen, auf die Wahrnehmung usw. Da aber alle Zellen die gleiche DNA-Sequenz in sich tragen, brauchte es eine andere Art von Gedächtnis, um den Zelltyp festzuhalten. Zu diesem Zweck wurde der DNA-Langzeitspeicher verfeinert. Die Information des Zelltyps wird gespeichert über Modifikationen am DNA-Molekül oder an seiner Verpackung – quasi Markierungen, die die eigentliche A-C-G-T-Sequenz nicht verändern. Der Wissenschaftszweig, der sich mit diesem Gedächtnis beschäftigt, heißt Epigenetik (siehe Anm. 5.5).

Anmerkung 5.5: Epigenetik oder der Speicher oberhalb des Speichers

Das Gebiet der Epigenetik wurde 1974 begründet und beschäftigt sich mit vererbten Veränderungen der Genfunktion, die sich nicht durch Veränderungen der DNA-Sequenz erklären lassen. Wir hatten ja gesehen, dass Schalterproteine die Genaktivität regulieren können. Es gibt jedoch noch andere Faktoren, die an der Verpackung der DNA festgemacht sind und die ganze Gengruppen ab- oder abschalten können. „Epi“ steht für „darüber“, was heißt, dass es um Phänomene „oberhalb“ der DNA-Sequenz geht. Es sind also Veränderungen an den Chromosomen – verpackte DNA-Moleküle –, die Abschnitte oder ganze Chromosomen betreffen und deren Aktivität steuern können. So lässt sich z. B. erklären, warum alle unsere Körperzellen die gleiche DNA in sich tragen und doch in verschiedenen Zelltypen unterschiedliche Gene aktiv sind. Es wurde bereits gezeigt, dass solche epigenetischen Markierungen auch durch die Umwelt beeinflussbar sind.

Die spezialisierten Zellverbände führten zu komplexeren Organismen mit einem komplexeren Verhalten im Vergleich zu den Einzellern. Doch die

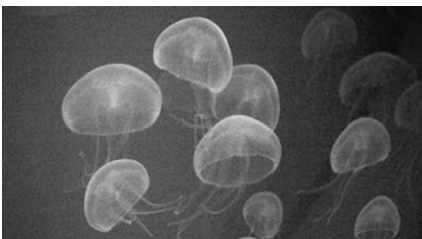


Abb. 5.23 Die ersten Mehrzeller neben den Schwämmen: Quallen. (© Judith Büsser)

Zellverbände müssen koordiniert werden, es braucht ein Zusammenspiel. Und um dieses Zusammenspiel zu verfeinern, hat sich das Nervensystem gebildet: Zellen, die darauf spezialisiert sind, Signale von einem Ort zum einem anderen schnell zu übertragen und gleichzeitig eine Verarbeitung, also die Koordination vorzunehmen. Einfache Tierstämme wie Seeanemonen und Quallen waren wohl die ersten Organismen mit Nervensystemen; viele weitere wirbellose Organismen haben vergleichsweise einfache Nervensysteme mit ein paar Tausend Nervenzellen, deren „Verdrahtung“ häufig fix, d. h. genetisch vorgegeben ist. Diese Nervensysteme stellen so etwas wie einen Speicher für Verhalten dar, quasi eine fest verdrahtete Schaltung. Wirbeltiere hingegen verfügen nicht nur über ein wesentlich komplexeres Verhaltensrepertoire, sondern besitzen auch die Fähigkeit des Lernens, also eine höhere Stufe der Anpassungsfähigkeit. Das lässt sich auch an ihren Gehirnen erkennen, in denen selbst kleinste Regionen mehrere Tausend bis mehrere Millionen Nervenzellen umfassen. Mit dem Gehirn kam ein Erfahrungsspeicher hinzu, der im Laufe des Lebens flexibel modifiziert werden kann. (Daneben gibt es übrigens noch andere Erfahrungsspeicher, z. B. das erworbene Immunsystem).

Wenn wir uns nun den Menschen anschauen, dann offenbart sich uns auch hier ein Spektrum an Speichertechnologien, die hierarchisch zueinander angeordnet sind und miteinander agieren (siehe Abb. 5.24). Jede Speichertechnologie ist auf einen bestimmten Zweck ausgerichtet und darauf optimiert. Und auch hier geht es um Faktoren wie Dauerhaftigkeit, Robustheit, Reproduzierbarkeit, Geschwindigkeit, Kapazität usw. Darüber hinaus haben wir uns externe Hilfsspeicher geschaffen, von der Tontafel



Abb. 5.24 „Speichertechnologien“ im Menschen. (© Susanne Staubli, Eckart Zitzler)

über das Buch bis zum Computer, die die Speicherkapazität unseres Gehirns erweitern. Spannend ist in diesem Zusammenhang, dass wir wieder zu den Anfängen zurückkehren: Es gibt Bestrebungen, computerlesbare DNA-Speicher zu bauen (Anm. 5.6). Lassen wir uns überraschen, welche Technologien noch zum Einsatz kommen werden.

Anmerkung 5.6: DNA als Langzeitspeicher

Dass DNA-Moleküle als Datenträger für Zeichenketten genutzt werden können, haben wir bereits in Kap. 1 gesehen. Sie bieten den Vorteil, dass sie große Mengen an Informationen auf kleinstem Raum speichern können, mehr als 1 Zetabyte (1 Mrd. Terabyte) in einem Gramm. DNA-Speicher weist eine enorm hohe Dichte auf, die um Größenordnungen höher ist als bei herkömmlichen Massenspeichern. Wenn er zudem richtig verpackt wird, wie z. B. in fossilen Knochen, dann kann DNA-Speicher mehrere Hunderttausend Jahre überdauern. Aus diesem Grund sind Forscher daran interessiert, die richtige Verpackung und – das ist wichtig – auch Verfahren zu finden, um die Daten korrekt wieder auszulesen. Beim Auslesen können sich verschiedene Fehler einschleichen, die chemisch begründet sind; durch fehlerkorrigierende Codes lässt sich das kompensieren. Natürlich dient DNA-Speicher zur Archivierung und nicht zum Massenspeicher im Dauerbetrieb.

Damit kommen wir nun von der natürlichen Speicherhierarchie zu den natürlichen Datenstrukturen. Wir tasten uns an die Erinnerungen heran.

Informationen nutzen

Die Fähigkeit, Erfahrungen – eigene oder von anderen übermittelte – festhalten und abrufen zu können, ist eine Grundvoraussetzung, damit wir Menschen uns in dieser komplexen Welt zurechtfinden können. Ohne Erinnerungen gibt es kein Lernen, kein Bewusstsein, keine Persönlichkeit. Doch wie werden Erinnerungen im Gehirn organisiert und gespeichert, und zwar im Detail? Und welcher Datenstrukturierungen bedient sich dieses Hochleistungsorgan? Das sind Fragen, mit denen sich viele Forscherinnen und Forscher beschäftigen – was wiederum darauf hindeutet, dass es noch viel zu entdecken gibt.

Klar ist, das Gehirn arbeitet ganz anders als die Speichergefäße, die wir beim Computer kennengelernt haben. Es gibt keine Adressen, also speziell angeschriebene Erinnerungsschubladen. Der Speicher in unserem Kopf funktioniert assoziativ. Wir nehmen die Welt um uns und in uns wahr, erfassen die Beschaffenheit von Dingen – wie sie aussehen, riechen, tönen

– und unser Gehirn bewahrt diese Eindrücke für die spätere Verwendung auf. Dabei koppelt es die einzelnen Sinneswahrnehmungen miteinander zu einem Gesamteindruck. Bei Ereignissen ist das ähnlich. Ereignisse, die zeitlich zusammen auftreten, werden miteinander verknüpft – jedoch nur dann, wenn diese Quasi-Gleichzeitigkeit regelmäßig beobachtet wird. Das Phänomen der Konditionierung ist ein Paradebeispiel hierfür (Bsp. 5.14). Schließlich kann ein Aspekt der Erfahrung genügen, vielleicht ein spezieller Geruch oder eine markante Form, um die Verknüpfungen zu aktivieren und das gespeicherte Gesamtbild abzurufen. Der Zugriff auf Erinnerungen erfolgt also über einzelne Wahrnehmungsmerkmale. Wenn wir uns noch einmal die Datenstrukturen in der Informatik vor Augen führen, dann klingt das hier nach der „Zauber-Datenstruktur“: Ein beliebiger Eintrag reicht, um die korrespondierende Karte mit all den restlichen Einträgen aufzufinden.

Beispiel 5.14: Konditionierung

Der Russe Iwan Petrowitsch Pawlow entdeckte in seinen berühmten Experimenten zu Beginn des 19. Jahrhunderts, dass Hunde eine angeborene Reaktion mit einem vormals neutralen Reiz assoziieren können. Er trainierte die Hunde, indem er, bevor das Futter gereicht wurde, eine Glocke ertönen ließ. Durch stetige Wiederholung koppelte sich die Erwartung des Futters schließlich an das Glockensignal. Auch wenn sie kein Futter bekamen: Es genügte das Ertönen der Glocke, dass den Hunden der Speichel im Maul zusammenlief. Das nennt man klassische Konditionierung (Abb. 5.25), und diese kennen wir selbst ja auch. Ich beispielsweise genieße den morgendlichen Kaffee in entspannter Atmosphäre; entsprechend ist der Reiz „Kaffee“ mit Pause und Entspannung gekoppelt. Diese Assoziation versuche ich gelegentlich abzurufen, z. B. wenn eine unangenehme Arbeit vor mir liegt – klappt nur leider nicht immer.

Des Weiteren ist das Gehirn nicht darauf ausgelegt, alles, was wir jemals erleben, eins zu eins wie in einem Film festzuhalten. Erinnerungen sind vielmehr sehr individuelle Aufnahmen, sie sind vorgeprägt durch unsere Geschichte, unsere Befindlichkeit, unsere Bedürfnisse und sie sind immer auf das Leben ausgerichtet, also geleitet von der Frage: Ist das wichtig für mein Leben? Erlebnisse werden demnach nicht originalgetreu, sondern eher in Form von Skizzen abgespeichert. Der Neurowissenschaftler António Damásio spricht in diesem Zusammenhang von Dispositionen. Es handelt sich dabei um Regeln, die Extrakte aus einer Menge von Erfahrungen darstellen und unbewusst gespeichert werden. Man könnte sie als Gerüste für Erinnerungen bezeichnen, denn beim Erinnern werden die ursprünglichen



Abb. 5.25 Die klassische Konditionierung. (© Reddogs/stock.adobe.com (links), Judith Büsser (rechts))



Abb. 5.26 Links ist dort, wo ein L entsteht – wie wir Informationen in andere Informationen einbetten, um uns besser an sie erinnern zu können. (© Judith Büsser)

verknüpften Wahrnehmungen, die inneren Bilder, aus diesen Skizzen rekonstruiert. Es ist eine Annäherung an die damalige Wirklichkeit, deswegen ist alles nicht mehr so lebendig und so detailliert wie beim Original. Das Gehirn trennt dabei Relevantes von Belanglosem, immer im Hinblick darauf, unser Überleben in unterschiedlichsten Situationen sicherzustellen.

Das ist vermutlich auch der Grund, warum uns unser Gedächtnis gelegentlich einen Streich spielt und wir Schwierigkeiten haben, zusammenhangslose Fakten zu memorieren. Doch wir wissen uns zu helfen: Wer kennt sie nicht, die Eselsbrücken? Indem wir Informationen in einen Merkspruch oder ein Bild einbetten, sie in etwas anderem verankern, können wir sie leichter wieder aufrufen (siehe Abb. 5.26). Wir nutzen also die assoziative Arbeitsweise des Gedächtnisses. Und übrigens gibt es auch beim Computer Speicherkomponenten, die auf inhaltlichen Verknüpfungen beruhen und assoziativ arbeiten (siehe Bsp. 5.15).

Beispiel 5.15: Assoziativer Speicher

Der klassische Computerspeicher, wie Sie ihn kennengelernt haben, zeichnet sich dadurch aus, dass auf Daten über das Wo zugegriffen wird. Im Hauptspeicher beispielsweise hat jedes Speicherelement eine Adresse, über diese Adresse wird auf den Inhalt des Speicherelements zugegriffen. Doch was wäre, wenn es keine Adressen gäbe und wir stattdessen über einen Teil des Inhalts auf den restlichen Inhalt zugreifen könnten? Wenn wir beispielsweise die unteren 8 Bit einer 16-Bit-Zahl angäben und dann die vollständige 16-Bit-Zahl zurückbekämen? Das wäre nämlich so ähnlich wie bei den Assoziationen: Aus einer Teilwahrnehmung wird die Gesamtwahrnehmung rekonstruiert.

Tatsächlich gibt es so eine Art Speicher. Er wird als assoziativer Speicher bezeichnet und häufig in Prozessor-Caches eingesetzt. In diesem Fall werden die Speicherelemente des Cache über einen Teil des gespeicherten Inhalts angesprochen. Der Inhalt eines Cache-Speicherelements ist ein Datenblock aus dem Hauptspeicher zusammen mit der Speicheradresse des untersten Datenblockelements. Durch die Angabe einer Hauptspeicheradresse kann der entsprechende Datenblock im Cache identifiziert werden – falls er dort enthalten ist.

Ist das Gehirn dann letztlich nicht eine riesige Datenbank, die ausgewählte Inhalte unserer Erlebnisse aufbewahrt und miteinander verknüpft? Und spiegelt das gespeicherte Beziehungsgeflecht nicht genau die Idee der allgemeinen Datenstruktur wider, wie wir sie mit dem Graphen kennengelernt haben? Trotz offensichtlicher Parallelen ist Vorsicht angebracht, denn diese allgemeinen Aussagen erklären nicht, wie der neuronale Speicher im Detail aussieht und funktioniert. In der Tat geben uns die Datenstrukturen des Gehirns viele Rätsel auf, und eigentlich ist der Begriff der Datenstruktur so nicht zutreffend; wenn schon, dann müssten wir von Daten- und Verarbeitungsstrukturen sprechen. Im Gehirn gibt es keine Trennung der Speicher- und Verarbeitungseinheiten, da stehen nicht Register dem Rechenwerk oder der Prozessor dem Hauptspeicher gegenüber. Die Nervenzellen realisieren beides, Schaltstellen und Verbindungen. So gesehen lässt die gewohnte Unterscheidung zwischen Software und Hardware, die die flexiblen von den unveränderlichen Bestandteilen eines Computersystems separiert, beim Gehirn so nicht anwenden. Das macht es auch so faszinierend für diejenigen, die sich mit Informationsverarbeitung auseinandersetzen. Sicher wird die Hirnforschung noch so manche Überraschung und Anregung für die Informatik bereithalten. Ob sich daraus völlig neue Speicherkonzepte ergeben – man hat bereits Hardware-Bausteine entwickelt, die das Verhalten von Nervenzellen nachbilden –, bleibt abzuwarten. Es wird auf jeden Fall spannend bleiben.

Fazit

Speichern und Erinnern stehen am Anfang des Lebens und bilden das Fundament für die Informationsverarbeitung. Obwohl das Thema auf den ersten Blick nur eine sekundäre Rolle zu spielen scheint, führt es zu einer Kernfrage der Informatik: Wie lassen sich Daten geschickt ablegen, sodass später schnell auf die Daten zugegriffen und gleichzeitig viele Daten gespeichert werden können? Der Wettstreit zwischen Zugriffsgeschwindigkeit, Kapazität und Haltbarkeit steht immer im Mittelpunkt und verlangt nach Lösungen. Je leistungsfähiger Computersysteme werden, desto aufwändiger werden die Lösungen, um mit diesem Dilemma möglichst gut umzugehen – das kann man auch hinsichtlich der Lebewesen sagen. Auf der einen Seite werden Hierarchien von Speichermethoden gebaut, um das Letzte an Geschwindigkeit und Kapazität herauszuholen. Die Komplexität dieser Systeme wird immer größer, immer mehr Ebenen werden in die Speicherhierarchie eingezogen. Auf der anderen Seite nutzen Programme Datenstrukturen, um möglichst effizient mit Zwischenergebnissen umzugehen. Dabei gibt es kein Patentrezept, sondern es hängt ganz davon ab, welche Daten man verarbeiten muss, um welche Datenmengen es sich handelt und wie die Daten genutzt werden sollen. Speicherverfahren sind häufig das Zünglein an der Waage, wenn es um die Gesamtperformanz eines Computersystems geht.

Zum Nach- und Weiterlesen

- Berger, R. (2008). *Warum der Mensch spricht: Eine Naturgeschichte der Sprache*. Frankfurt a. M.: Eichborn.
- Damasio, A. (2011). *Selbst ist der Mensch: Körper, Geist und die Entstehung des menschlichen Bewusstseins*. München: Siedler.
- Haarmann, H. (2011). *Geschichte der Schrift* (4. Aufl.). München: Beck.
- Ottmann, T., & Widmayer, P. (2012). *Algorithmen und Datenstrukturen* (5. Aufl.). Heidelberg: Spektrum Akademischer.
- Patterson, D., & Hennessy, J. L. (2011). *Rechnerorganisation und Rechnerentwurf: Die Hardware/Software-Schnittstelle* (4. Aufl.). München: Oldenbourg.
- Sadava, D., Orians, G., Heller, H. C., Hillis, D., & Berenbaum, M. R. (2011). *Purves, Biologie* (9. Aufl.). Heidelberg: Spektrum Akademischer.
- Schacter, D. L. (1999). *Wir sind Erinnerung: Gedächtnis und Persönlichkeit*. Reinbek bei Hamburg: Rowohlt.
- Tanenbaum, A. S., & Bos, H. (2016). *Moderne Betriebssysteme* (4. Aufl.). München: Pearson Studium.

Bildquellen

Abb. 5.13: cartarium, Bildnummer 123382789, <https://stock.adobe.com> (links, abgerufen am 30.05.2017); Deyan Georgiev, Bildnummer 57312608, <https://stock.adobe.com> (rechts, abgerufen am 30.05.2017)

Abb. 5.20: BERNMOBIL, <https://www.bernmobil.ch/DE/Fahrplan-Netz/Fahrplaene-Netzplan/Netz-Perronplan/> (links, abgerufen am 26.05.2017)

Abb. 5.25: Reddogs, Bildnummer 91300679, <https://stock.adobe.com> (abgerufen am 30.05.2017)

6

Netze spinnen

Überblick

Netze – jene geknüpften Gebilde, deren Konstruktion Spinnen so meisterhaft beherrschen und die in der Informatik einen weiteren Meilenstein markieren – sind das Thema dieses Kapitels. Wir werden uns anschauen, wie einzelne Computer zu einem Netzwerk zusammengeschlossen werden können, welche Technologien dabei zum Einsatz kommen und wie Nachrichten in einem sich ständig verändernden Verbindungsgeflecht ihren Weg finden. Schritt für Schritt hangeln wir uns vom einfachen Fall, in dem zwei Computer über eine Leitung verbunden sind, zum weltumspannenden Kommunikationsverbund, der Milliarden von Rechnern umfasst. Darüber hinaus werden wir sehen, wie sich Computernetzwerke gemauert haben, vom reinen Kommunikationsmedium zum Gesamtsystem mit neuartigen Eigenschaften. Indem Aufgaben hierarchisch oder gleichberechtigt aufgeteilt und verteilt bearbeitet werden, wird aus einem Verbund individueller Geräte ein Weltcomputer und das Internet zu einem Nervensystem unserer Erde.

Wir Menschen können nicht nicht kommunizieren. Der Austausch von Informationen ist essenziell, wenn eine Gemeinschaft funktionieren soll, und das gilt nicht nur für die menschliche Gesellschaft, sondern auch für einen mehrzelligen Organismus oder einen Ameisenstaat. Wir Menschen haben sogar diverse Hilfsmittel erfunden, die uns die Kommunikation über weite Entfernungen ermöglichen. Bereits vor dem Internet konnten Nachrichten im großen Stil von Einzelnen zu Einzelnen – Briefpost und Telefon – und von Einzelnen zu vielen – Radio und Fernsehen – übermittelt werden. Was also lag näher, als auch Computer miteinander zu verbinden, über kleine und große Distanzen? So wurde es auf einmal möglich, dass viele mit vielen kommunizieren können. Es war allerdings ein langer Weg

von den ersten Ideen Ende der 1950er-Jahre bis zum weltumspannenden Internet, welches sich in den 1990ern etablierte und seitdem rasant wächst. Die Kommunikation ist dabei nur ein Aspekt. Durch die Vernetzung von Computersystemen lassen sich Ressourcen teilen und Aufgaben koordinieren. Das Internet hat sich folglich immer mehr gewandelt von einem Kommunikationsnetzwerk, das einzelne Rechner miteinander verbindet, zu einem riesigen eingebetteten System, das sich über den gesamten Globus erstreckt und Milliarden von Geräten umfasst.

Wir werden uns nun aufmachen, die Untiefen des Internets auszuloten und zu erkunden, was es im Innersten zusammenhält. Zunächst begeben wir uns in die Niederungen der Hardware und knüpfen an die Diskussion in Kap. 4 an: Wie können zwei oder mehr Computer über dasselbe Kommunikationsmedium, eine Leitung oder einen Funkkanal, miteinander verbunden werden und Daten austauschen? Anschließend werden wir sehen, wie sich Computernetzwerke verschiedenster Art über ein Zwischennetz – das Internet – zu einem globalen Netzwerk verbinden lassen und welche Möglichkeiten sich dabei eröffnen. Natürlich hat die Natur auch hier wieder viele Anregungen und Vorbilder zu bieten. Zum Schluss schauen wir uns daher zwei Beispiele aus der Biologie an, wie dort Kommunikationsnetze realisiert und genutzt werden.

Zusammen ist man weniger allein

Ein Computernetzwerk ist ein Verbund von Computern, die mit Leitungen oder per Funk verknüpft sind und so miteinander kommunizieren können. Im Gegensatz zu dem internen Kommunikationsnetzwerk eines Computersystems gilt es dabei größere Distanzen zu überwinden. Ganz neu ist das natürlich nicht. Der Mensch hat vorher schon verschiedene Methoden entwickelt, um Informationen über weite Entfernungen zu übertragen. Dabei waren ähnliche Fragen zu lösen wie diejenigen, mit denen wir uns im Folgenden beschäftigen werden. Wir haben also schon Erfahrung mit diesem Thema und es ist hilfreich, uns die bekannten Ansätze einmal vor Augen zu führen, bevor wir uns ans Netz-Werk machen.

Um die Distanz zu einem anderen Organismus zu überbrücken, stehen uns verschiedene natürliche Kommunikationsmittel wie Geruch, Lautäußerungen, Berührungen sowie Mimik und Gestik zur Verfügung (Abb. 6.1). In unmittelbarer Nähe können äußerst umfassende Informationen übermittelt werden, denken wir nur an unseren Gesichtsausdruck mit all seinen Nuancen. Vergrößert sich die Distanz, reduziert sich die



Abb. 6.1 Unsere natürlichen Kommunikationsmittel. (© Frithjof Nolting (links), Judith Büsser (Mitte und rechts))

Informationsmenge, die in gleicher Zeit übertragen werden kann. Details gehen verloren und wir müssen sicherstellen, dass die Signale unterscheidbar sind und beim Gegenüber ankommen. Wollen wir die normale Sprechdistanz überwinden, so können wir schreien und zusätzlich noch mit unseren Händen einen Sprechtrichter formen. Soll es noch weiter gehen, können wir statt der üblichen Phoneme, der Lautbestandteile unserer Sprache, Pfeiflaute in verschiedensten Variationen verwenden; auf den kanarischen Inseln beispielsweise haben sich eigenständige Pfeifsprachen herausgebildet. Oder wir können uns optisch verständigen, aus kleinen Gesten werden dann ausladende Handzeichen.

Wollen wir noch größere Distanzen überwinden, so benötigen wir Hilfsmittel (Abb. 6.2). Mit Nachrichtentrommeln – hölzernen Hohlgefäßen, die an beiden Enden mit Tierfellen oder -häuten überzogen sind – lassen sich Signale über mehrere Kilometer hinweg austauschen; mit ihnen haben sich u. a. in Afrika Einheimische über weite Strecken verständigt. An der Chinesischen Mauer hingegen kamen Rauchzeichen zum Einsatz, und es gibt noch eine Vielzahl weiterer optischer Hilfsmittel. Mit Fahnen lassen sich beispielsweise Buchstaben codieren, und das Winkeralphabet, bei dem eine Person zwei Flaggen in der Hand hält und in eine vordefinierte Stellung bringt, wird selbst heute noch in der Schifffahrt eingesetzt. Diese Idee wurde durch künstliche Gestengeber noch weitergeführt: Beim optischen Telegrafen werden auf Türmen miteinander verbundene, schwenkbare Holzbalken installiert, die verschieden zueinander angeordnet werden können. Indem mehrere Türme hintereinander platziert und Zwischenstationen eingebaut wurden, ließen sich bereits Ende des 18. Jahrhunderts auf diese Weise über 200 km überbrücken.

Es gibt also einige Überlegungen, die wir bei der Konstruktion eines Computernetzwerks anstellen müssen. Einerseits braucht es eine technologische Grundlage, ein Kommunikationsmedium, über das Informationen



Abb. 6.2 Traditionelle Kommunikationshilfsmittel, um große Distanzen zu überbrücken. (© Judith Büsser (links und Mitte), Ricochet64/stock.adobe.com (rechts))

übermittelt werden können. Andererseits ist festzulegen, wie Informationen dargestellt werden, z. B. buchstabenweise wie beim Winkeralphabet oder in Form einer eigenen Sprache wie bei den Pfeif- und Trommelsprachen, bei denen ganze Aussagen codiert werden. Zudem braucht es noch Möglichkeiten, Kommunikationskanäle zu koppeln und in einer Struktur anzuordnen; so lassen sich noch größere Distanzen überwinden und neue Teilnehmer einbinden.

Hallo Du: eine Leitung für zwei

Beginnen wir mit dem einfachsten Fall: Zwei Computer sollen miteinander reden können (Abb. 6.3). Am naheliegendsten ist die Verwendung elektrischer Leitungen wie beim internen Bus, bei dem über Spannungsänderungen Zeichen übertragen werden. Anstatt elektrischer Verbindungen sind auch Lichtwellenleiter möglich, also extrem dünne Fasern, die Licht über weite Strecken bündeln und übertragen können. Schließlich gibt es noch die Möglichkeit der Funkübertragung. Indem in einer Sendeantenne Ladungen hin- und hergeschoben werden, wird periodisch elektrische und magnetische Energie abgestrahlt, die sich in Form einer Welle im Raum ausbreitet. Eine Empfangsantenne wandelt diese Welle wieder in bewegte Ladungen um, die gemessen und so verwertet werden können. Vereinfacht gesagt, werden beim Sender Zeichen in Wechselspannungen und über die Antenne in Wellen

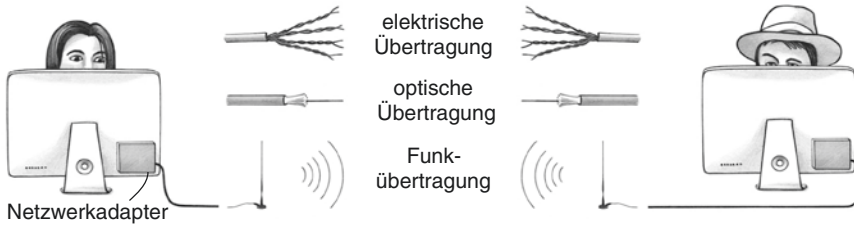


Abb. 6.3 Zwei Computer können über verschiedene Übertragungstechnologien gekoppelt werden. (© Susanne Staubli, Eckart Zitzler)

übersetzt und beim Empfänger wieder in Wechselspannungen bzw. Zeichen zurücktransformiert.

Um nun eine Sequenz von Nullen und Einsen von einem Computer an den anderen übermitteln zu können, müssen wir zunächst festlegen, durch welche Signale oder Signalfolgen die einzelnen Zeichen repräsentiert werden. Während es bei elektrischen und optischen Leitungen offensichtlich ist, wie die zwei Zeichen 0 und 1 repräsentiert werden können, ist das bei den elektromagnetischen Wellen nicht so klar. Zudem müssen wir uns überlegen, wie sich die Kommunikationspartner zeitlich abgleichen können. Wenn Alice von ihrem Computer aus eine Folge von Nullen sendet, woher weiß dann Bobs Computer, um wie viele Nullen es sich handelt? Ein Übertragungsschema muss beide Aspekte berücksichtigen. Je nach Kommunikationsmedium haben sich ganz unterschiedliche Lösungen etabliert, die auch als Leitungscode bezeichnet werden (siehe Bsp. 6.1).

Beispiel 6.1: Wie Bits über die Leitung geschickt werden

In Abb. 6.4 sehen Sie drei Möglichkeiten – für jede der erwähnten Übertragungstechnologien eine –, wie eine vorgegebene Bitfolge mittels eines Leitungscode als eine Folge von Signalen dargestellt werden kann.

Der Manchester-Code verwendet pro Bit, das zu übertragen ist, zwei Signale. Anstatt also die Bits eins zu eins über einen niedrigen bzw. hohen Spannungspegel – im Fall einer elektrischen Leitung – zu codieren, wird für jedes Bit ein Spannungswechsel eingebaut. Bei einer Null wird zuerst der untere Spannungspegel angelegt, anschließend der obere, bei einer Eins ist es genau umgekehrt. Über die Spannungswechsel lässt sich erkennen, wann ein Zeichen gesendet wird.

Der Block-Code verfolgt einen anderen Ansatz: Hier werden Gruppen von Bits codiert. Beispielsweise kann die originale 0-1-Sequenz in 4er-Gruppen aufgeteilt werden und jeder 4er-Block wird anschließend auf eine festgelegte Art und Weise in einen 5er-Block übersetzt, wie das in Abb. 6.4 für die optische Übertragung dargestellt ist. Was ist der Vorteil? Die 5er-Blöcke sind so gewählt, dass pro Block mindestens zwei Wechsel des Signalpegels (Licht an, Licht aus)

vorkommen – somit lässt sich das Zeitraster ebenfalls aus der Signalfolge eruieren. Im Gegensatz zum Manchester-Code werden weniger Bits „verschwendet“, nämlich nur ein Bit mehr pro 4 Bits und nicht doppelt so viele.

Beim Funk schließlich ist alles ein wenig anders, denn was ist ein Signal, wenn elektromagnetische Wellen übertragen werden? Der gemessene Spannungspegel wechselt ja ständig, nämlich wellenartig, und daher lässt sich nicht mehr so schön mit dem „An“ und dem „Aus“ operieren. Nun, die Idee ist, die Eigenschaften der Welle zu variieren, z. B. kann die Höhe der Welle – das nennt man Amplitudenmodulation – oder die Breite der Welle – das wird als Frequenzmodulation bezeichnet – modifiziert werden. Bei Funknetzen wird häufig die „Richtung“ der Welle verändert, was als Phasenmodulation bekannt ist. Was heißt das? Wie in Abb. 6.4 zu sehen, ist der Übergang zwischen den Bits das eigentliche Signal: Folgt eine 0, so ändert sich nichts an der Welle, folgt jedoch eine 1, so „kippt“ die Welle, d. h. sie wird um eine halbe Wellenlänge verschoben. Natürlich müsste noch das Zeitraster mitcodiert werden, aber wir wollen es hier bei dieser groben Übersicht belassen, im Detail wird es dann recht kompliziert.

Mit der Festlegung des Leitungscodes ist es allerdings noch nicht getan. Zwar kann Alice nun Bob eine 0-1-Sequenz übermitteln, doch woher weiß sie, dass ihre Nachricht auch korrekt ankommt? Es passiert immer wieder,

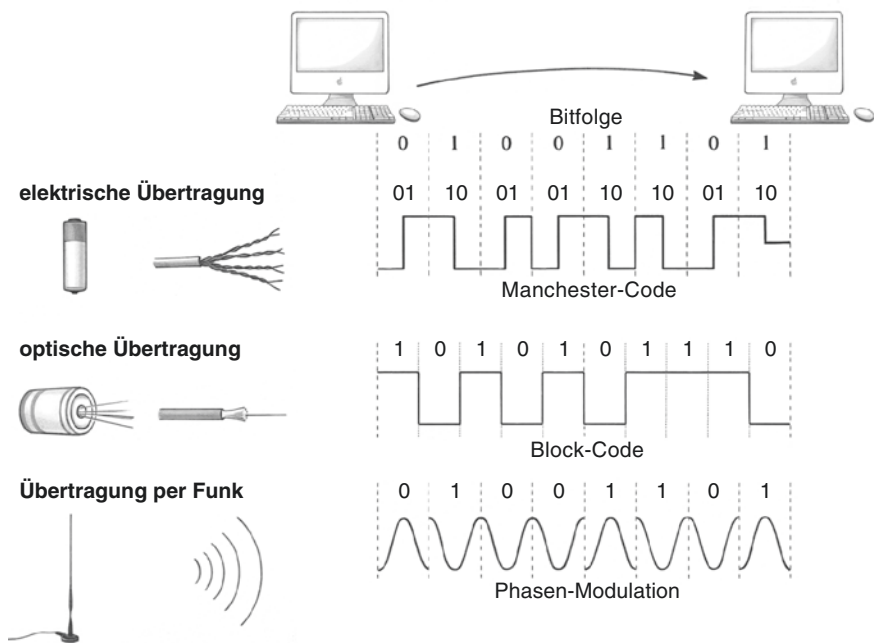


Abb. 6.4 Wie eine 0-1-Folge in verschiedenen Übertragungstechnologien über Signale codiert werden kann. (© Susanne Staubli, Eckart Zitzler)

insbesondere bei Funknetzen, dass Bits fehlerhaft übertragen werden – einfach weil es natürlicherweise Störungen gibt. Und was ist, wenn Bob etwas schicken möchte? Wie verständigen sich die beiden? Solche Fragen sind Bestandteil eines Protokolls, also eines festgelegten Schemas, nach dem die Kommunikation ablaufen soll. Protokolle haben wir im Zusammenhang mit den Bussen bereits in Kap. 4 angeschaut, und um nichts anderes handelt es sich hier auch. Protokolle regeln nicht nur, was genau übertragen wird, sondern auch die Art der Absprache, wer was zu welchem Zeitpunkt unternimmt. Protokolle können im Detail äußerst kompliziert und aufwändig sein, aber das Prinzip des Datenaustauschs ist kein Zauber. Bsp. 6.2 zeigt Ihnen, wie das beim Ethernet funktioniert, einer weit verbreiteten Netzwerktechnologie, mit der mehrere Computer in einem Gebäude oder im Umkreis weniger Kilometer miteinander verbunden werden können.

Beispiel 6.2: Das Ethernet-Protokoll

Beim Ethernet werden Nachrichten in kleine Blöcke aufgeteilt, die wie bei einem Brief mit Zusatzinformationen wie Empfänger- und Absenderadresse versehen und separat verschickt werden. Die resultierenden Sendeeinheiten werden Frames genannt und sind gemäß einem vordefinierten Format aufgebaut. In Abb. 6.5 sehen Sie ein Beispiel für einen Frame – der Einfachheit halber in dem Format, das ursprünglich im Ethernet verwendet und mittlerweile noch erweitert wurde. Ein Frame hat zunächst einmal eine Präambel, eine vorgegebene Bitfolge, die den Beginn eines Frames signalisiert und gleichzeitig den Empfänger auf den Datentransfer vorbereitet. Danach folgen die Kennungen des Empfängers und der Senderin, die beide eindeutig identifizieren; die Informationen über das Wohin, das Ziel, und das Woher, die Quelle, sind bei zwei Teilnehmern eigentlich überflüssig und spielen erst bei mehreren angeschlossenen Computern eine Rolle. Anschließend kommen die eigentlichen Daten. Sie sind ggf. mit Füllbits aufgestockt, weil die minimale Größe eines Datenblocks aus Gründen der sicheren Übertragung bei 64 Zeichen bzw. Bytes liegt. Daher wird auch die Länge der eigentlichen Datensequenz mitspezifiziert. Zum Schluss wird noch eine Prüfsumme mitgeschickt, eine Dualzahl, die aus den vorherigen Daten berechnet wird und als Indikator dient, ob die einzelnen Bestandteile korrekt übertragen wurden.

Und wie läuft die Übertragung im Gesamten ab? Nun, zu Beginn horchen Alice und Bob auf beiden Seiten der Leitung, ob es Änderungen am Signalpegel gibt, also der Beginn eines Frames gesendet wird. Jetzt sendet Alice den ersten Frame, Bob erkennt über die Präambel, dass Daten zu empfangen sind, und nimmt den Frame entgegen. Anschließend gibt es eine Pause, deren minimale Länge festgelegt ist, und Alice sendet den nächsten Frame usw. Bob seinerseits überprüft jeweils, ob die empfangenen Daten einen korrekten Frame repräsentieren und ob die Prüfsumme mit den Daten übereinstimmt; wenn nicht, ignoriert er den Frame. Letzteres ist natürlich problematisch, denn Alice weiß nicht, ob die Daten korrekt angekommen sind. Deshalb gibt es auf der Basis des Ethernet-Protokolls noch weitere Protokolle, die den Empfang von Frames bestätigen und ggf. die erneute Sendung anfordern.

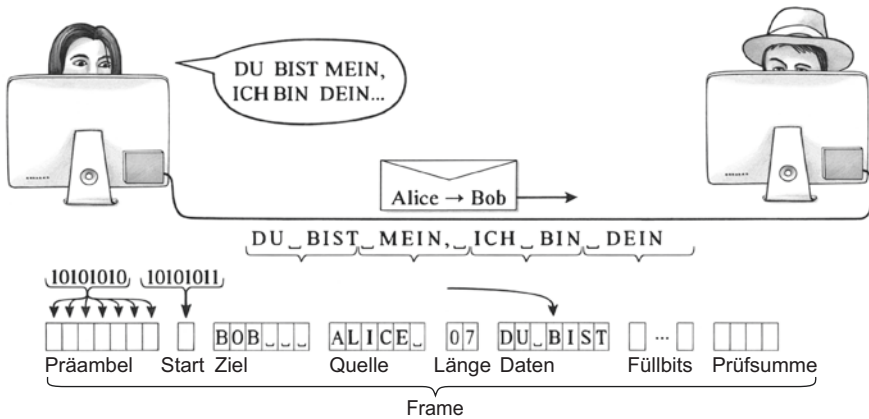


Abb. 6.5 Illustration, wie im Ethernet-Protokoll Daten mittels Frames, also verpackten Datenblöcken, übertragen werden. (© Susanne Staubli, Eckart Zitzler)

Damit wissen wir im Großen und Ganzen, wie Daten über eine Leitung oder per Funk von einem zu einem anderen Computer übertragen werden können. Beim Ethernet beispielsweise erfolgt die Übermittlung in wohlstrukturierten Bitbündeln, den Frames. Diese Funktionalitäten sind in der Regel auf der Hardware-Ebene implementiert und Bestandteil eines sogenannten Netzwerkadapters. Ein Netzwerkadapter ist an sich nichts anderes als ein weiteres Peripheriegerät eines Computers, das an den internen Bus angeschlossen ist. Der Netzwerkadapter stellt die Verbindung zum Netzwerk her und ist für ein bestimmtes Netzwerkprotokoll ausgerichtet. Wollen wir einen Computer an mehrere Netzwerke anschließen, z. B. an ein Funknetz und an ein Lichtfasernetz, so brauchen wir streng genommen für jedes Netzwerk einen eigenen Netzwerkadapter. Es gibt jedoch Adapter, die quasi mehrere Adapter in einem Gerät integrieren.

Die Familie wächst: eine Leitung für mehrere

Aus der Zweierkiste, die wir uns jetzt angeschaut haben, lässt sich auf einfache Art und Weise ein größerer Kommunikationsverbund konstruieren, und zwar indem wir die Computer paarweise verknüpfen. So ein vollvermaschtes Netzwerk lässt sich allerdings nur bei wenigen Teilnehmern realisieren: Bei zehn Computern benötigen wir bereits 45 Direktverbindungen, bei 100 Computern sind es sogar über 5000. So viele Verbindungen sind aber zumeist auch nicht notwendig, es genügt ja ein zusammenhängendes Gerüst, das alle Computer indirekt miteinander koppelt. Wenn die

Teilnehmer beispielsweise in einem Ring angeordnet sind, besitzt jeder Computer nur zwei Anschlüsse, nämlich zu seinem Vorgänger und zu seinem Nachfolger. Eine Nachricht von Alice wird in diesem Fall von Station zu Station weitergereicht, bis sie schließlich Bob erreicht. Die Extremform eines solchen Gerüsts ist ein Bus, d. h. ein einziger, gemeinsamer Kommunikationskanal, der von allen geteilt wird. Diese Topologie – sie kommt u. a. im Ethernet zum Einsatz – werden wir jetzt genauer unter die Lupe nehmen, um zu verstehen, wie die Kommunikation in einem größeren Verbund funktioniert.

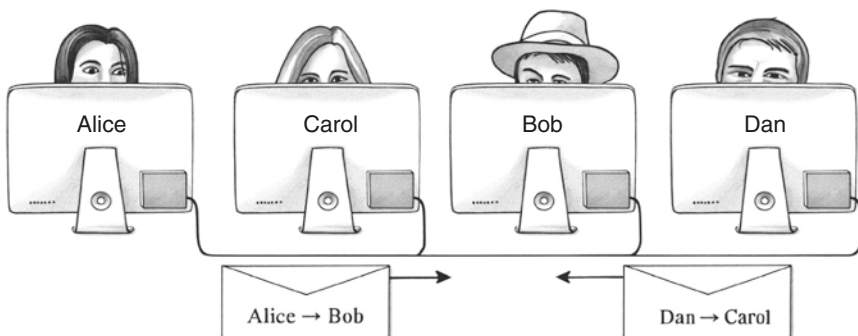
Sagen wir also, Carol und Dan gesellen sich zu Alice und Bob hinzu, sodass vier Computer dasselbe Übertragungsmedium nutzen. Wenn nun gleichzeitig Alice an Bob und Dan an Carol eine Nachricht senden möchten, ergibt sich die Frage, wer wann senden darf. Diese Situation lässt sich vergleichen mit einer Gesprächsrunde. Die einfachste Variante: Wir bestimmen eine Moderatorin, die den Teilnehmern das Rederecht erteilt. Diese Spielart kommt häufig bei Funknetzwerken zum Einsatz, wo das Senderecht von einer ausgewiesenen Station im Netz angefordert wird und das Senden erst nach Erhalt einer entsprechenden Bestätigung erfolgt. Eine andere Möglichkeit besteht darin, dass es eine „Redekarte“ gibt, die reihum weitergereicht wird; sprechen darf nur, wer aktuell im Besitz der Karte ist. Eine solche Lösung lässt sich einfach in Ringtopologien verwirklichen, wobei die Karte dann als Token bezeichnet wird. Die dritte Variante setzt auf Höflichkeit und ist in politischen Diskussionen kaum durchzusetzen. Jeder redet erst dann, wenn gerade kein anderer spricht. Ist die Bühne frei, redet er drauflos; wenn er allerdings feststellt, dass jemand anderes zum gleichen Zeitpunkt dieselbe Idee hatte, so verstummt er für einen Moment und probiert es später wieder. Dieser Ansatz wird im Ethernet verfolgt und ist im Detail in Bsp. 6.3 beschrieben.

Beispiel 6.3: Wie das Ethernet den Zugriff regelt

In Abb. 6.6 sind zwei Szenarien dargestellt, in denen Alice und Dan zu unterschiedlichen Zeitpunkten einen Frame über ein elektrisches Kabel an ihr jeweiliges Gegenüber Bob bzw. Carol schicken wollen. Im ersten Szenario ist Alice etwas früher dran als Dan. Sie prüft zunächst, ob die Leitung frei ist, und beginnt dann mit dem Senden. Genau dann beabsichtigt auch Dan zu senden; doch weil er den Kanal vorher abhört und merkt, dass er besetzt ist, wartet er so lange, bis Alice fertig ist, und sendet seinen Frame erst dann. Im zweiten Szenario möchten Alice und Dan zum nahezu gleichen Zeitpunkt senden. Beide hören zunächst den Kanal ab, mehr oder minder gleichzeitig, und nehmen ihn als frei wahr. Jetzt beginnen Alice und Dan zeitgleich, ihre Frames zu senden.

Es entsteht eine Überlagerung der Signale. Weil beide Netzwerkadapter die Leitung auch beim Senden abhören, wird diese Überlagerung früh erkannt, und Alice und Dan brechen ihre Übertragungen ab. Beide warten nun eine zufällig ausgewürfelte Zeit, bevor sie erneut das Medium abhören und wiederum zu senden versuchen. Durch das zufällige Auswürfeln ist sichergestellt, dass es höchst unwahrscheinlich ist, dass Alice und Dan wiederum gleichzeitig den Kanal abhören und erneut eine Kollision auftritt.

Im Fachjargon nennt sich dieses Verfahren CSMA/CD (*carrier sense multiple access with collision detection*). In Funknetzwerken ist es in der Regel nicht möglich, unmittelbar zu erkennen, ob Kauderwelsch entstanden ist bzw. ob es zu Kollisionen kam. Aus diesem Grund arbeitet man in Funknetzwerken häufig mit Bestätigungsframes, d. h., der Empfänger quittiert den Empfang eines Datenblocks. Bleibt die Quittierung aus, weiß der Sender, dass es eine Kollision oder einen Fehler in der Übertragung gegeben haben muss, und sendet den Block noch einmal. Zudem versucht man Kollisionen zu vermeiden, indem nach dem Abhören des Kanals und vor dem Senden noch eine zufällig gewählte Zeit verstreicht. Das nennt sich CSMA/CA, wobei CA für *collision avoidance* steht.



ohne Kollision

- ① Leitung frei?
- ② Frame senden
- ③
- ④

Leitung frei?
Leitung frei?
Frame senden

mit Kollision

- ① Leitung frei?
- ② Frame senden
- ③
- ④ zufällige Zeit warten
- ⑤ Leitung frei?

⚡ Kollision entdeckt ⚡

Leitung frei?
Frame senden
zufällige Zeit warten
Leitung frei?

Abb. 6.6 Wie beim Senden Kollisionen entstehen und aufgelöst werden können. (© Susanne Staubli, Eckart Zitzler)

Sobald mehr als zwei Teilnehmer an einem Kommunikationsverbund beteiligt sind, stellt sich auch die Frage nach der Identifizierung: Es braucht eine Kennung, die jeden Computer im Netzwerk eindeutig benennt. Weil viele Computer mobil und mal in einem, mal in einem anderen Netzwerk eingebunden sind, hat man sich ein Kennzeichnungssystem ausgedacht, das jedem Netzwerkadapter eine weltweit einmalige Nummer zuweist. Diese Nummer wird als MAC-Adresse bezeichnet – MAC steht für media access control, also Zugangskontrolle – und ist herstellerabhängig. Jeder Hersteller hat einen bestimmten Adressbereich zur Verfügung und vergibt die Nummern in eigener Regie.

Beim Ethernet umfassten die MAC-Adressen ursprünglich 48 Bit, die erste Hälfte ist fix für den Hersteller, die restlichen können für die Geräte vergeben werden, dienen also als Seriennummer. Traditionellerweise werden MAC-Adressen als Sequenzen von Hexadezimalzahlen, dem Zahlensystem zur Basis 16 (vgl. Kap. 2), dargestellt; in Abb. 6.7 sehen Sie ein Beispiel. Mittlerweile reicht der mit 48 Bits aufgespannte Adressraum nicht mehr aus, und man hat das Format auf 64 Bits ausgedehnt. Die Kennung nennt sich dann EUI-64-Adresse, EUI ist die Abkürzung für *extended unique identifier*. MAC-Adressen lassen sich einfach in EUI-64-Adressen überführen.

Jetzt kennen Sie die Konzepte, die einem Computerverbund mit einem Kommunikationskanal und mehreren Teilnehmern zugrunde liegen. Für das Ethernet lässt sich das wie folgt zusammenfassen: Eine Teilnehmerin verschickt einen Datenblock, indem sie ihn zusammen mit Zusatzinformationen über das Übertragungsmedium sendet; gleichzeitig achtet sie darauf, dass sie anderen nicht hineinredet. Während des Sendevorgangs hören alle anderen Teilnehmer mit. Der Teilnehmer, für den die Sendung bestimmt ist, erkennt dies an der mitgesendeten Adresse, alle anderen Teilnehmer ignorieren den Datenblock. „Warte mal!“, könnten Sie hier einwerfen, „dann kann ja jeder mithören!“ Ja, das stimmt, und tatsächlich sollte

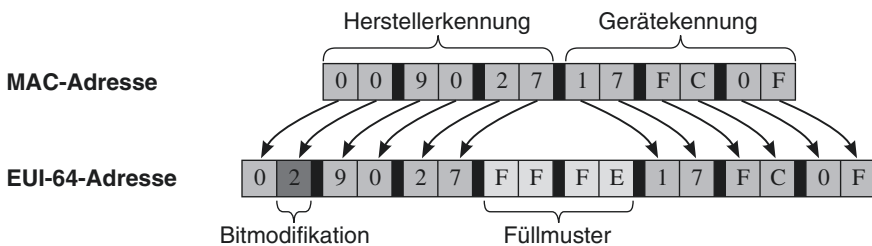


Abb. 6.7 Aufbau von MAC- und EUI-64-Adressen. (© Susanne Staubli, Eckart Zitzler)

man sich in solchen Netzwerken – wenn man nicht ausschließlich unter Bekannten kommuniziert – gegen das Mithören absichern, z. B. über die Verschlüsselungsmethoden, die in Kap. 7 behandelt werden.

Von der Familie zur Gemeinschaft: Leitungen überbrücken

Das Ethernet entstand in den 1970er-Jahren in den USA, wurde in den 1980ern standardisiert und etablierte sich in den 1990ern als die am weitesten verbreitete kabelgebundene Netzwerktechnologie für örtlich eingegrenzte Netzwerke, sogenannte LANs (LAN ist eine Abkürzung für *local area network*). Als jedoch immer mehr Computer hinzukamen und größere Distanzen zu überwinden waren, kam das „Ein gemeinsamer Kanal für alle“-Konzept an seine Grenzen. Zwar bietet es hohe Flexibilität – Teilnehmer können einfach hinzugefügt oder entfernt werden –, doch mit der Anzahl der involvierten Computer steigt auch die Anzahl der Kollisionen. Ist eine kritische Grenze überschritten, warten die meisten Teilnehmer auf die Leitungsfreigabe und der Datenverkehr stagniert. Des Weiteren darf das alle verbindende Kabel nicht beliebig lang werden, weil ansonsten die Signale zu schwach und zu stark verzögert am anderen Ende der Leitung ankommen. Und schließlich kamen mit der Zeit Funknetze, *wireless local area networks* oder kurz WLANs, hinzu; damit stellte sich die Frage, wie drahtlose und drahtgebundene Technologien gekoppelt werden können. In der Folge wurden Geräte entwickelt, die gleich mehrere Fliegen mit einer Klappe schlagen. Sie lassen sich unter dem Begriff der Bridge subsumieren.

Eine Bridge, der Name sagt es, schlägt eine Brücke zwischen zwei Übertragungskanälen und schließt sie zu einem Kommunikationsverbund zusammen. Sie verhält sich auf beiden Seiten wie eine unsichtbare Teilnehmerin, die nichts anderes macht, als die gesendeten Datenframes zu lesen und auf dem jeweils anderen Kanal weiterzuverbreiten. In der klassischen Fernübertragung entspräche dies einer Relaisstation, die möglicherweise auf einem Berg platziert ist und zwei Täler miteinander verbindet (siehe Abb. 6.8). In einem Tal verständigt man sich vielleicht mittels Flaggen, im anderen über Lichtsignale. Damit erfüllt eine Bridge zwei Funktionen. Zum einen dient sie als Signalverstärker und erlaubt so, die Kanallänge beliebig zu erweitern. Zum anderen kann sie verschiedene Übertragungsmedien koppeln, z. B. ein Funknetz mit einem drahtgebundenen Netzwerk, die jeweils verschiedene Übertragungsgeschwindigkeiten besitzen. In letzterem Fall müssen die Daten vor der Weiterleitung entpackt

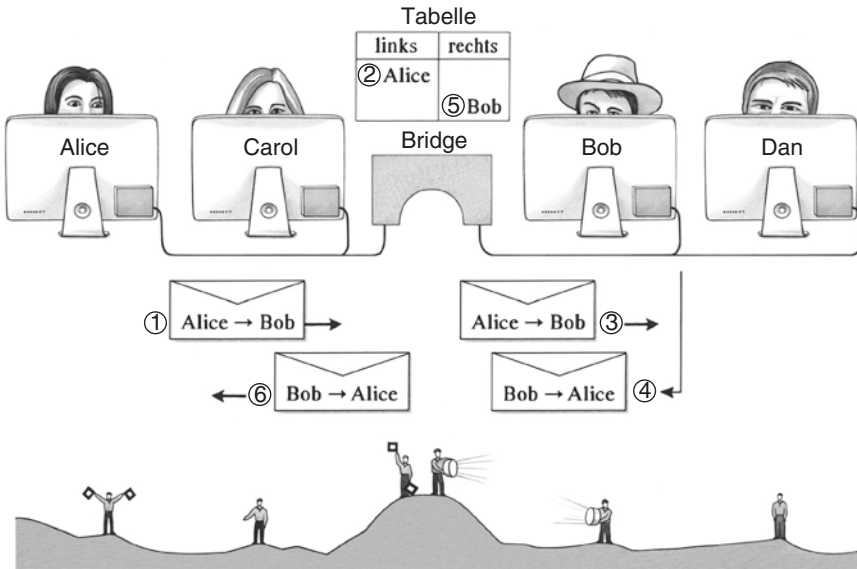


Abb. 6.8 Funktionsweise einer Bridge. (© Susanne Staubli, Eckart Zitzler)

und gemäß dem Netzwerkprotokoll auf der anderen Seite neu verpackt werden. Und schließlich kann eine Bridge die Belegung der Kanäle optimieren. Wenn sie nur diejenigen Frames weiterleitet, die tatsächlich auf die andere Seite gehören, kommen sich die Teilnehmer links und rechts weniger ins Gehege. Das bedingt natürlich, dass die Bridge weiß, an welchem Kanal welcher Computer angeschlossen sind. Dies ließe sich fix einprogrammieren, doch es gibt flexiblere Lösungen, wie Bsp. 6.4 erläutert.

Beispiel 6.4: Eine Bridge als Filter

Eine Bridge unterhält eine einfache Tabelle, die alle MAC-Adressen zur Linken und zur Rechten auflistet. Zu Beginn ist diese Tabelle leer, die Bridge lernt erst mit der Zeit anhand der versendeten Frames, welche Teilnehmer auf welcher Seite angeschlossen sind. Abb. 6.8 illustriert den Vorgang, wie die Tabelle gefüllt wird. Zuerst sendet Alice einen Frame mit Bob als Empfänger. Alle Computer einschließlich der Bridge, die am selben Kabel wie Alice angeschlossen sind, können diesen Frame empfangen. Die Bridge wird aus dem gesendeten Frame schließen können, dass Alice auf der linken Seite lokalisiert ist, und trägt das in ihre interne Tabelle ein. Da die Bridge zu diesem Zeitpunkt keine Informationen über Bob hat, wird sie den Frame von Alice auf der rechten Seite weiterleiten. Bob wiederum kann diesen Frame lesen und auspacken, weil er

sich als Empfänger erkennt. Anschließend sendet er einen Frame an Alice. Die Bridge empfängt diesen Frame ebenfalls und trägt die MAC-Adresse von Bob in ihrer Tabelle ein; sie weiß nun, dass Alice links und Bob rechts lokalisiert sind. Würde jetzt noch Carol an Alice einen Frame schicken, so würde die Bridge diesen nicht auf der rechten Seite weiterleiten.

Die selektive Weiterleitung ist der entscheidende Vorteil einer Bridge. Dadurch wird die Anzahl der Kollisionen massiv reduziert im Vergleich zur Situation, bei der alle Teilnehmer am selben Übertragungsmedium hängen. Tritt eine Kollision auf der linken Seite auf, ist die rechte Seite davon nicht betroffen – und umgekehrt. Das heißt auch, Bridges können bewusst eingebaut werden, um Übertragungskanäle in mehrere Netzwerksegmente aufzuteilen und so die Kollisionsbereiche zu verkleinern. Auf diese Weise lassen sich Ketten von Netzwerksegmenten kreieren, die durch Bridges verbunden sind. Wir können Bridges aber nicht nur einsetzen, um zwei Teilnehmergruppen zu koppeln, sondern sie auch selbst zu einem Verbund zusammenfügen. Auf der einen Seite einer Bridge befindet sich dann eine Gruppe von Computern, auf der anderen Seite eine Gruppe von weiteren Bridges. Und das Prinzip lässt sich beliebig weiterspinnen: Gruppen von Bridges können wiederum über Bridges miteinander verbunden werden. Wie in Abb. 6.9 zu sehen ist, ergibt sich so eine hierarchische Baumstruktur, wie wir sie in Kap. 5 kennengelernt haben. Jetzt entsteht tatsächlich ein Gebilde, das dem Begriff des Netzes gerecht wird!

Die hierarchische Konstruktion hat zum Ziel, dass Datenframes auf dem Weg zum Empfänger möglichst wenige Kollisionsbereiche passieren müssen. Es ist sogar möglich, Kollisionen gänzlich zu vermeiden, indem eine Weiterentwicklung der Bridge, der Switch, zum Einsatz kommt. Sie können sich einen Switch vorstellen als ein Gehäuse um einen Bridge-Verbund mit einem Übertragungskanal in der Mitte, sodass nach außen nur die einseitigen freien Anschlüsse der Bridges sichtbar sind. Intern ist ein Switch aber über eine Schaltung realisiert, die Frames direkt – also nicht über ein kollisionsgefährdetes Kommunikationsmedium – von einem Anschluss zu einem anderen Anschluss weiterreichen kann. Wenn wir einzelne Computer unmittelbar an Switches anhängen und diese Frame-Verteilzentren nur direkt miteinander koppeln, können keine Kollisionen mehr auftreten.

Mit Bridges und Switches lassen sich im Prinzip beliebige Kommunikationsstrukturen realisieren, doch je größer solche Netzwerke in der Realität werden, desto unübersichtlicher werden sie auch. Und dann können auf einmal ungeahnte Probleme entstehen, wie Anm. 6.1 aufzeigt.

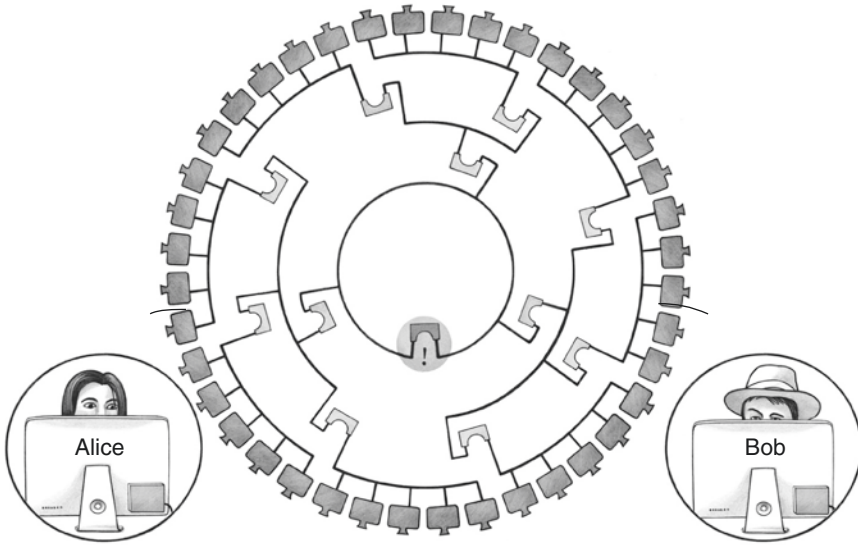


Abb. 6.9 Ein hierarchisches Computernetzwerk, in dem Netzwerksegmente über ein Netz aus Bridges verbunden sind; in der Mitte ist ein potenzielles Problem markiert. (© Susanne Staubli, Eckart Zitzler)

Anmerkung 6.1: Wie man ein Netzwerk lahmlegt

Werfen Sie einen Blick auf Abb. 6.9 und betrachten den hervorgehobenen runden Ausschnitt, der mit dem Ausrufezeichen versehen ist. Die Bridge, die in diesem Ausschnitt hervorgehoben ist, führt zu einem Kreis im Netzwerk. Wieso ist das ein Problem?

Nun, betrachten wir dazu die Anfangssituation, in der alle Bridge-Tabellen leer sind. Will Alice an Bob einen Frame schicken, so wird dieser effektiv alle Bridges und Kanäle passieren, weil ja noch nichts bekannt ist über die Verortung einzelner Computer. Wenn der Frame den inneren Kreis in Abb. 6.9 erreicht, erhält unsere markierte Bridge diesen Frame einmal von links und einmal von rechts (auf der gleichen Leitung wohlgemerkt). Doch was soll die Brücke nun in ihrer Tabelle eintragen: Ist Alice nun links oder rechts? Diese Widerspruchssituation kann in der Tat das ganze Netzwerk lahmlegen und zu erheblichen Problemen führen.

Aus diesem Grund hat man spezielle Protokolle erfunden, mit denen Bridges miteinander kommunizieren und Kreise erkennen und eliminieren können. Eliminieren heißt in diesem Fall einfach, dass bestimmte Bridge-Anschlüsse temporär deaktiviert werden, in unserem Beispiel könnte das der rechte Anschluss der markierten Bridge sein. Das ist zwar kompliziert und aufwändig, erlaubt aber andererseits, redundante Brücken einzubauen und somit den Ausfall von Brücken – das kommt immer wieder vor – zu kompensieren.

Fassen wir zusammen: Bridges helfen einerseits, Netzwerke auf beliebige Distanzen hinaus auszuweiten, und andererseits Kollisionen, die durch die Mehrfachnutzung von Übertragungsmedien entstehen, zu reduzieren oder – mit Switches – gänzlich zu vermeiden. Dennoch bleibt ein durch Bridges und Switches verbundenes Computernetzwerk vom Prinzip her eine „Alle hören mit“-Gemeinschaft. Denn wann immer die Tabellen der Bridges leer sind, müssen alle Frames in alle Richtungen weitergeleitet werden, wodurch wiederum ein enormer Datenverkehr entsteht. Weltweit lässt sich auf diese Weise kein verbundenes Netzwerk bauen. Dafür braucht es eine weitere Ausbaustufe, nämlich das Internet.

Netze vernetzen

Das Internet ist ein weltumspannendes Netzwerk, genau genommen ein Netzwerk aus Netzwerken, das als Zwischennetz gedacht ist. Es hat die Aufgabe, viele Computerverbünde – die zugrunde liegenden Technologien können sich unterscheiden, das Ethernet ist nur ein prominentes Beispiel – zu einem kohärenten Gesamtverbund zusammenzuschließen. Einerseits muss es riesige Datenverkehrsaufkommen bewältigen können und erweiterbar sein. Andererseits muss jede Nachricht zuverlässig ihren Weg von einem beliebigen Sender zu einem beliebigen Empfänger finden, auch wenn niemand die Topologie des Internets im Detail kennt und die Teilnehmer in lokalen Netzwerken mit unterschiedlichen Frame- und Adressformaten eingebunden sind. Dabei stellt sich nicht nur die technologische Herausforderung, eine bestehende Infrastruktur zu erweitern. Es geht auch darum, die Nutzung der Infrastruktur zu regeln, d. h. das Adresssystem und den Datenverkehr zu vereinheitlichen. Wie langwierig so etwas sein kann, führt uns die traditionelle Post vor Augen.

Der internationale Postverkehr hat eine lange Entwicklung hinter sich und konnte erst Wirklichkeit werden, als verschiedene Postsysteme miteinander abgeglichen wurden und so zusammenwuchsen. Die Ursprünge des Postwesens reichen bis weit zurück in die Geschichte, bis nach Ägypten und Persien. Es wurden Boten eingesetzt, sowohl zu Fuß als auch zu Pferde, aber auch per Schiff, die schriftliche Nachrichten überbrachte. So entstanden erste Systeme der Nachrichtenübermittlung. Später kamen dann noch Brieftauben hinzu. Es wurden damals auch schon Relaisstationen eingesetzt; dort konnten z. B. berittene Boten ihre Pferde wechseln. Es sollte aber noch dauern, bis aus solchen Kurierdiensten eine Post für alle wurde. Im deutschsprachigen Raum bildeten sich zunächst einfache Linien, Postkutschen



Abb. 6.10 Das Weltpostdenkmal in Bern. (© Judith Büsser)

z. B. von Leipzig nach Hamburg – das war im 17. Jahrhundert. Es kamen mit der Zeit immer mehr Linien hinzu mit der Konsequenz, dass sich im 19. Jahrhundert über ein Dutzend selbstständige Postgebiete gebildet hatten, und jedes funktionierte nach seinen eigenen Bestimmungen. Um die Linien zu verbinden, brauchte es Absprachen, z. B. die einheitliche Einführung der Briefmarke, die Abstimmung der Tarife und eine entsprechende Transitfreiheit. Beschleunigt wurde das Ganze schließlich mit der Eisenbahn, über die größere Distanzen in kürzerer Zeit zurückgelegt werden konnten. Bald brauchte es ein noch größeres, übergeordnetes System, die internationale Post, bei dem es sich im Wesentlichen um eine Übereinkunft handelt. Zu diesem Zweck wurde 1874 in Bern ein Allgemeiner Postverein gegründet, dem zu Beginn 22 Staaten beitraten; das Weltpostdenkmal erinnert daran (Abb. 6.10). Vier Jahre später wurde aus dem Allgemeinen Postverein der Weltpostverein, der bis heute die internationale Zusammenarbeit der Postbehörden ebenso regelt wie den grenzüberschreitenden Postverkehr.

Auf Routen von Netz zu Netz

Zurück zum Internet. Konzentrieren wir uns zunächst auf die Hardware-Infrastruktur, die verschiedene Computernetzwerke – man spricht in diesem Zusammenhang von Netzwerksegmenten – koppelt, und blenden den Aspekt des netzwerkübergreifenden Adresssystems für den Moment aus. An sich ist die Idee der Kopplung naheliegend, wir müssen nur das Konzept der Bridge weiterspinnen. Es braucht ein Gerät, das unterschiedliche Netzwerktypen unterstützt und die entsprechenden Protokolle beherrscht. Dieses Gerät heißt Router. Im Gegensatz zu einer Bridge, die im Verborgenen agiert, ist ein Router in allen angeschlossenen Netzwerksegmenten als regulärer Teilnehmer sichtbar. Wann immer

ein Computer eine Nachricht an einen Computer in einem anderen Netzwerksegment schicken will, sendet er sie an den Router, der als Tor zur Außenwelt fungiert. Der Router nimmt die Nachricht entgegen, decodiert sie vollständig, verpackt sie anschließend wieder gemäß dem Frameformat im anderen Netzwerksegment – möglicherweise ergeben sich sogar mehrere Frames – und leitet sie an den Empfänger weiter.

Ein Router kann wie eine Bridge lediglich zwei oder wie ein Switch mehrere Netzwerksegmente bedienen, wie in Abb. 6.11 links dargestellt ist. Allerdings ist ein zentraler Router, der alle Netzwerke der Welt miteinander verbindet, keine reale Option. Das Gesamtnetzwerk hätte dann die Form eines Sterns, in der Mitte der Router und außen als Zacken die einzelnen Netzwerksegmente. Stattdessen können wir ein Netzwerk von Routern in die Mitte setzen, das eine Art Zwischennetz bildet (siehe Abb. 6.11 rechts). Dieses Zwischennetz kann verschieden organisiert sein: wie vorher mittels gemeinsamer Kommunikationskanäle, per Direktverbindungen oder in Mischformen, zumeist hierarchisch. Das Internet besteht aus vielen solcher Routernetzwerke und kennt verschiedene Hierarchiestufen. Beispielsweise betreiben Internetdienstanbieter ihre eigenen Routernetzwerke, die wiederum auf einer höheren Ebene miteinander verknüpft sind; zudem gibt es Routernetzwerke, die auf direktem Weg Länder und Regionen verbinden.

Die Art und Weise, wie Nachrichten durch ein Routernetzwerk hindurchgeschleust werden, ähnelt der Wegleitung in einem Switch-Netzwerk. Die Router verfügen über Tabellen – Routing-Tabellen genannt –, anhand derer sie entscheiden, wohin ankommende Nachrichten weitergeleitet werden. Allerdings sind in einer Routing-Tabelle nicht Einzeladressen, sondern Adressbereiche aufgeführt, zusammen mit den jeweils zuständigen Routern.

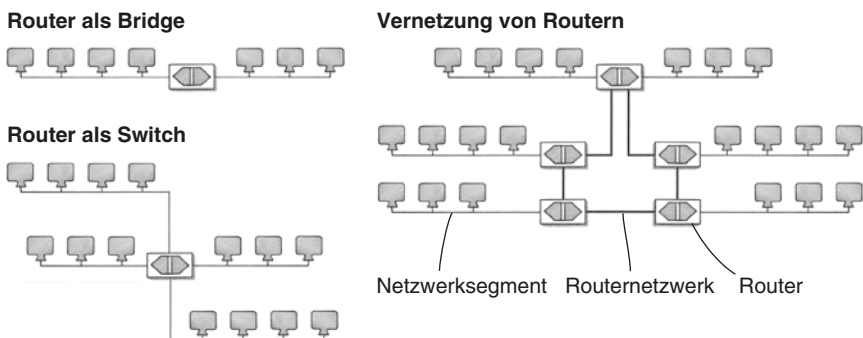


Abb. 6.11 Wie mehrere lokale Netzwerke über einen Router verbunden werden können. (© Susanne Staubli, Eckart Zitzler)

Schließlich kann ein Router ja weder die Adressen aller Endgeräte im Internet speichern, noch eine Nachricht einfach mal in alle Richtungen versenden, wenn der Empfänger unbekannt ist. Die Tabelleneinträge können von Hand gesetzt werden oder die Router tauschen die Informationen, welche Netzwerksegmente wo angeschlossen sind, automatisch aus. Routing-Tabellen können zudem alternative Routen enthalten, was den Routern die Möglichkeit gibt, je nach Belastung der Übertragungskanäle flexibel zu reagieren. In Bsp. 6.5 können Sie nachlesen, wie so eine Nachrichtenübermittlung via Router konkret abläuft.

Beispiel 6.5: Der Weg durchs Zwischennetz

Nehmen wir an, Alice wollte in dem in Abb. 6.12 dargestellten Szenario eine Nachricht an Bob schicken, dessen Computer in einem anderen lokalen Netzwerk eingebunden ist. Als Erstes kontaktiert Alice den Router G1 in ihrem Netzwerksegment, da Bob ja in einem anderen Netzwerksegment lokalisiert ist. Sie schickt ihm einen Frame, der im Datenteil die Nachricht an Bob sowie Bobs Adresse enthält: „Bob/Hufflepuff/1. Obergeschoss“ (die effektive Adresse sieht natürlich anders aus, aber das spielt hier keine Rolle). Der Router G1 nimmt den Datenframe auseinander, analysiert Bobs Adresse und schaut intern in seiner Tabelle nach. Dort ist die Adresse nicht explizit aufgeführt, sondern nur vermerkt, dass Router G4 für alle unbekannten Adressen zuständig ist. Daher nimmt G1 zu G4 Kontakt auf und übermittelt einen Frame, dessen Datenteil wiederum Alices Nachricht und Bobs Adresse beinhaltet. Der Router G4 verfährt nun genauso wie G1, d. h., er entpackt den von G1 erhaltenen Frame, schaut die Zieladresse nach und gleicht diese mit seiner Tabelle ab. Im Prinzip könnte er die Nachricht nun Richtung S4 oder Richtung R4 schicken, doch in der Tabelle ist nur der Eintrag R4 vermerkt, weshalb G4 wiederum einen Frame an R4 schickt. R4 wiederholt das Ganze, kommuniziert nun mit H4, H4 leitet den Datenblock weiter an H2, und H2 schließlich liefert die Nachricht an Bob aus.

Fassen wir zusammen: Router verbinden Netzwerke miteinander und agieren in diesen wie normale Kommunikationspartner, nur dass bei ihnen keine Benutzer dahintersitzen. Sie nehmen Nachrichten entgegen und vermitteln diese an Empfänger in anderen Netzwerksegmenten weiter. Dabei ist ihre Aufgabe, Routen zu finden, auch wenn sie nicht notwendigerweise den gesamten Weg kennen müssen: Sie leiten jede Nachricht an einen nächsten Router weiter, und von dort aus geht es dann ggf. über weitere Router bis ins Zielnetzwerk zum Empfänger. Dafür ist ein neues Protokoll erforderlich, das die Kommunikation zwischen Sender und Router, Router und Router bzw. Router und Empfänger regelt – und es braucht ein einheitliches Adresssystem.

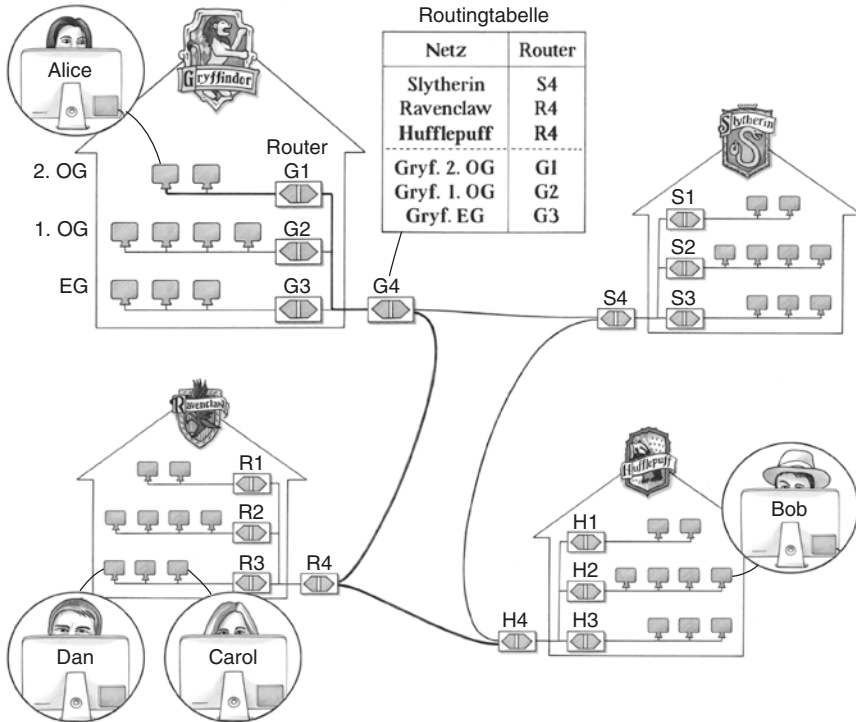


Abb. 6.12 Ein hierarchisches Routernetzwerk, speziell für Fans von Joanne K. Rowlings „Harry Potter“-Romanreihe. (© Susanne Staubli, Eckart Zitzler)

Weltweite Anschriften

Das bekannteste Protokoll für solche Zwischennetze ist das Internet-Protokoll, kurz IP. Es ist weltweit verbreitet und legt die grundlegenden Spielregeln für die Kommunikation im Internet fest. Weil sich die Art und Weise, wie Nachrichten als Frames formatiert werden, je nach Netzwerktyp unterscheiden kann, hat man ein eigenes Nachrichtenformat aufgesetzt. Analog zum Ethernet werden Nachrichten in Übertragungseinheiten unterteilt, die allerdings als Datenpakete oder Datagramme bezeichnet werden. Datenpakete werden auf ihrem Weg in ein oder mehrere Frames verpackt – je nachdem, welcher Netzwerktyp gerade passiert wird. Und Datagramme enthalten wie Frames neben den eigentlichen Daten verschiedene Zusatzinformationen, u. a. die Sender- und Empfängeradressen. Bei den Adressen handelt es sich allerdings diesmal um netzwerkübergreifende

Kennungen, anhand derer Sender und Empfänger weltweit eindeutig lokalisiert werden können. Schauen wir uns jetzt an, wie diese Kennungen aussehen und wie das globale Adresssystem des Internets konzipiert ist.

Eine Internetadresse bzw. IP-Adresse ist eine Dualzahl mit einer fixierten Stellenzahl und wie eine Adresse bei der klassischen Briefpost hierarchisch organisiert (siehe Abb. 6.13). Die Hierarchie drückt sich dadurch aus, dass die Bits der Dualzahl von links aus immer weiter festgezurret werden. Die oberste Hierarchiestufe im Internet sind Regionen. Für jede Region gibt es einen Adressbereich, der von einer regionalen Adressverwaltung, einer *regional internet registry*, verantwortet wird. Dieser Adressbereich ist durch ein Präfix beschrieben – das ist eine spezifische Bitfolge, die die linke Seite der Dualzahl festlegt. Die darunter liegende Hierarchiestufe bilden die großen Internetdienstanbieter, die von der jeweiligen *regional internet registry* ein Präfix zugeordnet bekommen, das sie nicht verändern können; die restlichen Stellen des Adressbereichs können sie hingegen frei einteilen. Die Kunden wiederum erhalten weiter eingeschränkte Adressbereiche, also nochmals erweiterte Präfixe, anhand derer sie den in ihrem lokalen Netzwerk angeschlossenen Computern eindeutige Adressen zuordnen können. Wer wissen möchte, wie solche IP-Adressen konkret aussehen, findet in Anm. 6.2 zwei Beispiele.

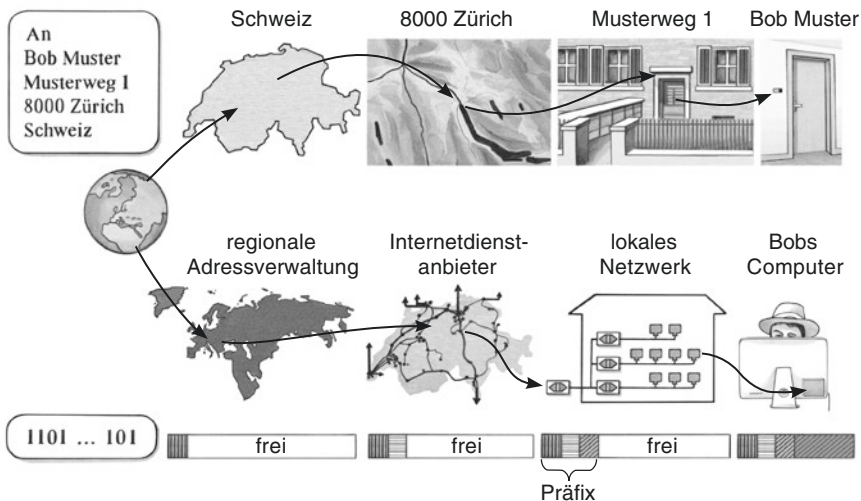


Abb. 6.13 Gegenüberstellung der traditionellen Adressierung bei der Briefpost (oben) und der Internetadressierung (unten). (© Susanne Staubli, Eckart Zitzler)

Anmerkung 6.2: Wie Adressen im Internet aussehen

Das Adresssystem, das bei der Verbreitung des Internets eingesetzt wurde, bezieht sich auf die Version 4 des Internet-Protokolls, deswegen wird manchmal abgekürzt von IPv4-Adressen gesprochen. IPv4-Adressen sind 32 Bits lang, sie werden in der Regel als eine Folge von vier Dezimalzahlen geschrieben, wobei jeweils 8 Bits zu einer Dezimalzahl zwischen 0 und 255 zusammengefasst werden und die Dezimalzahlen durch Punkte getrennt sind. Eine gültige Adresse wäre z. B. die Dualzahl 110000001010100000000000001111. Schreiben würde man sie als 192.168.0.15 in menschenlesefreundlicherer Form.

Als das IPv4-Adresssystem 1981 konzipiert wurde, konnte man sich nicht vorstellen, dass man mit 32-Bit-Adressen jemals an eine Grenze stoßen könnte. Doch bereits einige Jahre nach dem Jahrtausendwechsel wurden die Adressen knapp. Deswegen hat man sich frühzeitig um einen größeren Adressraum gekümmert, das IPv6-Adresssystem. IPv6-Adressen sind Teil der Version 6 des Internet-Protokolls. Sie bestehen aus 128 Bits und können eine unvorstellbar große Anzahl von Teilnehmern eindeutig identifizieren. IPv6-Adressen werden nochmals anders geschrieben, nämlich als Folge von vierstelligen Hexadezimalzahlen, die durch Doppelpunkte getrennt sind. Eine Hexadezimalziffer repräsentiert 4 Bits, eine vierstellige Hexadezimalzahl also 16 Bits. Eine konkrete IPv6-Adresse könnte z. B. so aussehen: fe80:0000:0000:0000:3ab0:01fe:2110:04cc.

Weil die IP-Adressen für die Benutzer so unpraktisch sind, hat man symbolische Namen eingeführt, sogenannte Domains. Eine Domain steht für einen Adressbereich, der über ein Präfix festgelegt ist. Dieses Namenssystem ist in Analogie zum numerischen Adresssystem hierarchisch strukturiert, wird aber von rechts nach links aufgebaut und die Bestandteile werden durch Punkte getrennt. So ergeben sich dann zusammengesetzte Namen wie [bern.ch](#), [berlin.de](#) oder [wien.at](#). Die Namen auf der obersten Ebene heißen Top-Level-Domains, ihre Bezeichner stehen ganz rechts in der symbolischen Adresse. Sie wurden eingeführt, um Anwendungsbereiche zu trennen. In Abb. 6.14 sehen Sie die Top-Level-Domains, die es zu Beginn des Internets gab, also vor 1998; Länder-Domains wie .ch, .de und .at kamen erst später hinzu. Und klar, es braucht noch eine Übersetzung von Namen in numerische IP-Adressen, dafür gibt es spezielle Computer im Netzwerk – DNS-Server, wobei DNS für *domain name system* steht –, die Übersetzungstabellen bereithalten.

Das System der globalen Internetadressen wurde nachträglich geschaffen und quasi über die bestehenden Netzwerke gestülpt. Allerdings ersetzt es nicht die bestehenden Adresssysteme in den diversen Netzwerktypen, z. B. die MAC-Adressen beim Ethernet. Es wird parallel zu diesen betrieben, sodass lokale Netzwerke dezentral und individuell verwaltet werden können.

Endung	englische Bezeichnung	ursprüngliche Verwendung
.com	commercial	kommerzielle Unternehmen
.org	organization	Non-Profit-Organisationen
.net	network	Internetdiensteanbieter
.int	international organizations	internationale Organisationen
.edu	education	Bildungsinstitutionen
.gov	U.S. national and state government agencies	Regierungsbehörden der USA
.mil	U.S. military	Streitkräfte der USA

Abb. 6.14 Die ersten Top-Level-Domains im Internet. (© Susanne Staubli, Eckart Zitzler)

Die Frage ist nur, wie ein friedliches Nebeneinander realisiert werden kann. Die Lösung liegt auf der Hand: Innerhalb eines Netzwerksegments werden die typeigenen Adressen verwendet, und sobald es über einen Router nach draußen geht, kommen die IP-Adressen zur Anwendung. Allerdings bedarf es eines Verfahrens, bei dem die Adressen transformiert werden können. Bsp. 6.6 demonstriert, wie das beim Ethernet funktioniert: die Umsetzung von IP-Adressen, die auch als logische Adressen bezeichnet werden, in MAC-Adressen, die die physikalischen Adressen darstellen.

Beispiel 6.6: Logische in physikalische Adressen umsetzen

Schauen wir uns noch einmal Abb. 6.12 an und gehen diesmal davon aus, dass Carol etwas an Dan senden möchte und sie einzig seine Internetadresse kennt, jedoch nicht weiß, ob er sich im gleichen Netzwerksegment befindet. In diesem Fall schickt Carol einen Frame an eine spezielle reservierte MAC-Adresse, die Broadcast-Adresse genannt wird; die Broadcast-Adresse ist unbenutzt und alle wissen, dass damit eine Nachricht an alle geschickt wird. Die Nachricht enthält Carols IP-Adresse, ihre MAC-Adresse und die IP-Adresse, an die sie senden will. Dan wird die Nachricht lesen, erkennen, dass er gemeint ist, und anschließend einen Frame an Carol zurückschicken, damit sie seine MAC-Adresse erhält. Von da ab ist dann alles wie gehabt.

Wenn Dans Computer an ein anderes Netzwerksegment angeschlossen wäre, würde Dans Antwort ausbleiben. In dieser Situation kontaktiert Carol den Router – seine IP-Adresse hat sie fest einprogrammiert. Mit dem Router kommuniziert sie auf die gleiche Weise, wie es das lokale Netzwerkprotokoll festlegt. Daneben teilt sie dem Router noch mit, an wen sie das Paket eigentlich senden will, nämlich an die IP-Adresse von Dan.

Diesen auf die bestehende Netzwerkinfrastruktur, z. B. das Ethernet, aufgesetzten Vorgang der Adressauflösung – von der logischen IP-Adresse zur physikalischen MAC-Adresse – nennt man übrigens ARP-Anfrage, wobei ARP für *address resolution protocol* steht.

Die Doppelspurigkeit bei den Netzwerkadressen hat zur Konsequenz, dass Router zum einen eine IP-Adresse und zum anderen in jedem angeschlossenen Netzwerksegment eine physikalische Adresse besitzen. In den Routing-Tabellen sind jedoch nur IP-Adressbereiche vermerkt, also Präfixe für Internetadressen.

Die Stockwerke des Internets

So, jetzt haben wir uns die Grundlagen von Computernetzwerken und des Internets erarbeitet. Dabei müssen wir uns vor Augen halten: Niemand hat das Internet auf dem Reißbrett geplant, es ist sukzessive aus verschiedenen Netzwerken und Technologien entstanden und entwickelt sich permanent weiter. Damit das Ganze nicht in völligem Wildwuchs ausartet, hat man frühzeitig versucht, der Netzwerkarchitektur – hierbei ist nicht die Topologie, sondern das Zusammenspiel der Protokolle gemeint – eine Struktur zu geben. Die Internationale Organisation für Standardisierung (ISO) hat dazu 1984 das OSI-Schichtenmodell veröffentlicht, welches Konzeptionsebenen von Netzwerken definiert und jeder Ebene eine klare Funktionalität zuweist. Das OSI-Modell – OSI steht für *Open Systems Interconnection* – legt keine Protokolle fest, es hilft vielmehr, Protokolle zu verorten. Auf jeder Ebene können verschiedenste Protokolle zur Anwendung kommen, um das gewünschte Verhalten zu erzielen. Abb. 6.15 gibt Ihnen einen Eindruck von diesem Schichtenmodell und illustriert, wie die untersten Ebenen aufeinander aufbauen: Eine Textnachricht wird mit den IP-Adressen von Sender und Empfänger versehen und zu einem Datenpaket verschnürt; das Datenpaket wird ergänzt um die MAC-Adressen von Sender und Empfänger und schließlich als Datenframe versendet; ein Datenframe wiederum wird als Bitfolge übertragen.

Auf der untersten Schicht – sie wird Schicht 1 oder Bitübertragungsschicht genannt – geht es darum, wie ein Computer Bits über einen Kanal schicken kann. Hier ist noch nicht von Adressen die

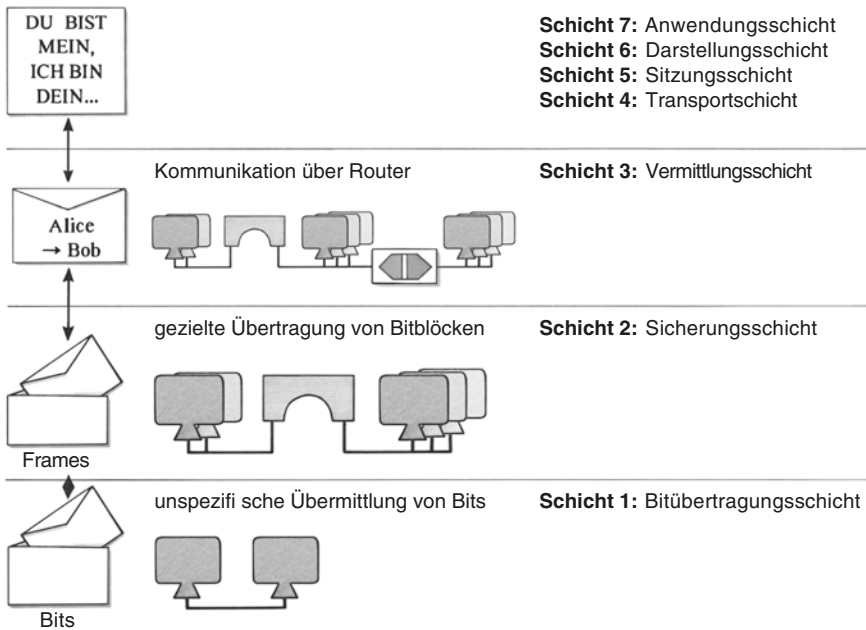


Abb. 6.15 Das Schichtenmodell eines Computernetzwerks. (© Susanne Staubli, Eckart Zitzler)

Rede, nur von dem Übertragungskanal, an den ein Netzwerkadapter angeschlossen ist. Die nächste Schicht – Schicht 2 oder Sicherungsschicht – führt die MAC-Adressen ein und die gezielte, fehlerfreie Kommunikation auf einem gemeinsamen Übertragungssystem, beim Ethernet eine gemeinsame Leitung. Hier ist definiert, wie ein Frame aussieht und was bei Konflikten passiert, wenn mehrere Teilnehmer gleichzeitig kommunizieren. Bridges sind auf Schicht 2 angesiedelt. Und die dritte Schicht, die Vermittlungsschicht, ist diejenige der Router. Sie führt die IP-Adressen ein, regelt die Wegleitung über Zwischennetze und erlaubt es, von einem Computer ein Datenpaket an einen beliebigen anderen Computer im Internet zu schicken. Diese Schichten sind Ihnen bereits vertraut. Es gibt aber noch weitere Ebenen. Schicht 4 beispielsweise, die Transportschicht, ermöglicht es, wie bei einem Telefonanruf eine stehende Verbindung zwischen zwei Computern zu etablieren. So lässt sich ein kontinuierlicher Datenverkehr – denken Sie nur an eine Videoübertragung – sicherstellen (siehe Bsp. 6.7).

Beispiel 6.7: Das Internet- und das Transport-Protokoll

Das Transport-Protokoll – im Englischen TCP für *Transmission Control Protocol* – baut auf dem vorgestellten Internet-Protokoll auf und ergänzt es um weitere Möglichkeiten. Das Internet-Protokoll erlaubt ja, von einem Computer ein Datenpaket an einen anderen zu schicken, doch das Problem, wie man einen kontinuierlichen Datenstrom übermittelt, der in mehrere Pakete unterteilt werden muss, löst es nicht. Was ist z. B., wenn die Pakete in unterschiedlicher Reihenfolge ankommen? Das kann ja durchaus vorkommen, weil die Pakete verschiedene Wege nehmen können. TCP löst dieses Problem, indem es Verbindungen zwischen zwei Rechnern ermöglicht, über die Daten beliebiger Größe in beide Richtungen versendet werden können. Und es achtet dabei nicht nur auf die Reihenfolge der Pakete, sondern fordert solche, die verloren gehen, erneut an. Und es schaut, dass das Netz nicht überlastet wird. Viele Dienste, die wir uns im nächsten Themenblock anschauen werden, z. B. ein Dateitransfer oder das World Wide Web, nutzen das TCP. TCP wurde übrigens Anfang der 1970er-Jahre von Robert E. Kahn und Vinton G. Cerf entwickelt (Abb. 6.16).

Im Folgenden werden wir uns den obersten Schichten zuwenden und die Nutzung des Internets in den Vordergrund stellen. Dabei setzen wir eine Infrastruktur voraus, die die Schichten 1 bis 4 realisiert.



Abb. 6.16 Vint Cerf und Bob Kahn: zwei Internetpioniere, die an der Entwicklung von Internet-Protokoll und Transport-Protokoll maßgeblich beteiligt waren. (© Camille Büsser)

Alles wächst zusammen

Das Kommunikationsgerüst, welches das Internet zur Verfügung stellt, eröffnet weit mehr Möglichkeiten als den reinen Datenaustausch. Es stellt vielmehr den Kitt dar, mit dem aus einer verbundenen Ansammlung von einzelnen Computern ein Ganzes wird. Als die Menschen Metropolen bildeten, war es ähnlich. Weil so viele Individuen miteinander agieren konnten, bildete sich eine gemeinsame Kultur mit ihren eigenen Regeln und aus einer reinen Menschenansammlung formte sich eine übergeordnete Gesellschaft.

Die vermutlich erste Großstadt war Uruk in Mesopotamien, im heutigen Irak. Sie beherbergte auf einer Fläche von ca. 2,5 km² ungefähr 25.000 Einwohner, eine für damalige Verhältnisse enorme Größe. Und weil sich hier so viele Menschen auf engem Raum niederließen, erforderte und ermöglichte dies neuartige Formen des Zusammenlebens und der Organisation. Tatsächlich war Uruk eine Brutstätte für neue Ideen und ein Experimentierfeld. Verschiedene Errungenschaften des Gesellschaftslebens, die maßgeblich für unsere heutigen Zivilisationen sind, wurden dort erprobt. Im Mittelpunkt stand dabei die Arbeitsteilung in verschiedenster Ausprägung (Abb. 6.17). Es bildeten sich Spezialisten für bestimmte Aufgaben heraus, Bauern, Handwerker usw. Es gab eine Verwaltung, die das komplexe Gesellschaftsleben regulierte und steuerte. Und durch Standardisierung und Strukturierung wurden nicht nur Arbeitsabläufe optimiert, sondern neuartige Möglichkeiten geschaffen, Informationen festzuhalten und auszutauschen, z. B. mittels Schriftzeichen.

Natürlich gibt es ganz unterschiedliche Organisationsformen, wie eine Menge von Einzelementen kooperieren kann – das gilt sowohl für menschliche Gemeinschaften als auch für Computernetzwerke. Sie hängt von der zu lösenden Aufgabe ab, aber auch von der Größe der Gemeinschaft.



Abb. 6.17 Spezialisierung und Arbeitsteilung im Alltag. (© Thomas Aumann/[stock.adobe.com](https://www.stock.adobe.com) (links), majorosl/Getty Images/iStock (rechts))

Je größer die Gesellschaft, desto komplexer die Organisationsform und desto höher der Verwaltungsaufwand, der erforderlich ist, um die einzelnen Individuen zu koordinieren. In kleinen Gemeinschaften kann Arbeitsteilung bedeuten, dass alle alles machen und jeder seinen Teil beiträgt. In größeren Gemeinschaften hat sich die Spezialisierung durchgesetzt, und das in großem Stil: Einige konzentrieren sich auf ausgewählte Aspekte des Gesellschaftslebens und erbringen Dienste für alle anderen. Im Internet lassen sich beide Modelle wiederfinden: Das erste heißt Peer-to-Peer, das zweite Client-Server.

Von Kunden und Dienstleistern

Das Client-Server-Modell stand am Anfang der Computernetzwerke, und dabei ging es primär um die gemeinsame Nutzung von knappen Ressourcen. Die ersten Großrechner mussten sich viele teilen, schließlich konnten sich nur große Institutionen diese Kolosse leisten. Die Benutzerinnen erhielten über Terminals, also Bildschirmenheiten mit Tastatur, Zugang zu der Ressource Computer. Das waren quasi die Vorläufer des Netzwerks. Später, als Computer immer kleiner und erschwinglicher wurden, hielten sie auch in den Büros Einzug, und so entstand der Bedarf nach einer gemeinsamen Infrastruktur. Geräte wie etwa Drucker waren sehr teuer und so war es wichtig, dass diese von vielen Computern genutzt werden konnte. Gleichzeitig war es sehr umständlich, kollektiv an etwas zu arbeiten: Eine zentrale Datenablage gab es nicht. Und schließlich wurde der Aufwand, alle Computer in einem Bürogebäude zu warten, immer größer. Es bedurfte schon bald geeigneter Lösungen, damit ein Administrator nicht von Computer zu Computer laufen musste, sondern viele seiner Aufgaben über das Netzwerk von einem Arbeitsplatz aus ausführen konnte.

Die neuen Anforderungen führten schließlich zur Idee, einzelne, gut ausgestattete Computer als Dienstleister einzusetzen, während andere Computer die Dienste als Kunden in Anspruch nehmen. Erstere werden Server genannt, letztere Clients. Beispielsweise kann ein Drucker an einen Server angeschlossen sein und alle Computer können Dokumente drucken, indem sie sie an den Server schicken. Natürlich braucht es auch wieder spezielle Protokolle, die die Kommunikation zwischen Client und Server regeln: Wie stellt Ersterer Anfragen, wie antwortet Letzterer, welche Informationen werden in welchem Format zur Verfügung gestellt usw.? Für den Transfer von Dateien wurden z. B. das *File Transfer Protocol* (FTP) entwickelt, auf dessen Basis sich eine gemeinsame Dateiablage realisieren lässt (siehe Bsp. 6.8).

Beispiel 6.8: Die Zentralbibliothek in einem Computernetzwerk

Ein Dateiserver ist die neuzeitliche Variante einer Bücherbibliothek: Auf einem Computer werden gemeinsame Dateien gespeichert, auf die all die anderen Computer im Netzwerk zugreifen können (siehe Abb. 6.18). Er realisiert für alle oder ausgewählte Computer in einem Netzwerk eine gemeinsame Dateiablage und ist mit diversen Massenspeichern ausgestattet, auf die der Datenbestand für alle Computer im Netzwerk gespeichert wird. Speichert also Carol eine Bilddatei auf dem Dateiserver ab, dann kann Bob später darauf zugreifen und sie vom Dateiserver herunterladen. Natürlich können einzelne Computer auch noch Daten lokal speichern, auf die können die anderen jedoch nicht zugreifen.

Es gibt unterschiedlichste Dienste, die in einem Computernetzwerk zur Verfügung gestellt werden können, das Teilen von Druckern oder Massenspeichern illustriert nur das Prinzip. Auch für den Fernzugriff auf andere Computer gibt es spezielle Protokolle und dazugehörige Software: Anfangs war es Telnet, später wurde daraus SSH (*Secure Shell*). Telnet und SSH ermöglichen es, sich von einem Computer über das Netzwerk in einen anderen einzuloggen und dort Wartungsarbeiten vorzunehmen. Die Übersetzung von symbolischen in numerische Internetadressen wird ebenfalls von Servern übernommen, den erwähnten DNS-Servern; diese tauschen sich sogar untereinander aus, um die Dualzahl-Adressen zu unbekannten Domain-Namen zu ermitteln. Häufig ist es auch so, dass ein Server mehrere Dienste anbietet, und aus diesem Grund hat man Möglichkeiten geschaffen, spezifizieren zu können, welchen Dienst man meint. Dafür gibt es sogenannte Ports (siehe Anm. 6.3). Sie sind nichts anderes als Kommunikationskanäle, über die die ankommenden Pakete sortiert und

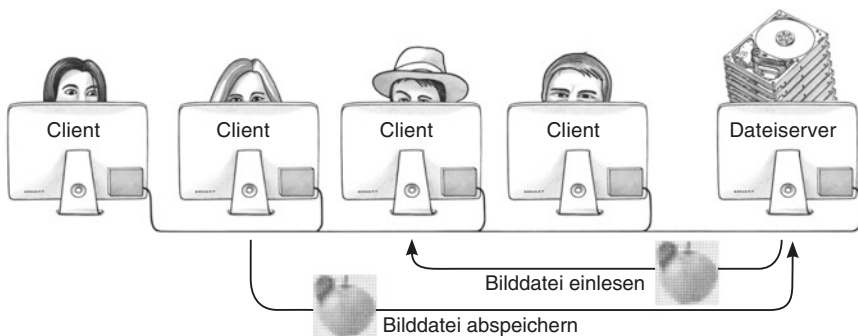


Abb. 6.18 Prinzip eines Dateiservers. (© Susanne Staubli, Eckart Zitzler)

den entsprechenden Kommunikationsprogrammen zugeordnet werden können. Natürlich muss ein Server, der gleich mehrere Aufgaben wahrnimmt, entsprechend leistungsstark sein, um große Mengen von Anfragen bewältigen zu können.

Anmerkung 6.3: Ports und Häfen für den Datenverkehr

Ein Port ist vergleichbar mit einem Briefkasten in einem Mehrfamilienhaus. Er besitzt eine eindeutige Nummer, über die ankommende Datenpakete kanalisiert und dem entsprechenden Empfänger zugeordnet werden können. Jedem Server-Programm, das auf einem Computer läuft, wird ein fixer Port zugewiesen. Es handelt sich hierbei um eine Zahl zwischen 1 und 65536 (richtig geraten, eine Portnummer ist eine 16-Bit-Dualzahl). Alle Pakete, die an den spezifischen Port geschickt werden, werden an das Server-Programm vermittelt, das dann die weitere Verarbeitung übernimmt. Und auch auf der Client-Seite gibt es einen Port, an den das Server-Programm seine Antwort schickt – woher wüsste der Client-Computer sonst, für welches Client-Programm die Server-Antwort gedacht ist? Für viele Protokolle ist ein Port weltweit festgelegt, beispielsweise arbeitet Telnet über Port 21, SSH über Port 22 und DNS über Port 53. Ein bestimmter Bereich ist aber auch frei, d. h., er kann vom Betriebssystem eines Computer an die laufenden Prozesse dynamisch vergeben werden.

Es gibt also einerseits einen Computer, der als Server fungiert und Dienste anbietet, andererseits können auf dem Computer mehrere Server-Programme laufen, die unterschiedliche Dienste implementieren und über Ports ansprechbar sind. Deswegen muss man, um ganz genau zu sein, sagen: Ein Server ist eine Kombination aus einem Server-Programm und dem Computer, auf dem dieses Programm abläuft – das Gleiche gilt für den Client. Manchmal wird auch nur der Computer oder nur das Programm als Server bzw. Client bezeichnet, aber streng genommen handelt es sich um diese Kombination, denn ein Server-Computer kann ja auch ein Client-Programm abarbeiten. Zerbrechen wir uns aber nicht den Kopf, das grundlegende Prinzip ist so wie in Abb. 6.19 links dargestellt. Nebenbei bemerkt: Dienste können auch auf mehrere Computer verteilt sein, das ist z. B. beim E-Mail-System der Fall: Da gibt es einen Server zum Empfang von E-Mails, quasi der Briefkasten zuhause, und einen Server zum Versenden von E-Mails, quasi die Poststelle. Ein anderes Beispiel ist das World Wide Web.

Das World Wide Web, abgekürzt als WWW oder Web, ist ursprünglich eine über das Internet verteilte Sammlung von Dokumenten, den sogenannten Webseiten. Die Dokumente enthalten Verweise auf andere Dokumente; die Verweise werden nach dem englischen Ausdruck als Links bezeichnet. Das Web – häufig und fälschlicherweise mit dem Internet gleichgesetzt – ist ein System, das aus Webservern (Computer, auf denen die Dokumente

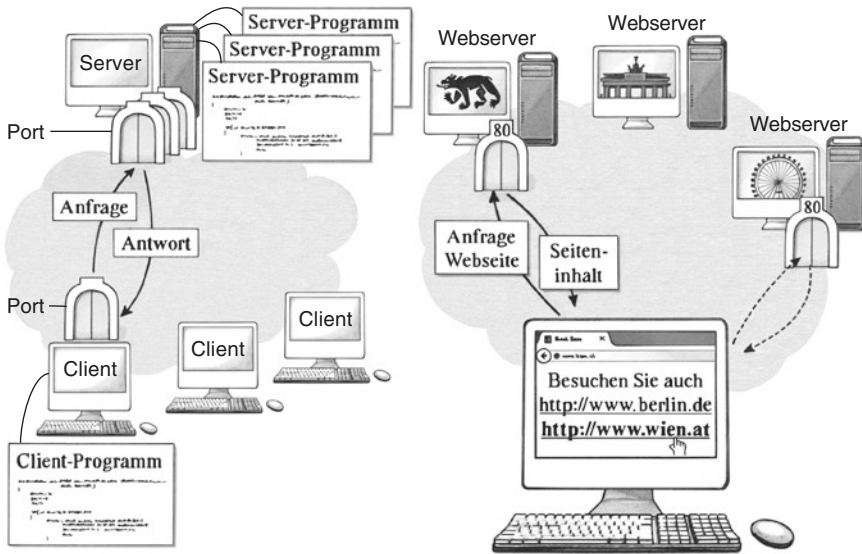


Abb. 6.19 Das Client-Server-Modell (links) und eine Implementierung des Prinzips am Beispiel des World Wide Web (rechts). (© Susanne Staubli, Eckart Zitzler)

abgelegt sind) und Webbrowsern (Client-Programme, die Dokumente von Webservern anfordern und auf dem eigenen Computer darstellen) besteht. Auf der rechten Seite in Abb. 6.19 ist diese Situation exemplarisch visualisiert. Anfangs waren es nur Textdokumente, die angezeigt werden konnten, später wurden es Multimedia-Inhalte und noch später kam die Möglichkeit hinzu, Programme in Webseiten einzubetten, um so z. B. auf Benutzereingaben reagieren zu können. Die Hintergründe zum Web erläutert Bsp. 6.9.

Beispiel 6.9: Das World Wide Web

Die Grundlagen für das World Wide Web wurden maßgeblich von Tim Berners-Lee (Abb. 6.20) am europäischen Forschungszentrum CERN entwickelt. Sie umfassen ein Dokumentenformat, das die Darstellung von Webseiten mittels reinem Text erlaubt. Dieses Webseiten-Textformat heißt HTML, was für *Hypertext Markup Language* steht; Hypertext werden verlinkte elektronische Textdokumente genannt. Daneben gibt es ein Protokoll, das die Kommunikation zwischen Webbrowsern und Webservern regelt und die Übertragung von Webseiten realisiert. Das ursprüngliche Protokoll, das *Hypertext Transfer Protocol* oder kurz HTTP, wurde später ergänzt um eine Erweiterung, die die verschlüsselte Übertragung ermöglicht: HTTPS, wobei das S für *secure* steht. Und schließlich bedarf es eines Adressierungssystems, um



Abb. 6.20 Tim Berners-Lee gilt als Erfinder des World Wide Web. (© Camille Büsser)

Webseiten kennzeichnen und lokalisieren zu können. Hier hat man kurzerhand die symbolische Internetadresse des Servers genommen und dann einfach den Pfad des internen Dateisystems (mit Dateinamen) angehängt.

Um eine bestimmte Webseite anzuschauen, müssen wir ihre Adresse in den Browser eingeben. Der Webbrowser löst erst den symbolischen Namen auf, schickt auf Port 80 (im Fall von HTTP) bzw. Port 443 (im Fall von HTTPS) des entsprechenden Webserver die Anfrage mit der Webadresse, und dieser liefert daraufhin die Textdatei, die die Webseite beschreibt. Der Webbrowser bereitet die Seite grafisch auf und stellt sie dar. Wählen wir nun einen Link an, dann ist dahinter wiederum eine Webadresse verborgen. Der Webbrowser macht nichts anderes, als den dazugehörigen Webserver zu kontaktieren, die Webseite anzufordern und die neue Seite darzustellen.

Im World Wide Web erbringen viele Server im Kollektiv einen Dienst. Die Webseiten sind quer verteilt über den Globus auf verschiedensten Computern gespeichert, obwohl theoretisch(!) alle Dokumente auch auf einem Server gespeichert werden könnten. Die dezentrale Organisation hat enorme Vorteile, vor allem ermöglicht sie immense Speicherkapazitäten. Diese Überlegungen leiten über zum zweiten Modell der Arbeitsteilung, dem Peer-to-Peer-Prinzip.

Einer für alle, alle für einen

Neben der Client-Server-Architektur etablierte sich mit der Zeit auch ein anderer Ansatz, bei dem die Unterscheidung in Kunden und Dienstleister nicht mehr so strikt war. Im Peer-to-Peer-Modell – Peer



Abb. 6.21 Dass Peer-to-Peer-Prinzip am Beispiel eines Chats. (© Susanne Staubli, Eckart Zitzler)

steht für seinesgleichen – liegt die Betonung auf dem Gemeinsamen und dem Gleichberechtigten: Eine Gruppe erbringt zusammen einen Dienst für Individuen. Ein einfaches Beispiel für ein solches Szenario ist in Abb. 6.21 angedeutet: ein Chat zweier Personen. Auf den beiden Computern läuft jeweils ein Chat-Programm, und jedes davon kann den Datentransfer zur Gegenseite initiieren.

Sein volles Potenzial entfaltet das Peer-to-Peer-Konzept jedoch erst, wenn viele Computer involviert sind (siehe Abb. 6.22). Zum ersten Mal einer breiten Öffentlichkeit bekannt wurde es durch Napster, einer Musiktaschbörse, die 1999 online ging, innerhalb kürzester Zeit einen enormen Mitgliederzulauf verzeichnete und zwei Jahre später aufgrund juristischer Auflagen (Copyright-Verletzungen) in dieser Form geschlossen werden musste (Bsp. 6.10). Es handelte sich dabei wie beim World Wide Web um eine verteilte Datenablage, nur ging es nicht um Hypertext-Dokumente, sondern um Audio-Dateien. Jeder Peer stellte seine Musiksammlung den anderen Peers zur Verfügung und konnte dafür Musik von allen anderen beziehen. Damit fungierte jeder in die Gruppe eingebundene Computer gleichzeitig als Server und Client. Die Inhalte wurden über einen Super-Peer (an sich ein Server) katalogisiert, der somit wusste, wo welche Dateien gespeichert waren. Der Super-Peer nahm Musikanfragen entgegen und vermittelte dann zu einem entsprechenden Peer. So funktionieren beispielsweise auch Dienste, die Video-Anrufe über das Internet ermöglichen. Hier authentifiziert man sich über einen dezentralen Server, d. h., man meldet sich an und erhält die Kontaktdaten der anderen; die Kommunikation selber findet aber ggf. nur zwischen den beiden Kommunikationspartnern direkt statt.

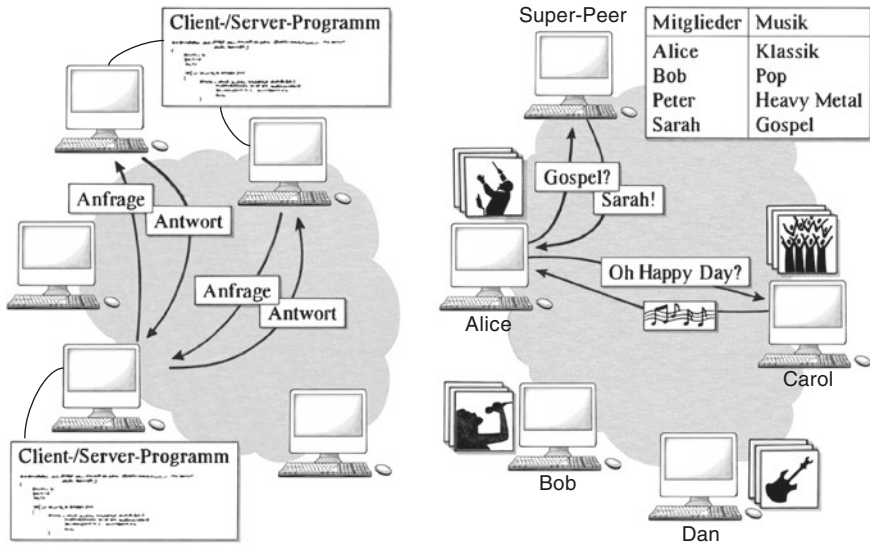


Abb. 6.22 Das Peer-to-Peer-Prinzip in der Theorie (links) und in der Praxis als Musikaustauschbörse (rechts). (© Susanne Staubli, Eckart Zitzler)

Beispiel 6.10: Die Musikaustauschbörse Napster

Die Architektur des ursprünglichen Napster-Systems war wie folgt: Auf jedem Peer-Computer lief ein Programm, das lokal die Anbindung an die gesamte Napster-Gemeinschaft ermöglichte. Des Weiteren gab es einen (an sich mehrere, aber das spielt hinsichtlich des Konzepts keine Rolle) Server, den Super-Peer, der eine Liste mit allen angeschlossenen Computern und deren zur Verfügung gestellten Songs führte und aktuell hielt.

Wenn nun – wie in Abb. 6.22 rechts skizziert – eine Benutzerin einen bestimmten Song herunterladen wollte, dann kontaktierte ihr Napster-Programm den Napster-Server, der daraufhin in seiner Liste nachschaute. Wenn der Server eine Teilnehmerin fand, die diesen Song vorrätig hatte, dann lieferte er dem anfragenden Napster-Programm die Koordinaten des entsprechenden anderen Computers. Daraufhin nahm das Napster-Programm mit dem Napster-Programm des anderen Computers Kontakt auf und die Datei wurde per FTP übertragen – das setzte natürlich voraus, dass beide Computer online und erreichbar waren.

Der Erfolg von Napster – kurz vor der Abschaltung umfasste die Sammlung mehr als 2 Mrd. Dateien – legte eigentlich den Grundstein für die kommerzielle Verbreitung von Musik über das Internet, womit gleichzeitig das schnelle Ende der CD eingeleitet wurde.

Die verteilte Dateiablage, bei der sich eine große Menge von Nutzern Daten gegenseitig zur Verfügung stellen, ist die klassische Anwendung für ein Peer-to-Peer-Modell. Anstatt Speicherplatz lässt sich allerdings auch Rechenpower verbinden. So ersann man Erweiterungen von Betriebssystemen, die erlauben, Unterprogramme auf entfernten Rechnern auszuführen – *remote procedure call* (RPC) genannt. Der einzige Unterschied zum gewöhnlichen Unterprogrammaufruf ist, dass die Übergabeparameter und die Rückgabewerte als Pakete über das Netz geschickt werden. So lassen sich – wie bei einem Mehrprozessorsystem – Rechenaufgaben auf viele Computer verteilen und aufwändige Problemstellungen wie das Knacken von kryptografischen Codes oder die Vorhersage von Proteinfaltungen vereint lösen. Es gibt noch unzählige weitere Anwendungsgebiete; interessant ist z. B. das Zahlungssystem Bitcoin, bei dem Geldtransfers im Kollektiv genehmigt werden (siehe Anm. 6.4).

Anmerkung 6.4: Ein Geschichtsbuch für Geldflüsse

Bitcoin steht für ein virtuelles Zahlungssystem, das 2008 erstmalig beschrieben und 2009 praktisch in Betrieb genommen wurde, und bezeichnet gleichzeitig die verwendete Geldeinheit. Es basiert auf einer dezentralen Datenbank, die über ein Peer-to-Peer-Netzwerk gespeichert und gepflegt wird. Diese Datenbank führt Buch über alle Vorgänge, bei denen Geld seinen Besitzer wechselt; Sie können sie sich als riesigen Kontoauszug vorstellen, in dem alle Überweisungen, die je im System vorgenommen wurden, aufgelistet sind. Die Datenbasis selbst ist als sogenannte Blockchain organisiert, wobei zeitgleiche Vorgänge in einem Datensatz – einem Block – gebündelt werden und die Datensätze hintereinander – als Kette – in zeitlicher Reihenfolge angeordnet sind (siehe Abb. 6.23). Dabei sind die Datensätze so miteinander verwoben, dass eine nachträgliche Manipulation im Wesentlichen ausgeschlossen ist. Denn jeder Datensatz erhält eine Prüfsumme, analog zur Parität (vgl. Kap. 4) oder zu einer Hash-Funktion (vgl. Kap. 5). Diese Prüfsumme ist im nachfolgenden Datensatz gespeichert und beeinflusst wiederum dessen Prüfsumme. Die Prüfsummenfunktion ist so gewählt, dass jede Änderung in einem Datensatz auch eine Änderung der Prüfsumme zur Folge hat und es nahezu unmöglich ist, einen anderen Datensatz mit der gleichen Prüfsumme zu finden.

Die Datenbasis ist auf allen Peer-Computern vollständig gespeichert, also vielfach repliziert. Das Besondere ist nun, wie Daten zur Datenbank hinzugefügt werden, also Überweisungen vorgenommen werden – und zwar ohne dass ein Betrüger sein Geld einfach mehrfach verwenden kann. Eine Überweisung wird von dem Teilnehmer, der sie vornehmen möchte, an alle Peers im Netzwerk verschickt, quasi angekündigt. Jetzt versuchen – vereinfacht gesagt – alle Mitglieder des Peer-to-Peer-Netzwerks, diese Überweisung an die Blockchain anzuhängen. Dazu muss ein Block generiert werden, der die Prüfsumme des Vorgängers, die Überweisungsdaten selbst sowie eine beliebige Zahl enthält, die – und das ist das Schwierige – eine Prüfsumme besitzt, die als Dualzahl dargestellt mit einer fixen Anzahl von Nullen beginnt. Dafür muss ein Peer die richtige Zahl finden, und das ist eine knifflige Aufgabe, die Rechenzeit

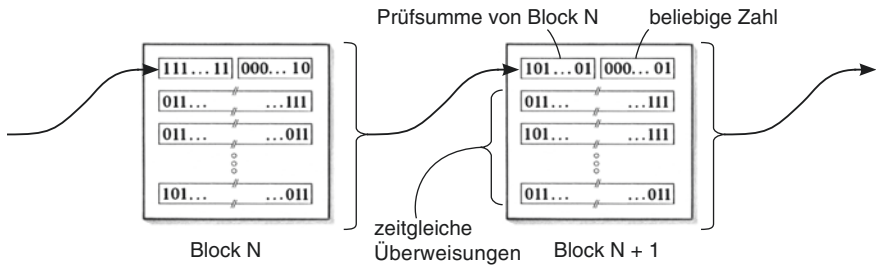


Abb. 6.23 Das Konzept der Bitcoin-Blockchain. (© Susanne Staubli, Eckart Zitzler)

beansprucht. Der Peer, der am schnellsten ist, wird monetär belohnt und schickt seinen Block an alle Peers. Die akzeptieren ihn, wenn er korrekt ist und das Geld nicht bereits anderweitig verwendet wurde. Es kann zwar passieren, dass mehrere Peers gleichzeitig einen Block generieren und die Kette eine Zeit lang aufgesplittet ist. Doch letztlich wird sich irgendwann einer der Zweige durchsetzen, d. h. von der Mehrheit der Peers akzeptiert werden, und der andere Zweig wird dann nicht mehr weiterverfolgt.

Beide Modelle der Arbeitsteilung – Client-Server und Peer-to-Peer – können in Reinkultur vorkommen, häufiger sind aber Mischformen, wie wir es z. B. bei Napster gesehen haben. Noch vielfältiger und vielschichtiger wird das Ganze, wenn solche Dienste hierarchisch aufeinander aufbauen oder beliebig miteinander vernetzt sind. Es entstehen Gebilde, die sehr an das komplexe Gefüge in modernen Gesellschaften erinnern.

Der Weltcomputer

Das Internet als Ganzes hat ein enormes Wachstum erlebt. Waren es Anfang der 1980er-Jahre noch ein paar hundert Computer, so umfasste das Netz gut 30 Jahre später bereits eine Milliarde Knoten. Gleichzeitig ist die Bandbreite – das Datenvolumen, das in einer bestimmten Zeiteinheit übermittelt werden kann – massiv gestiegen. Somit übernahm das Netzwerk, an das ein Computer angeschlossen war, quasi die Funktion eines Busses, über den periphere Komponenten angekoppelt wurden: entfernte Massenspeicher, entfernte Ein- und Ausgabegeräte, entfernte Zusatzprozessoren usw. Das Kommunikationsgerüst selbst trat dabei immer mehr in den Hintergrund und seine Struktur verschwamm. Denn während zu Anfang des Internets noch die einzelnen Komponenten, d. h. Rechner und sonstige Hardware, klar erkennbar waren, wurde das Internet über die Zeit zu einer undurchsichtigen

Wolke, der Cloud. Das riesige Konglomerat von Computern trat auf einmal als Gesamtsystem in Erscheinung und wurde so zum Weltcomputer.

Dabei wurden die Ideen, die wir im vorherigen Abschnitt kennengelernt haben, konsequent weiterentwickelt. Anstatt dass ein Server einen Dienst erbringt, kann es nun ein Teilnetzwerk von Computern sein, die zusammen den Dienst realisieren – und evtl. greifen sie selbst wiederum auf andere Dienstleister zurück (siehe Abb. 6.24). Dienste können, wie vorher schon angedeutet, Speicherplatz oder Rechenleistungen anbieten, doch mit der Zeit sind sie immer mehr zu eigenen Anwendungen geworden, so als würden wir ein Programm auf unserem eigenen Computer ablaufen lassen. Diese Anwendungen werden vielfach über einen Webbrowser bedient; wie und wo sie realisiert werden, ist dabei für die Benutzerin nicht ersichtlich. Immer mächtigere, komplexere und aufwändigere Dienste entstehen dabei, und sie lassen sich sogar miteinander kombinieren, quasi zu einem übergeordneten Programm kombinieren. Unser Computer zu Hause ist das Terminal, mit dem wir zum erwähnten Weltcomputer Zugang haben. Das Web, ursprünglich als verteilte Dokumentenablage gedacht, hat sich längst gewandelt zu einer generellen Benutzungsoberfläche, mit der wir z. B. beim Online-Banking die Computer unserer Bank ansteuern können. Und mittlerweile ist eine weitere Entwicklung zu beobachten, nämlich dass nicht nur Computer, sondern beliebige Geräte in das Internet eingebunden werden (siehe Anm. 6.5).

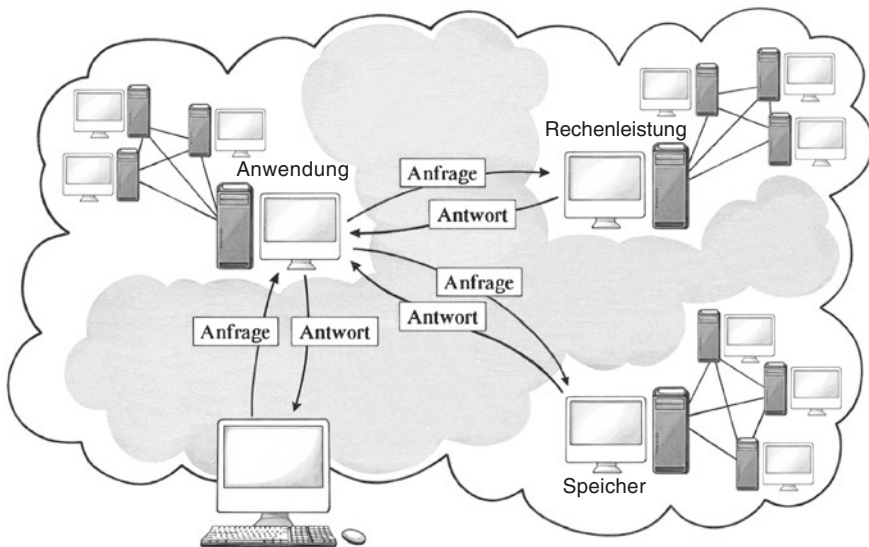


Abb. 6.24 Das Internet als Wolke, deren innere Struktur für den Nutzer verborgen ist. (© Susanne Staubli, Eckart Zitzler)

Anmerkung 6.5: Sensoren und Aktoren weltweit

Wenn beliebige Geräte zu Endbenutzern im Internet werden können, geschieht etwas, was wir schon im Zusammenhang mit dem Verwendungszweck von Computern diskutiert hatten. Der Weltcomputer wird zu einem riesigen eingebetteten System, dessen Nervensystem das Internet ist. Dabei können verschiedenste technische Systeme autonom oder teilautonom eingebunden sein, miteinander in Verbindung treten und konzentriert Aufgaben bearbeiten (siehe Abb. 6.25). Alltagsgeräte, die ursprünglich ohne Computertechnik auskamen, wie z. B. die Beleuchtung zu Hause, der Kühlschrank oder die Waschmaschine, sind zu Teilen des Internets geworden und darüber miteinander verbunden. So kann das Auto mit der Heizungsanlage zu Hause kommunizieren, damit es bei Ankunft wohlig warm im Wohnzimmer ist.

Aus dem Internet ist also mittlerweile eine Art Nervensystem der technischen Welt geworden, und unzählige Vorgänge in unserer Gesellschaft laufen über dieses Nervensystem. Wir können die gesellschaftlichen und ethischen Fragen hier nicht diskutieren und halten nur fest, dass die Technologie des Internets – genauso wie die Schrift, der Telegraf, das Telefon, das Fernsehen usw. – ein weiteren Meilenstein in der Entwicklung

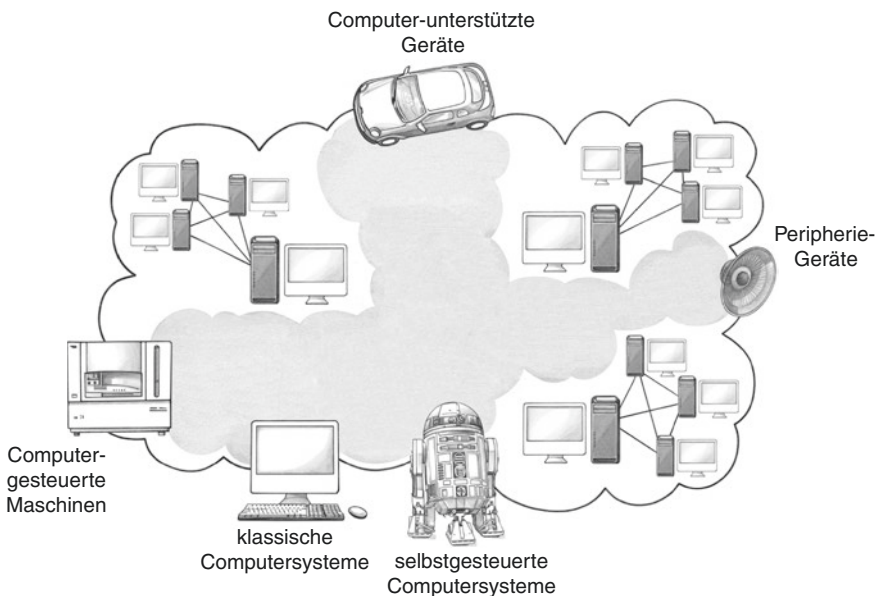


Abb. 6.25 Das Internet als Nervensystem des Weltcomputers. (© Susanne Staubli, Eckart Zitzler)

der menschlichen Gesellschaft markiert: Die Welt wird zum Dorf. Ohne Computernetzwerke wäre das alles undenkbar.

Zwischennetze in der Natur

Kommen wir am Ende dieses Kapitels noch auf die natürlichen Kommunikationsnetze zu sprechen. Systeme zum Austausch von Informationen sind essenziell für das Leben, sowohl für individuelle Organismen als auch für Gemeinschaften von Organismen. Wir werden uns für beide Fälle im Folgenden ein Beispiel herauspicken: den Menschen und den Ameisenstaat.

Voller Netze: der Mensch

Wir Menschen tragen ebenfalls verschiedenste Kommunikationsgeflechte in uns, schließlich muss unser Körper unsere Bestandteile genauestens koordinieren. Ihr Ursprung liegt weit zurück: Als sich aus Einzellern vielzellige Organismen mit spezialisierten Organen bildeten, mussten gleichzeitig geeignete Kommunikationswege geschaffen werden, um die unzähligen einzelnen Zellen, die verschiedenen Gewebe und Organe miteinander abzustimmen und zu steuern. Im Laufe unserer Evolution sind dabei zwei Kommunikationsnetze entstanden, das Nervensystem und das Hormonsystem. Bei Ersterem werden Signale über elektrische Impulse repräsentiert, die schnelle Informationsübertragung steht im Vordergrund. Bei Letzterem sind es Moleküle, die Nachrichten überbringen; entsprechend ist das Hormonsystem auf eine langsamere Wirkweise eingestellt und auf langfristige Prozesse ausgerichtet.

Das Hormonsystem reguliert verschiedene Vorgänge in unserem Körper, so z. B. Wachstum, Fortpflanzung und Ernährung (Abb. 6.26). Hormone – ihrer griechischen Wortwurzel gemäß stellen sie Antreiber dar – sind Moleküle, die zumeist in speziellen Drüsen, u. a. der Schilddrüse und der Hirnanhangsdrüse, gebildet und über die Blutbahn zu den Zielorten transportiert werden. Die Blutbahn dient quasi als Leitung für alle, die an die Organe angeschlossen sind. Wie beim Ethernet: Alle hören und bekommen mit, was für Botschaften da übermittelt werden. Doch Hormone sind in der Regel spezifisch, d. h., sie sprechen bestimmte Organe und Gewebe an, genauso wie die Frames ja auch nur für gewisse Kommunikationspartner gedacht sind. Das Adresssystem des Hormonsystems funktioniert nach dem



Abb. 6.26 Vorgänge, die durch das Hormonsystem reguliert werden: Zucker- und Fettstoffwechsel, Knochenwachstum sowie Fortpflanzung. (© Judith Büsser)

Schlüssel-Schloss-Prinzip (vgl. Kap. 7): Die Beschaffenheit des Botenstoffs – u. a. seine Form und die chemische Zusammensetzung – hat zum Zweck, dass die Zielorgane mittels spezieller, jeweils passender Rezeptoren das durch das Hormon übertragene Signal erkennen und anschließend darauf reagieren können. Bei uns Menschen gibt es in etwa 50 verschiedene Hormone.

Demgegenüber ist das Nervensystem eher mit einem Routernetzwerk vergleichbar, das über Direktverbindungen funktioniert. Denn Nervenfasern sind immer Punkt-zu-Punkt-Verbindungen. Leitungen werden im Nervensystem nur insofern mehrfach benutzt, als dass ein Axon mit mehreren Dendriten anderer Nervenzellen verbunden sein kann; doch bei solchen Verbindungen ist die Kommunikation immer ein Broadcast, d. h., alle angeschlossenen Empfänger nehmen das Signal entgegen. Die Organe und Gewebe werden von Nervenfasern direkt angesprochen, die Adressen sind implizit über die Verkabelung gegeben, es gibt nicht wie beim Hormonsystem spezifische Adressen und Unterscheidungen bei den Signalen.

Hormon- und Nervensystem sind eng miteinander gekoppelt, und das hat seinen Grund. Die Kommunikation über Hormone ist nämlich äußerst sensitiv. Nur geringere Änderungen an den Konzentrationen einzelner Hormone im Blut können enorme Auswirkungen haben. Daher muss die Hormonausschüttung sehr genau gesteuert werden – und zwar über das Gehirn. Viele Informationen über das innere Milieu und die körperliche Befindlichkeit laufen im Hypothalamus zusammen, einem Teil des Zwischenhirns, das die vegetativen Funktionen des Körpers kontrolliert. Von dort aus werden Hormone ausgesandt an eine andere Region im Gehirn, die Hirnanhangsdrüse, welche daraufhin wiederum über Hormone spezifische Drüsen im Körper anweist, wie hoch die Ausschüttung zielgerichteter Hormone sein soll. Manchmal werden jedoch hier Ebenen

übersprungen. Das lässt sich beispielsweise beim Stressverhalten sehen: Über das Hormonsystem wird der Körper vorbereitet, während die eigentliche Reaktion – sei es nun Flucht oder Angriff – dann letztlich über das Nervensystem gesteuert wird (siehe Bsp. 6.11).

Beispiel 6.11: Was bei Stress passiert

In Abb. 6.27 ist der Ablauf einer Stressreaktion beispielhaft dargestellt. Wann immer wir in Gefahr sind und diese über das Nervensystem wahrnehmen, wird über das Hormonsystem eine Vielzahl körperlicher Veränderungen eingeleitet, die uns auf eine Flucht oder anderweitige Reaktion vorbereiten sollen. Ausgehend vom Gehirn wird über das Nervensystem das Nebennierenmark angewiesen, die Hormone Adrenalin und Noradrenalin in einem bestimmten Verhältnis zueinander zu produzieren. Des Weiteren veranlasst der Hypothalamus über Hormone und den Zwischenweg der Hirnanhangsdrüse die Nebennierenrinde dazu, Glukokortikoide herzustellen. Das führt innerhalb kürzester Zeit zu diversen körperlichen Veränderungen: Herzfrequenz und -kontraktionskraft nehmen zu, die Durchblutung gewisser innerer Organe und der Haut wird gedrosselt, während Herz und Lunge besser durchblutet werden und die Bronchien sich weiten.

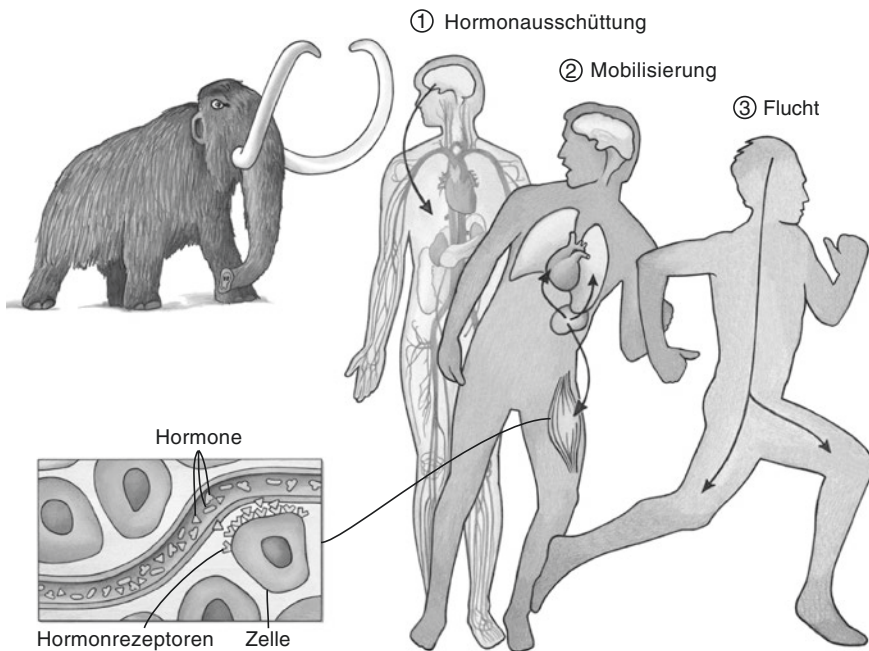


Abb. 6.27 Das Zusammenspiel von Hormon- und Nervensystem bei einer Stressreaktion. (© Susanne Staubli, Eckart Zitzler)

Interessant ist in diesem Zusammenhang, dass auch das Gehirn über Hormone, z. B. Dopamin oder Noradrenalin, beeinflusst wird. Diese Hormone übermitteln nicht nur Signale, sondern fungieren zudem als Neurotransmitter, d. h., sie wirken auf die Informationsübertragung im synaptischen Spalt. Es handelt sich also bei Nervensystem und Hormonsystem um zwei Kommunikationsnetze, die auf unterschiedlichen Technologien beruhen. Der Hypothalamus agiert hier als ein Mittler zwischen diesen Netzen und übernimmt ähnliche Aufgaben wie eine Bridge oder ein Router.

Den Hormonen ähnlich sind übrigens die Pheromone, die allerdings nicht nach innen – in den Blutkreislauf –, sondern nach außen – in die Luft, auf den Boden usw. – abgegeben werden. Es handelt sich dabei um Duftstoffe, auf denen das Kommunikationssystem von Ameisen beruht; das wollen wir uns jetzt noch anschauen.

Gut vernetzt: der Ameisenstaat

Ameisen sind sehr erstaunliche Wesen (Abb. 6.28). Nicht nur, dass sie ungefähr 10 % der Biomasse aller auf dem Land lebenden Tiere ausmachen. Sie sind wie wir Menschen eusozial, d. h., sie leben in Verbänden, die mehrere Generationen umfassen, und zeigen im Zusammenhang mit der Arbeitsteilung altruistische Verhaltensweisen. Und auch hier gilt: Arbeitsteilung und Kommunikationswege sind das A und das O, damit komplexe Organisationsformen wie Ameisenstaaten erfolgreich sein können. Es ist in der Tat imposant, was Ameisen so alles bewerkstelligen können, obwohl jedes Individuum für sich nur über ein stecknadelgroßes, gehirnartiges Organ verfügt. Da gibt es Ameisenarten, die Landwirtschaft betreiben. Sie züchten einen Pilz und zeigen eindrucksvoll, wie man die Zucht



Abb. 6.28 Ameisen und ihre Nester. (© Judith Büsser (links), Eckart Zitzler (rechts))

und die Ernte arbeitsteilig vornimmt. Große Artgenossen schneiden von Bäumen, Blumen oder Gräsern Stücke ab und schleppen diese zum Nest. Dort zerkleinern kleinere Ameisen diese Stücke. Ein weiteres Team kleinerer Ameisen zerkaut diese Stücke, bis ein Brei von der Konsistenz eines schleimigen Grießbreis entsteht. Und auf diesem Brei wird schließlich der Pilz ausgesetzt, der darauf bestens gedeiht und schließlich geerntet werden kann. Andere Arten betreiben Viehzucht. Sie halten sich Schildläuse-Herden, die sie zu ihren Nahrungsorten auf Blätter bringen, wo diese Pflanzensäfte aufsaugen. Die Schildläuse scheiden Teile wertvoller Nahrungsbestandteile – Aminosäuren, Zucker etc. – aus und dienen dann den Ameisen als Nahrung. Und Treiberameisen können mehrere Meter breite Fronten bilden und dann einen Nahrungsfeldzug vollziehen, indem sie alles, was kreucht und fleucht, also andere Insekten und selbst kleine Schlangen töten.

Das Kommunikationssystem der Ameisen beruht vor allem auf den erwähnten Pheromonen. Diese Duftstoffe sind ebenfalls Drüsensekrete und werden von anderen Ameisen über ihre Fühler wahrgenommen; sie lösen bei den Empfängern stereotype Verhaltensweisen aus. Wie die Hormone sind die Pheromone hochspezifisch, d. h., verschiedene chemische Zusammensetzungen stehen für unterschiedliche Botschaften. Diese Duftstoffe können Verschiedenstes bewirken und haben verschiedenste Funktionen. Sie können das eigene Nest markieren, sie dienen Individuen als Ausweis, dass sie effektiv Teil der Kolonie sind, sie können anzeigen, wo Nahrung ist, oder geben das Marschtempo vor, mit dem eine Kolonie ausschwärmt. Auf diese Weise lösen Ameisen erstaunliche Probleme: Beispielsweise können sie im Kollektiv kürzeste Wege vom Nest zu Futterquellen finden, nur dadurch, dass die einzelnen Ameisen auf ihren individuellen Wegen Pheromone hinterlassen. Weil auf kürzeren Wegen die Pheromonkonzentration mit der Zeit schneller ansteigt als auf längeren Wegen und weil sich ausströmende Ameisen nach dem stärksten Duft richten, etablieren sich kurze Routen. Das wurde sogar in der Informatik nachgeahmt (siehe Anm. 6.6).

Anmerkung 6.6: Ameisen-Algorithmen

Das Verhalten von Ameisenvölkern wird übrigens auch als Inspiration genutzt, um das Hamilton-Pfad-Problem aus Kap. 1 zu lösen. Dazu wird eine Ameisenkolonie im Computer simuliert. Im Computer lässt man jede einzelne Ameise vom Startpunkt aus einen Weg zum Ziel finden; welche Abzweigung sie bei jeder Kreuzung nimmt, ist von der Anzahl der Pheromone abhängig, die auf der entsprechenden „Fahrbahn“ liegen (am Anfang sind alle Fahrbahnen frei von Pheromonen). Die Wahl ist zufallsgesteuert, die Wahrscheinlichkeit

errechnet sich aus dem Pheromonverhältnis der abzweigenden Fahrbahnen. Sind alle Ameisen so gelaufen, wird entschieden, welche Ameise auf ihrem Weg wie viele Pheromone hinterlassen darf. Das hängt wiederum von der Länge des gefundenen Weges ab: Je kürzer, desto mehr Pheromone darf die entsprechende Ameise auf ihrem Weg verteilen. Anschließend werden alle Ameisen wieder zurück zum Start gebracht und die Simulation der Wegsuche beginnt erneut, diesmal mit angepassten Pheromonkonzentrationen. Diese Prozedur wird ausreichend häufig wiederholt. Die Näherungslösung zum Hamilton-Pfad-Problem lässt sich dann an den Pheromonen ablesen, und zwar über eine gierige Strategie: Wir folgen vom Start immer der Fahrbahn, die am meisten Pheromone aufweist und die wir bisher noch nicht betreten haben.

Bemerkenswert ist an dieser Stelle, dass bei einem Ameisenstaat mehrere Kommunikationssysteme schichtenartig übereinander gestülpt sind (siehe Abb. 6.29). Das erinnert an das Schichtenprinzip des OSI-Modells. Mit jeder Schicht kommen eine neue Funktionalität und eine neue Komplexitätsstufe hinzu. Ameisen bilden Ameisenkolonien, also große Gemeinschaften, und koordinieren sich untereinander mittels der Pheromone; es entsteht ein großes Kommunikationsnetz. Jede einzelne Ameise verarbeitet die Signale in

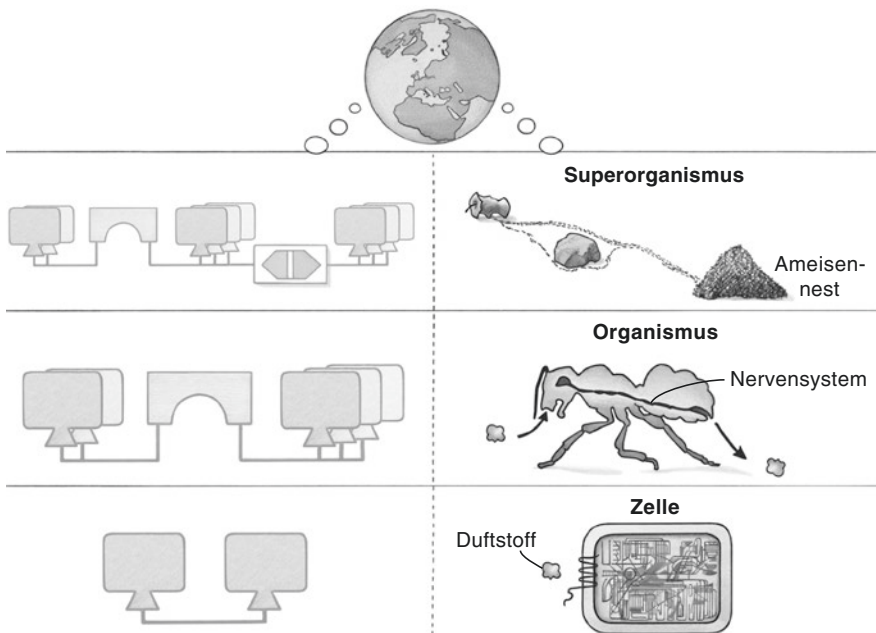


Abb. 6.29 Ein Ameisenstaat als geschichteter Superorganismus. (© Susanne Staubli, Eckart Zitzler)

ihrem Organismus, der mit seinem Nervensystem wiederum ein umfassendes Kommunikationsnetz ausbildet. Und wenn wir noch weiter hinabsteigen auf die Zellebene, dann hat es dort auch Kommunikationsnetze, z. B. gibt es ein Geflecht an Reaktionen und Stoffwechselvorgängen, an genetisch regulatorischen Abläufen. Ein Netz im Netz im Netz.

Es gibt also diverse biologische Organisationsebenen, und wenn wir uns den Ameisenstaat anschauen, dann spricht man von einem Superorganismus. Die Elemente dieser Lebensform sind nicht Zellen oder Gewebe, sondern individuelle Organismen, die zusammen den Superorganismus bilden. Die einzelne Ameise als solche hat keine Überlebenschance, sie ist auf die Gemeinschaft ausgerichtet, und damit Gemeinschaften mit mehreren Hunderttausend Einzelorganismen funktionieren, braucht es eben die Arbeitsteilung. Es ist schon verblüffend, wie sehr das an menschliche Gesellschaften erinnert: Auch Ameisen kennen unterschiedlichste Berufsgruppen, vom Wächter bis zum Sammler, von der Putzfrau bis zur Krankenschwester, vom Polizisten bis zum Bauern. Als das ist nur möglich über das faszinierende Kommunikationssystem der Pheromone.

Fazit

Die Geschichte der Computernetzwerke ist eigentlich eine Geschichte des Weltcomputers. Von Anfang an ging es dabei nicht nur um die Kommunikation, sondern auch um das Teilen und die Zusammenlegung von Ressourcen. Zuerst wurden einzelne oder mehrere Computer eingesetzt, um spezielle Aufgaben wie Datenspeicherung, Nachrichtenversand, Hochleistungsrechnen etc. zu übernehmen. Die Aufgabenteilung ist dann mit dem Aufkommen hoch performanter Übertragungsmedien stetig ausgebaut worden. Im globalen Verbund erbringen Gemeinschaften von Computern eine Masse an Dienstleistungen, die aufeinander aufbauen und eng miteinander verflochten sind. Dadurch hat sich ein eigenständiges, weltumspannendes Computersystem gebildet, dessen Kommunikationsgerüst das Internet ist. Darin sind jedoch nicht nur einzelne Computer eingebunden, sondern beliebige Geräte, die verschiedenste Sensoren und Aktoren beinhalten können. Dieses riesige eingebettete System ist ein wachsendes, schwer kontrollierbares Gebilde. Und auch hier lässt sich wieder beobachten, dass die zugrunde liegenden technischen Grundlagen – von den Bridges über die Router bis zu den Client-Server- und Peer-to-Peer-Konzepten – von der Idee her einfach sind, in ihren Ausprägungen aber enorm vielfältig sein und zu gänzlich neuen Phänomenen führen können. Das war in der Natur

nicht anders: Diverse ineinandergreifende Kommunikationssysteme haben das Fundament für immer komplexere Organismen und Verhaltensweisen gelegt.

Zum Nach- und Weiterlesen

- Diamond, J. (2006). *Arm und Reich: Die Schicksale menschlicher Gesellschaften* (9. Aufl.). Frankfurt a. M.: Fischer Taschenbuch.
- Gleick, J. (2011). *Die Information: Geschichte, Theorie, Flut*. München: Redline Verlag.
- Hölldobler, B., & Wilson, E. O. (2016). *Auf den Spuren der Ameisen: Die Entdeckung einer faszinierenden Welt* (3. Aufl.). Berlin: Springer Spektrum.
- Huch, R., & Jürgens, K. D. (2015). *Mensch, Körper, Krankheit: Anatomie, Physiologie, Krankheitsbilder* (7. Aufl.). München: Urban & Fischer & Elsevier.
- Schreiner, R. (2016). *Computernetzwerke: Von den Grundlagen zur Funktion und Anwendung* (6. Aufl.). München: Hanser.
- Tanenbaum, A. S., & Wetherall, D. J. (2012). *Computernetzwerke* (5. Aufl.). München: Pearson Studium.

Bildquellen

- Abb. 6.1: Ricochet64, Bildnummer 68431755, <https://stock.adobe.com> (rechts, abgerufen am 30.05.2017)
- Abb. 6.17: Thomas Aumann, Bildnummer 901378, <https://stock.adobe.com> (links, abgerufen am 30.05.2017); majorosl/Getty Images/iStock, Bildnummer 471158377, <http://www.istockphoto.com> (rechts, abgerufen am 30.05.2017)

7

Die Kunst der Darstellung

Überblick

Dieses Kapitel stellt die Gestalt der Information ins Zentrum. Ausgehend von der Frage, was Informationen sind und in welcher Form sie uns begegnen, untersuchen wir die Möglichkeiten, Informationen so darzustellen, dass sie von einem Computer verarbeitet werden können. Kurzum: Es geht um Codes, die Zeichen jedweder Art in Folgen von Nullen und Einsen überführen. Wir lernen einfache Codes zur Darstellung von Zahlen und Texten kennen und handeln uns von dort aus weiter zu komplexeren, zusammengesetzten Codes, wie sie bei Bildern oder Textlayouts zur Anwendung kommen. Anschließend setzen wir uns mit dem Thema der Kompression auseinander: Wie kompakt kann ein Code eigentlich sein? Im Gegenzug lernen wir aber auch Codes kennen, die man absichtlich aufbläst, um Relevantes in einem Wust von Irrelevantem zu verbergen. Am Ende werden wir sehen, wie sich Informationen so codieren lassen, dass sie von anderen nicht ohne Weiteres gelesen werden können. Kryptografie ist das Stichwort, dessen Grundlagen wir in diesem Kapitel ebenfalls durchleuchten werden.

Die automatisierte Verarbeitung von Zahlen stand ursprünglich im Vordergrund, als die ersten Computer entwickelt wurden. Doch schon bald konnten die elektronischen Rechner weit mehr als das, was sich Charles Babbage mit seinem Computerentwurf erträumt hatte: Texte, Bilder, Töne, ja sogar gesprochene Sprache aufnehmen, auswerten und produzieren. Aus dem Computer wurde ein universelles Gerät zur Informationsverarbeitung. Auf dem Weg dorthin gab es allerdings einige knifflige Fragen zu lösen, u. a. die der Darstellung: Wie lassen sich beliebige Information durch Aktivitätsmuster von Stromkreisen oder – von der technischen Realisierung

einmal abstrahiert – als Folgen von Nullen und Einsen repräsentieren? Um die verschiedenen Antworten, die dafür gefunden wurden, geht es hier.

Wir werden uns mit Codes beschäftigen, und das nicht nur, aber auch im Sinne von Geheimsprachen. Codes sind Vorschriften bzw. Vereinbarungen, wie Informationen mittels eines definierten Zeichenvorrats ausgedrückt werden. Der bekannte Morse-Code, bei dem Texte durch Abfolgen von kurzen und langen Signalen, möglicherweise Töne, übertragen werden, ist ein klassisches Beispiel. Je nachdem, für welchen Anwendungszweck wir einen Code konzipieren – für die Speicherung, die Verarbeitung oder die Übermittlung –, können sich ganz unterschiedliche Lösungen ergeben. Ja, die Kapitelüberschrift deutet es bereits an: Häufig ist es eine Kunst, einen guten Code zu finden. Es erwartet Sie ein reichhaltiges Thema, das Sie nicht unbedingt in einem Rutsch durchdringen müssen, sondern auch gut selektiv erkunden können.

Die Welt in Nullen und Einsen

Fangen wir vorne an: Wenn wir uns jetzt Gedanken über computergerechte Codes machen wollen, dann müssen wir uns zwangsläufig mit dem Begriff der Information auseinandersetzen. Leichter gesagt als getan, denn leicht zu fassen ist dieser Begriff nicht. Das macht aber nichts, für unsere Zwecke reicht es, wenn wir ein intuitives Verständnis davon haben, was mit Information gemeint ist.

Was ist Information?

Information ist ein Wort, das uns alltäglich umgibt: Wegweiser, Thermometer, Anleitungen, Gesteinsschichten – sie alle liefern uns Informationen. Auch in der Informatik ist dieser Begriff allgegenwärtig, schließlich geht es hier um die systematische Verarbeitung von Informationen. Umso verwunderlicher erscheint es, dass es für den neben Energie und Materie wichtigsten Grundbegriff in den Natur- und Ingenieurwissenschaften keine einheitliche, fassbare Definition gibt. Tatsächlich ist die Information ein Begriff, über den man philosophische Grundsatzdiskussionen führen kann, so unterschiedlich sind die Auffassungen in verschiedenen Disziplinen, was darunter eigentlich genau zu verstehen ist. Weil das Leben schon kompliziert genug ist, wollen wir uns hier nicht den Kopf zerbrechen. Wir können vereinfacht sagen,

eine Information ist eine Auskunft über die Welt; sie repräsentiert einen Ausschnitt der Welt und vermittelt uns so einen Eindruck von ihr. Eindruck ist das Wort, das uns zur Wortwurzel zurückführt, die der Altphilologe Klaus Bartels (2012) offen gelegt hat. Das griechische Wort *entýposis* und das davon abgeleitete lateinische Wort *informatio* bedeuten so viel wie „Einprägung“. War damit ursprünglich die Einprägung auf einer Münze gemeint, wurde daraus später der „Sinneseindruck“ – Informationen als Markierungen in der Welt, die ihren Abdruck in uns hinterlassen wie auf weichem Wachs. Oder aktiv ausgedrückt: Informationen sind Spuren in der Welt, die wir aufspüren. Das klingt fast schon poetisch.

Solche Spuren können sehr unterschiedlich daherkommen, wie in Abb. 7.1 anhand des Begriffs Apfel illustriert ist. Wir können einen konkreten Apfel

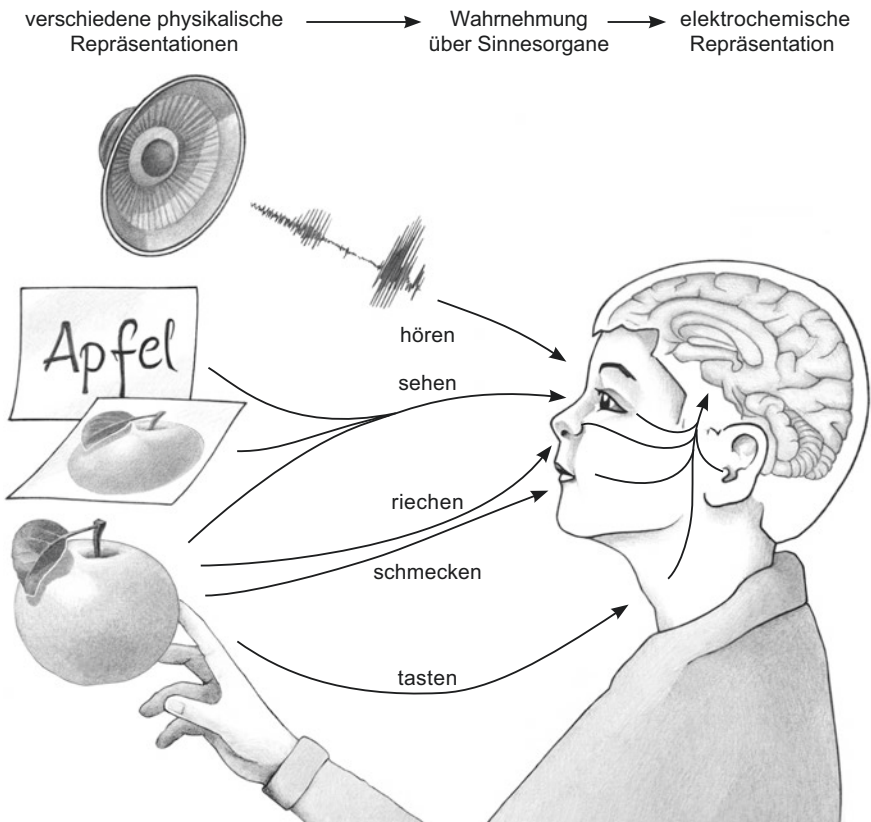


Abb. 7.1 Die gleiche Information kann auf unterschiedliche Arten und Weisen physikalisch repräsentiert und wahrgenommen werden. (© Susanne Staubli, Eckart Zitzler)

sehen, ihn jedoch genauso ertasten, oder, wenn wir hineinbeißen, schmecken. Alternativ können wir abstrahierte Beschreibungen eines Apfels erfassen, z. B. in Form einer Zeichnung, eines Schriftzugs oder eines gesprochenen Wortes. Die gleiche Information kann also auf verschiedene Arten physikalisch repräsentiert werden, z. B. optisch oder akustisch, ja sie wird auf ihrem Weg zum Empfänger sogar mehrfach umgewandelt, in unserem Gehirn liegt sie elektrochemisch vor. Das heißt auch, eine Information kann niemals losgelöst von dieser Welt existieren, sie ist immer in der ein oder anderen Form physikalisch präsent. Allerdings müssen wir eine physikalische Spur lesen können. Unsere Sinnesorgane eröffnen uns nur einen Teil der Welt, viele Aspekte bleiben uns verborgen, falls wir nicht über Hilfsmittel wie Mikroskope oder spezielle Messgeräte verfügen. Darüber hinaus müssen wir in der Lage sein, die wahrgenommene Spur zu interpretieren und die darin enthaltene Bedeutung zu erfassen. Ein Kind, das noch nie im Leben einen Apfel gesehen hat, aber Nektarinen kennt, wird der Apfel-Zeichnung eine ganz andere Bedeutung zuordnen. Interpretation setzt also Vorwissen voraus. Warum sonst wäre es wohl so schwierig, aus Gesteinsschichten die Erdgeschichte abzulesen? Ein Geologe muss ein großes Wissen anhäufen, um die Spuren richtig deuten können.

Es gibt also eine physikalische Ebene und eine Bedeutungsebene. Zudem können wir eine Zwischenebene betrachten, nämlich die der symbolischen Darstellung. Drücken wir eine Information über unsere Schriftsprache aus, so verwenden wir Buchstaben als Symbole und abstrahieren dabei von der physikalischen Repräsentation. Wenn wir von einem A sprechen, dann meinen wir ein A, unabhängig davon, ob es als getrocknete Tinte auf Papier, als Kombination von Lichtpunkten auf einem Bildschirm oder als Gravur auf einer Steinplatte in Erscheinung tritt. Informatikerinnen reden von Daten, wenn sie die symbolische Repräsentation einer Information meinen, und von Datenträgern, wenn sie sich auf die konkrete physikalische Speichermethode beziehen. Codieren wir das Foto eines Apfels als Sequenz von Nullen und Einsen, so können diese Daten auf dem Datenträger Festplatte durch magnetische Spuren festgehalten werden. Die Vorstellung vom Apfel entsteht schließlich beim Betrachter, wenn er das Foto auf dem Bildschirm anschaut (siehe Abb. 7.2).

Datenträger, Daten, Information – etwas abstrakt, das Ganze? Um die Verwirrung komplett zu machen, kann ich noch anfügen, dass die gleiche Information durch unterschiedliche Daten dargestellt und dass dieselben Daten unterschiedliche Informationen repräsentieren und durch unterschiedliche Datenträger festgehalten werden können. Doch keine Sorge, vieles wird sich im Folgenden klären, wenn wir konkrete, computerlesbare Darstellungen diskutieren. Wenn Sie sich an dieser Stelle fragen, ob es nun

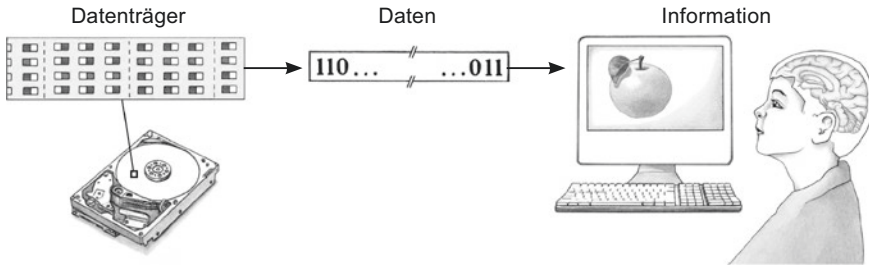


Abb. 7.2 Drei Betrachtungsebenen: Datenträger, Daten und Information. (© Susanne Staubli, Eckart Zitzler)

Daten- oder Informationsverarbeitung heißen müsse, erlauben Sie mir noch folgende Bemerkung. Ein Computer verarbeitet immer Daten; doch erst wenn er ihnen Bedeutungen zuordnet, verarbeitet er auch Informationen. Solange ich den Computer nur einsetze, um meine Fotosammlung zu organisieren, begnügt er sich mit Datenverarbeitung; ein selbstfahrender Roboter, der aufgrund von Kameradaten einen Weg ermittelt, muss hingegen aus Bildern Informationen extrahieren. Aber halten wir uns nicht mit Spitzfindigkeiten auf, sondern wenden uns lieber den Codes zu.

Von Zeichen und Codes

Wenn sich beim Volleyball Spielerinnen mittels geheimer Handzeichen verständigen, dann verwenden sie einen Code. Wollen wir am Geldautomaten Geld ziehen, benötigen wir einen PIN-Code. In Spionagethrillern werden die Namen der Agenten durch Tarnnamen codiert und auf VIP-Partys gibt es einen „Dresscode“. Ein Code trennt also diejenigen, die vom Code Kenntnis haben, von den anderen, doch ein Code muss nicht notwendigerweise geheim sein. Zunächst einmal ist ein Code nichts anderes als ein System von vereinbarten Zeichen, mit dem wir Informationen darstellen und austauschen. Zeichen können dabei Gesten, Laute, Schriftzeichen oder ganz allgemein unterscheidbare Markierungen sein. Die Gesamtheit der Zeichen, die einer Darstellung zugrunde liegt, nennt man Zeichenvorrat. In der Informatik verstehen wir unter einem Code allerdings nicht nur einen gemeinsamen Zeichenvorrat, sondern auch eine Übersetzungsvorschrift: Jedes Zeichen eines ursprünglichen Zeichenvorrats wird abgebildet auf ein sogenanntes Codewort, welches ein oder mehrere Zeichen des vereinbarten Zeichenvorrats umfasst. So gesehen dient ein Code dem Zweck, Informationen von einer Darstellung in eine andere Darstellung zu überführen. Bsp. 7.1 illustriert die Idee anhand des Morse-Codes.

Beispiel 7.1: Der Morse-Code

Der internationale Morse-Code beruht auf einem Verfahren, das sich der amerikanische Künstler Samuel F. B. Morse in den 1830er-Jahren ausgedacht hat. Er wollte Informationen mittels Telegrafenleitungen übermitteln und ersann dazu ein Verfahren, das Zahlen in Folgen von elektrischen Impulsen übersetzte; später wurde Morses Ansatz um Buchstaben und anderweitige Zeichen erweitert. Die Idee besteht darin, drei Arten von Signalen für die Codierung zu verwenden (Abb. 7.3): einen kurzen Stromimpuls, einen langen Stromimpuls und eine Pause, in der kein Impuls gesendet wird. Jedes Zeichen in einem Text wird übersetzt in eine Folge von kurzen und/oder langen Stromimpulsen, wobei zwischen jedem Impuls eine einheitlich lange Pause erfolgt. Ist ein Wort abgeschlossen, beträgt die Pause bis zum nächsten Wort drei Einheitspausen, bei Ende eines Satzes sieben Einheitspausen. Ein S beispielsweise besteht aus drei kurzen Signalen, ein O aus drei langen. Darüber hinaus besitzen einige Buchstaben wie das E oder das I sehr kurze Signalsequenzen, während wenig gebräuchliche Buchstaben wie das Q oder das Y mehrere Impulse umfassen.

In einem Computer besteht der zur Verfügung stehende Zeichenvorrat auf der Datenebene aus den zwei Symbolen 0 und 1. Und darum geht es im Folgenden: Wie lassen sich Zeichen eines beliebigen Zeichenvorrats so als 0-1-Folgen bzw. 0-1-Muster codieren, dass sie in ein Computersystem eingespeist, verarbeitet und letztlich auch wieder rücktransformiert werden können? Das Ziel dabei ist nicht, irgendeinen Code zu finden, sondern einen praktikablen. Je nach Einsatzgebiet sind manche Codes besser und manche schlechter

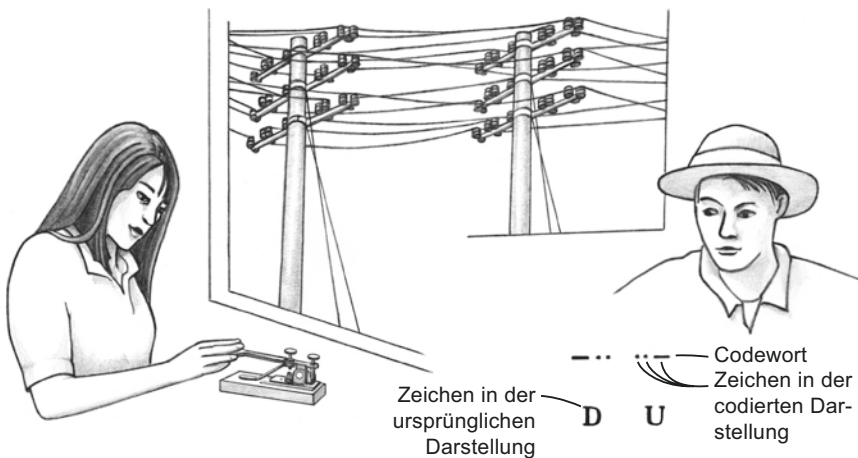


Abb. 7.3 Kommunikation über Telegrafenleitungen unter Verwendung des Morse-Codes. (© Susanne Staubli, Eckart Zitzler)

geeignet – es hängt von den Kriterien ab, die wir anlegen. Ein Kriterium ist häufig, dass der Code schnell verarbeitet werden kann. Ein anderes kann sein, dass die Anwendung des Codes, also die Codierung, sowie die Rückübersetzung, also die Decodierung, einfach zu bewerkstelligen sind. Wiederum ein anderes ist, dass die Codewörter möglichst kompakt sind. Das hatten wir ja bereits bei der Darstellung von Zahlen gesehen (vgl. Kap. 2). Der Dualzahlencode ist sowohl gut verarbeitbar als auch kompakt, ganz im Gegensatz zu einer möglichen Strichlistendarstellung, bei der die Anzahl der Einsen die Wertigkeit einer Zahl repräsentiert.

Grundlegende Darstellungsarten

Schauen wir uns zunächst einmal die grundlegenden Codes an, die in jedem Computersystem zum Einsatz kommen, und beginnen wir mit den Zahlen. Wir wissen bereits, wie sich natürliche Zahlen darstellen lassen, doch was ist mit negativen Zahlen, Zahlen mit Nachkommastellen, extrem großen oder extrem kleinen Zahlen? Hier lässt sich der einfache Dualzahlencode erweitern. Beispielsweise kann das Vorzeichen einer Zahl kurzerhand durch eine zusätzliche Binärziffer repräsentiert werden. Eine Dezimalzahl in Kommaschreibweise ist codierbar über zwei Dualzahlen: Die erste repräsentiert die Ziffern vor dem Komma, die zweite die Stellen nach dem Komma. Wenn man es geschickt anstellt, lassen sich sogar mehrere Fliegen mit einer Klappe schlagen; Bsp. 7.2 erläutert Ihnen, wie das geht.

Beispiel 7.2: Ein Zahlencode mit flexiblem Komma

Eine Möglichkeit, sowohl kleine als auch große Zahlen mit Nachkommastellen binär zu codieren, basiert auf der wissenschaftlichen Darstellung von Zahlen. Hierbei wird eine Grundzahl, Mantisse genannt, mit einer anderen Zahl – es handelt sich um die Zahl 10 potenziert mit einem Exponenten – multipliziert. So lässt sich z. B. die Anzahl der Nervenzellen im menschlichen Körper (30 Mrd.) als 3 mal 10 hoch 10 darstellen oder die Breite eines Spermienkopfes (3 μm) als 3 mal 10 hoch -6.

Abb. 7.4 illustriert den Aufbau des Zahlencodes, wobei anstelle der 10 die Zahl 2 als Basis für die Potenzierung genommen wird. Sie zeigt die Mantisse, ein Bit für das Vorzeichen (eine 1 heißt, die Zahl ist negativ, eine 0 positiv) und dann gibt es noch die Charakteristik, eine Dualzahl, aus der sich der Exponent berechnet, indem die Zahl 126 abgezogen wird (so lassen sich auch negative Exponenten einfach darstellen). Dieser Code nennt sich Gleitkommadarstellung, weil die Anzahl der Ziffern vor und nach dem Komma variieren kann; das Gegenstück dazu heißt Fixkommadarstellung. Damit Gleitkommazahlen verarbeitet werden können, braucht es spezielle Operationen. Prozessoren besitzen daher spezielle Gleitkommazahl-Verarbeitungseinheiten.

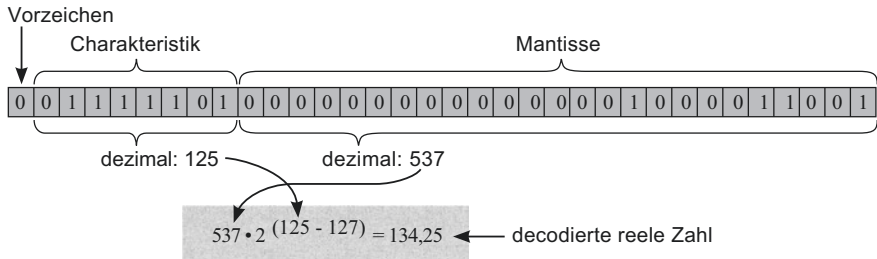


Abb. 7.4 Wie eine reelle Zahl binär codiert werden kann. (© Susanne Staubli, Eckart Zitzler)

Neben Zahlen sind es vor allem Texte in verschiedenster Ausprägung, mit denen Computer gefüttert werden. An sich ist es ja einfach, Buchstaben und andere Satzzeichen durch Dualzahlen darzustellen. Der Morse-Code macht es vor: Genauso wie man jedem Buchstaben eine Sequenz von Elementarsignalen zuweisen kann, kann er auch als eine 0-1-Sequenz fixierter Länge dargestellt werden. Schwierig kann es hingegen sein, sich auf einen Code zu einigen. Als die ersten Großcomputer kommerziell Fuß fassten und in verschiedenen Institutionen in Betrieb genommen wurden, herrschte Wildwuchs. Computer des einen Herstellers verstanden die Texte, die auf dem Computer eines anderen Herstellers gespeichert waren, nicht. Im übertragenen Sinne verwendeten die einen europäische Zeichen, die anderen arabische und wieder andere chinesische. Mit der Verbreitung des Internets wurde es natürlich notwendig, einen Code zu definieren, der alle Sprachen berücksichtigt, und so wurden Standardcodes wie ASCII und Unicode geboren (Bsp. 7.3).

Beispiel 7.3: Warum Text nicht gleich Text ist

Der erste Versuch der Standardisierung war der *American Standard Code for Information Interchange* (ASCII). Er wurde von einem Komitee in den USA entwickelt und am 17. Juni 1963 verabschiedet. Nicht dass damit alle Probleme gelöst waren, denn die ASCII-Darstellung war zu Anfang 7 Bit breit, später in den 1970er-Jahren arbeiteten Computer aber mit 8-Bit-Datenleitungen, sodass Hersteller das freie achte Bit wiederum dazu nutzten, herstellereigene Sonderzeichen zu definieren.

Eine Weiterentwicklung ist der Unicode, der von einem internationalen gleichnamigen Konsortium definiert wird. Der Unicode weist wie der ASCII-Code Schriftsymbolen eindeutige Nummern zu, die sich auf diverse Arten in Form von Dualzahlen darstellen lassen. Die mit der Verbreitung des Internets am weitesten verwendete Variante ist in Abb. 7.5 skizziert – sie hat die kryptische Bezeichnung UTF-8, eine Abkürzung für die Wortschlange *Universal Coded*

Nummer	Byte-Codierung
0 – 127	0 x x x x x x x
128 – 2047	1 1 0 x x x x x 1 0 x x x x x x
2048 – 65535	1 1 1 0 x x x x 1 0 x x x x x x 1 0 x x x x x x
65535 – 1114111	1 1 1 1 0 x x x 1 0 x x x x x x 1 0 x x x x x x 1 0 x x x x x x
Beispiele:	
D → 68 →	0 1 0 0 0 1 0 0
Ď → 270 →	1 1 0 0 0 1 0 0 1 0 0 0 1 1 1 0

Abb. 7.5 Eine mögliche Implementierung des Unicode-Standards. (© Susanne Staubli, Eckart Zitzler)

Character Set + Transformation Format – 8-bit. Je nachdem, wie groß die entsprechende Nummer ist, werden ein bis vier Bytes verwendet. Die Nummern 0 bis 127 entsprechen den ASCII-Codes und werden in der gleichen Art und Weise als achtstellige Dualzahlen dargestellt. Bei höheren Zahlen werden mehr Bytes verwendet, dabei sind gewisse Bits fixiert und nur die mit x gekennzeichneten Bitpositionen können genutzt werden, um die entsprechende Zahl darzustellen. Beispielsweise hat der Großbuchstabe D im Unicode die Nummer 68, als Dualzahl 1000100; die Dualzahl wird in einem Byte codiert, wobei sieben Stellen für die Dualzahl zur Verfügung stehen. Der Großbuchstabe D mit Häkchen (Hatschek), der beispielsweise im Tschechischen verwendet wird, hat die Nummer 270 und wird daher mittels zwei Bytes codiert; die elf mit x gekennzeichneten Stellen werden rechtsbündig mit der entsprechenden Dualzahl 100001110 gefüllt.

Die Darstellungsformen für ganze Zahlen, reelle Zahlen und Texte bilden die Grundlage für viele weitere Codes, wie Sie gleich sehen werden.

Aufs Format kommt’s an

Müssen komplexe Informationen abgebildet werden, kommen häufig zusammengesetzte Codes zum Einsatz. Sie umfassen mehrere Codebestandteile, die möglicherweise hierarchisch organisiert sind. Ein Beispiel für einen solchen zusammengesetzten Code haben wir bereits in Kap. 3 kennengelernt: den Maschinencode. Das vorgestellte Instruktionsformat beschreibt in einem Byte, um welchen Befehl es sich handelt, und dabei werden Zahlencodes (die Nummern der Register) und anderweitige Codes (um welchen Befehlstyp es sich handelt) miteinander kombiniert.

Ein in der Informatik weit verbreitetes Codeschema bildet Informationen in Form von Zahlenketten ab. Bilder werden beispielsweise so codiert. Dazu wird die Wirklichkeit gerastert, d. h., ihr wird ein Gitter übergestülpt, wie in Abb. 7.6 dargestellt. Am einfachsten ist es, wenn wir ein Abbild in Form eines Schwarz-Weiß-Bildes speichern wollen; hier müssen wir pro Rasterquadrat entscheiden, ob die Helligkeit eher weiß oder eher schwarz ist, und weisen ihm so eine 1 oder 0 zu. Das nennt man eine Bitmap, weil jedes Rasterquadrat durch ein einzelnes Bit dargestellt werden kann. Die 0-1-Folge ergibt sich, wenn wir das Raster wie ein Buch durchlaufen, also zeilenweise von links nach rechts und die Zeilen in der Reihenfolge von oben nach unten. Wenn wir hingegen an einem Graustufenbild interessiert sind, bei dem jeder Bildpunkt – ein Rasterquadrat – einen Grauwert annimmt, benötigen wir mehr Bits pro Rasterquadrat. Um beispielsweise 256 verschiedene Stufen zwischen Weiß und Schwarz zu unterscheiden, benötigen wir 8 Bits bzw. 1 Byte; jeder Grauwert ist als achtstellige Dualzahl repräsentiert, wie in Abb. 7.6 skizziert. Das Gesamtbild ergibt sich demnach als Folge von Bytes, wiederum von links nach rechts und von oben nach unten. Übrigens nennt man die Bildpunkte auch Pixel, eine Kombination aus den englischen Wörtern *picture* und *element*.

Die gerasterte Form der Bilddarstellung, bei der sich eine Folge von Zahlen ergibt und jede Zahl ein Pixel codiert, ist die natürliche Bilddarstellung in einem Computer. Selbst Farbbilder lassen sich so codieren,

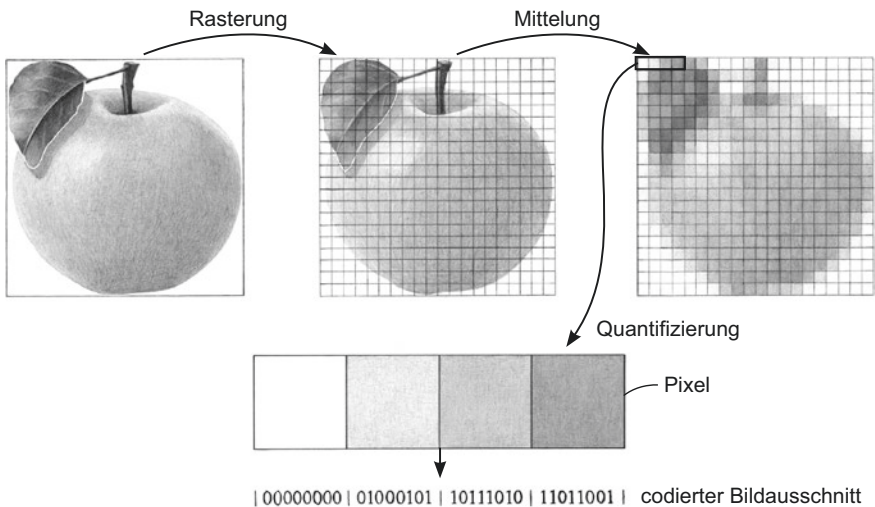


Abb. 7.6 Wie visuelle Informationen durch Rasterung in eine 0–1-Folge umgewandelt werden können, die ein Graustufenabbild darstellt. (© Susanne Staubli, Eckart Zitzler)

wie Bsp. 7.4 aufzeigt. Denn einerseits sind Bildschirme so aufgebaut, dass sie ein Bild als Gitter von winzig kleinen Bildpunkten darstellen. Andererseits funktioniert auch die Aufnahme von Bildern so: Bildsensoren bestehen wie die Facettenaugen einer Fliege oder wie unsere Netzhaut aus vielen kleinen Sensoren, die Licht registrieren und Lichtintensitäten melden. So gesehen ahmen Rasterbilder das Auge nach. Ähnlich ist das bei anderen Arten von Informationen: Audiosignale werden ebenfalls gerastert, über die Zeit werden viele kleine Einzelmessungen (Abtastpunkte) vorgenommen, die dann als Sequenz von Dualzahlen repräsentiert werden (vgl. Bsp. 4.1).

Beispiel 7.4: Darstellung von Farbbildern

Es gibt mehrere Arten von Farbsystemen, und zwei sind uns bestens aus dem Alltag bekannt (Abb. 7.7). Unsere Wahrnehmung basiert auf der additiven Farbmischung, bei der es drei Grundfarben gibt, nämlich Rot, Grün und Blau; für jede dieser drei Farben existieren in unserem Auge spezialisierte Lichtrezeptoren. Weißes Licht entsteht, wenn wir alle Grundfarben überlagern. Bei diesem Schema wird ein Pixel durch drei Dualzahlen beschrieben, die die Intensitäten der drei Grundfarben darstellen. Diese Darstellung ist als RGB (als Abkürzung der drei Grundfarben) bekannt.

Die subtraktive Farbmischung kennen wir vom Malen: Basierend auf drei Grundfarben, den Primärfarben Gelb, Rot und Blau, können wir fast alle Farben mischen; Schwarz ergibt sich durch die Kombination aller Mischfarben. Weiß ergibt sich, wenn keine der Primärfarben vorhanden ist. Dieses Farbsystem wird beim Drucken angewendet, das entsprechende Code-Schema nennt sich CMYK, wobei C für *cyan* (Türkisblau), M für *magenta* (Rotviolett), Y für *yellow* (Gelb) und K für *black* (Schwarz) steht. Schwarz wird zusätzlich verwendet, um die Abdeckung zu maximieren. Ein Pixel wird also durch vier Dualzahlen beschrieben.

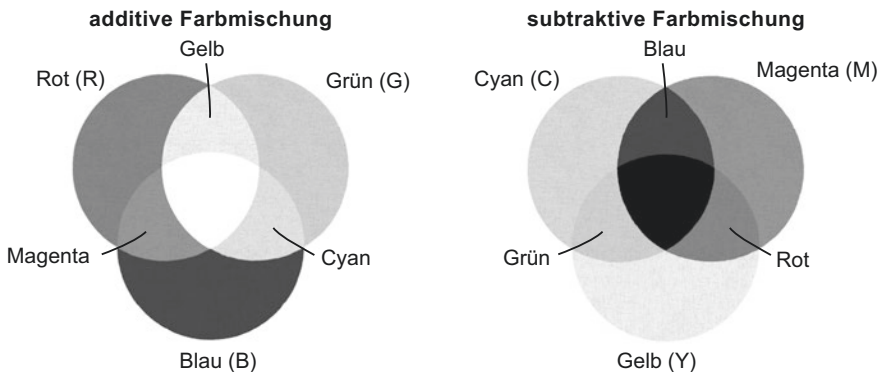


Abb. 7.7 Der RGB-Farbcode (links) und der CMYK-Farbcode (rechts). (© Susanne, Staubli, Eckart Zitzler)

Es gibt jedoch noch andere Möglichkeiten, Bilder zu beschreiben, und zwar wenn es um Zeichnungen geht, insbesondere solche, die mit dem Computer erstellt wurden. Das Prinzip nennt sich Vektorgrafik, und hierbei wird das Bild über die Objekte beschrieben, die auf diesem Bild zu sehen sind: Punkte, Linien, Flächen, Kreise und anderweitige Körper, Text usw. Eine Vektorgrafik ist also eine Liste von Objekten, die in einem Koordinatensystem angeordnet sind, und jedes Objekt hat bestimmte Merkmale. Ein Kreis beispielsweise ist definiert durch einen Mittelpunkt und einen Radius, ein Rechteck durch zwei gegenüberliegende Eckpunkte. Darüber hinaus können noch zusätzliche Eigenschaften wie Dicke der Kontur, Füllung, Schraffur etc. spezifiziert werden. Dies ist dann ein Vorteil, wenn die Auflösung verändert werden soll; bei Rastergrafiken sind Vergrößerungen nur beschränkt möglich. Vektorgrafiken werden z. B. bei technischen Zeichnungen, digitalen Landkarten und Computerspielen eingesetzt. Häufig werden Vektorgrafiken in Form von Texten beschrieben, die dann standardmäßig mit dem Unicode dargestellt werden. Die Beschreibungssprache PostScript, die in den 1980er-Jahren entwickelt wurde und als Vorläufer des *Portable Document Format* (PDF) gilt, ist ein Beispiel hierfür.

Viele Arten von Informationen, vor allem wenn sie vom Menschen direkt gelesen und verändert werden sollen, sind textbasiert und beschreiben die Informationen in strukturierter Textform. Das ist insbesondere für die traditionelle Programmierung wichtig, denn Programme werden zumeist in Textform codiert, d. h. in pseudosprachlichen Ausdrücken, die mathematischen Ausdrücken ähnlich sind. Die Assemblersprache aus Kap. 3 ist ein Beispiel, bei dem der Maschinencode in für den Menschen lesbarer Form dargestellt wird. Textbasierte Codierungen kommen jedoch auch dann zum Einsatz, wenn sie auf vielen Computern einfach lesbar sein sollen. Beispielsweise werden Webseiten über eine textbasierte Beschreibungssprache dargestellt, wie Bsp. 7.5 aufzeigt.

Beispiel 7.5: Sprachen zur Beschreibung von Seiten

Als das World Wide Web entstand, konzipierten seine Entwickler auch ein Format – die Hypertext Markup Language, kurz HTML –, um Dokumente mit Browsern darzustellen. Der Inhalt einer Webseite wird ganz normal unter Verwendung des Unicodes als Text dargestellt. Dieser Text wird ergänzt durch Markierungen, die die innere Struktur des Dokuments explizit machen, so wie in Abb. 7.8 oben gezeigt. Der Beginn einer Markierung wird mit der Zeichenfolge „<...>“ und das Ende mit der Zeichenfolge „</...>“ eingeläutet, die Pünktchen werden durch einen Buchstabencode ersetzt, der den Typ der Markierung beschreibt. Zum Beispiel kennzeichnet „<P>“ bzw. „</P>“ Anfang

HTML-Source-Code

```
<!DOCTYPE html>
<html>
<body>

<h1>Großzügige Überschrift</h1>
<h2>Kleine Überschrift</h2>
<p>Dies ist ein Absatz gefolgt
von einer Aufzählung:</p>
<ol type="1">
<li>Erster Punkt.</li>
<li>Zweiter Punkt.</li>
<li>Dritter Punkt.</li>
</ol>

</body>
</html>
```

HTML-Ausgabe

Große Überschrift**Kleine Überschrift**

Dies ist ein Absatz gefolgt von einer Aufzählung:

1. Erster Punkt.
2. Zweiter Punkt.
3. Dritter Punkt.

LaTeX-Source-Code

```
\documentclass{article}
\usepackage[german]{babel}

\begin{document}

\section*{Große
Überschrift}
\subsection*{Kleine
Überschrift}
Dies ist ein Absatz gefolgt von
einer Aufzählung:
\begin{enumerate}
\item Erster Punkt.
\item Zweiter Punkt.
\item Dritter Punkt.
\end{enumerate}

\end{document}
```

LaTeX-Ausgabe

Große Überschrift**Kleine Überschrift**

Dies ist ein Absatz gefolgt von einer Aufzählung:

1. Erster Punkt.
2. Zweiter Punkt.
3. Dritter Punkt.

Abb. 7.8 HTML- und LaTeX-Code zur Beschreibung einer einfachen Textseite. (© Susanne Staubli, Eckart Zitzler)

bzw. Ende eines Paragraphen, während „“ und „“ einen hervorgehobenen Textausschnitt markieren. Es gibt viele andere Markierungen, z. B. zur Kennzeichnung von Überschriften.

Es gibt noch viele weitere solcher textbasierter Beschreibungssprachen, z. B. LaTeX. LaTeX beruht auf einem von dem berühmten amerikanischen Informatiker Donald E. Knuth, ebenfalls Turing-Award-Träger, entwickelten Satzsystem, mit dem Informatiker häufig ihre wissenschaftlichen Papers schreiben. Der Vorteil bei solchen textbasierten Seitenbeschreibungen: Die Daten sind übertragbar und lesbar auf verschiedensten Computern, und auch die Darstellung kann den Möglichkeiten eines Computers angepasst werden.

Die textbasierten Darstellungsarten zeigen auf, dass zusammengesetzte Codes sehr kompliziert werden können, und entsprechend aufwändig kann die Decodierung auch sein. Häufig spricht man in diesem Zusammenhang auch von Datenformaten. Ein Datenformat strukturiert Daten und definiert, wie eine 0-1-Folge zu interpretieren ist. Es gibt Datenformate für jedwede Art von Informationen, in der Regel werden dabei diverse Zusatzinformationen mit codiert. Bei einem Bild wird z. B. angegeben, in welcher Auflösung, mit welchem Graubereich, mit welchem Koordinatensystem usw. das Bild ausgerüstet ist oder auch mit welcher Kamera, an welchem Ort, zu welcher Zeit ein Foto aufgenommen wurde. Und weil man genau wissen muss, ob es sich bei einer 0-1-Sequenz um ein Maschinenprogramm handelt, um ein Bild, um eine Zahlensequenz, um einen Text etc., hat man sich angewöhnt, Dateien mit Endungen zu versehen, die das eingesetzte Datenformat widerspiegeln. Es würde den Rahmen dieses Buches sprengen, wenn ich Ihnen hier konkrete Formate im Detail vorstellen wollte. Schauen wir uns lieber an, welche Codes es in der Biologie gibt.

Biologische Codes

Natürlich hat der Mensch die Codes nicht erfunden, auch wenn er unbestritten ein Faible für Zeichen, insbesondere für geheime Zeichen hat. Codes sind in der Biologie allgegenwärtig: Prächtige Geweihe und Gefieder stehen für Stärke und das Strotzen vor Gesundheit; Gebärden signalisieren Drohungen und Angriffsbereitschaft; grelle Farben können „Achtung, ich bin giftig!“ anzeigen. Unsere Sprache ist ebenfalls ein Code, wenn auch ein äußerst komplexer: In der Schriftsprache umfasst der Zeichenvorrat Klein- und Großbuchstaben sowie die Satzzeichen, in der Lautsprache sind es Phoneme, also Lautelemente, die zu Wörtern kombiniert werden; Wörter werden zu Sätzen kombiniert und Sätze wiederum zu komplexen Aussagen. Auch Singvögel verwenden einen artspezifischen Code, wenn sie ihre Arien trällern; in diesem Fall sind die Zeichen fixe Gesangselemente, aus denen sich Phrasen, Silben, Motive und ganze Strophen zusammensetzen (Abb. 7.9).

Doch ein Code stand am Anfang des Lebens, und der ist auch heute noch die Basis jeglichen Lebens: Die Rede ist vom genetischen Code. Der genetische Code legt fest, wie die Baupläne von Proteinen auf der DNA codiert werden. Dabei wird ein Zeichen, das dem Bauteil eines Proteins entspricht, durch drei Zeichen auf der DNA repräsentiert (siehe Bsp. 7.6). Der genetische Code ist universell und wird in allen Lebewesen verwendet.

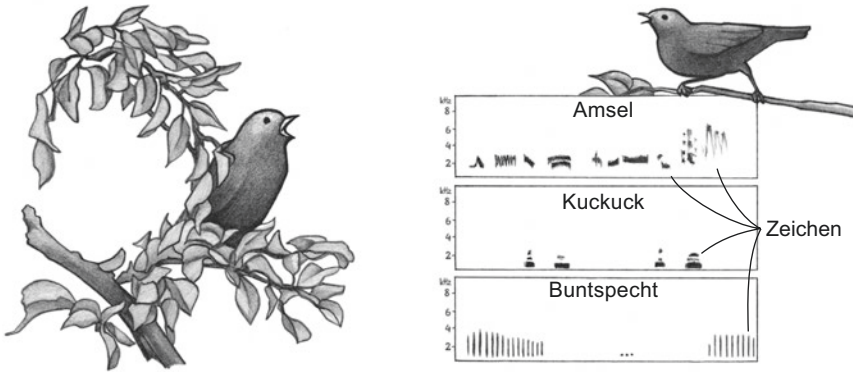


Abb. 7.9 Elementare Zeichen im Gesang verschiedener Vogelarten. (© Susanne Staubli, Eckart Zitzler)

Beispiel 7.6: Der genetische Code

Wie wir in Kap. 1 gesehen haben, codieren Sequenzen auf der DNA, Gene genannt, die Baupläne für Proteine, und Proteine wiederum sind Sequenzen von Aminosäuren. Die Bestandteile der DNA, die vier möglichen Nucleotide, hatten wir mit den Buchstaben A, C, G, T abgekürzt, und auch die Aminosäuren – mit wenigen Ausnahmen verwenden alle Zellen 20 Aminosäuren – werden mit Buchstaben abgekürzt. Die Frage ist nun, wie ein aus 20 Buchstaben bestehender Text durch einen Text, der nur vier Buchstaben verwenden darf, abgebildet werden kann.

Auch hier gibt es einen Code, und zwar den genetischen Code. Dieser ist so beschaffen, dass immer drei aufeinanderfolgende Nucleotide – auch Basentriplett genannt – eine Aminosäure repräsentieren, d. h., drei Zeichen auf der DNA-Seite werden in ein Zeichen auf Protein-Seite übersetzt. So wird beispielsweise GCA in A (Alanin) und ACA in T (Threonin) übersetzt. Dazwischen gibt es noch die Abschrift in Form eines mRNA-Moleküls; diese ist im Wesentlichen identisch zur DNA-Sequenz, doch werden T-Nucleotide durch ein anderes Molekül repräsentiert, das mit U abgekürzt wird. Der genetische Code ist also ein zweistufiger, wie in Abb. 7.10 illustriert ist. Dabei gibt es für einige Aminosäuren mehrere Codewörter, während andere nur über ein Basentriplett dargestellt werden können.

Biologische Codes begegnen uns in diesem Buch an verschiedenen Stellen, seien es nun die Anfangs- und Endmarkierungen von Genen (Kap. 2), die Frequenzcodierung beim Nervensystem (Kap. 4), die Signalcodierung mittels Hormonen (Kap. 6) oder die Codierung von Düften, auf die ich später noch zu sprechen komme.

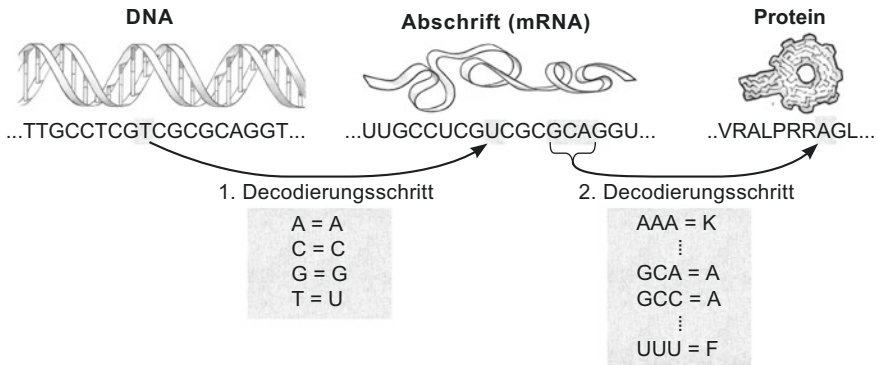


Abb. 7.10 Illustration des genetischen Codes. (© Susanne Staubli, Eckart Zitzler)

Heiße Luft

Eine zentrale Frage beim Entwurf eines Codes ist, wie viele Nullen und Einsen nötig sind, um die erforderliche Informationsmenge darzustellen. Zumeist ist man an kompakten Codes interessiert, vor allem bei der Speicherung und der Übertragung von Daten. Manchmal wählt man jedoch absichtlich Codewörter, die überflüssige Zeichen enthalten. Es geht also um die berühmte heiße Luft, die in einem Rede- bzw. Datenschwall stecken kann.

Unsere Sprache ist dazu angetan, heiße Luft aufzunehmen – schließlich kommt der Ausdruck ja daher, dass jemand schwätzt, ohne effektiv etwas zu sagen. In der Politik werden unangenehme Aussagen oft in einer Wolke heißer Luft versteckt. Dem stehen sprachliche Kunstformen entgegen, bei denen jedes Wort zählt. Gedichte verdichten häufig Erfahrungen oder Erkenntnisse und bringen diese auf den Punkt; manchmal werden sogar Aspekte absichtlich ausgelassen, um Doppeldeutigkeiten und Interpretationen zu ermöglichen. Doch heiße Luft ist nicht per se negativ, im Gegenteil: Ein großer Vorteil der Luftigkeit ist, dass wir Schreib- oder Sprechfehler gut erkennen und problemlos korrigieren oder sogar ignorieren können. Selbst wenn wir einen Text radikal verändern, wie in Abb. 7.11 zu sehen ist, können wir erstaunlicherweise dessen Sinn dennoch erschließen.

Heiße Luft kann also bewusst hinzugefügt oder gezielt entfernt werden. Das nutzt man auch in der Informatik aus, wir werden uns konkrete Anwendungen später anschauen. Doch vorher müssen wir klären, was genau unter heißer Luft zu verstehen ist, wenn wir diese Bezeichnung im Zusammenhang mit Codes verwenden.

Wir können einen Text selbst
dann noch lesen, wenn die
untere Hälfte jeder Zeile ab-
geschnitten ist.

Gunseao knönen wir die
Bbahucestn im Innenrn jdees
Wetros blieeibg udmoren
und den Txet troztedm lseen.

Abb. 7.11 Heiße Luft in Texten. (© Susanne Staubli, Eckart Zitzler)

Wie viel Information steckt da drin?

Wir haben bereits gesehen, dass sich dieselben Informationen auf unterschiedliche Art und Weise codieren lassen. Mal ist das resultierende 0-1-Muster länger, mal kürzer. Doch lässt sich auch ermitteln, wie viele Nullen und Einsen es im Minimum braucht, um eine bestimmte Anzahl unterschiedlicher Informationen – das können ganze Aussagen sein, aber auch nur einzelne Buchstaben, was hier aber keine Rolle spielen soll – darzustellen? Tatsächlich lässt sich so eine untere Grenze für die Länge der Codewörter angeben. Sie wird in Bits bemessen (Anm. 7.1), und die Differenz zwischen dem effektiven und dem minimalen Bitbedarf spiegelt das Ausmaß der enthaltenen heißen Luft wider.

Anmerkung 7.1: Das berühmte Bit

Vom Bit war schon mehrfach die Rede. Es handelt sich dabei um ein Kunstwort aus dem englischen Begriff *binary digit* und steht – wenn wir 0-1-Sequenzen als Dualzahlen interpretieren – für eine bestimmte Ziffernposition. Eine Dualzahl mit fünf Stellen bzw. die Zeichensequenz 10101 umfasst also fünf Bits. In einer digitalen Schaltung überträgt jede Leitung genau ein Bit (es fließt Strom oder eben nicht), aber wir können, wie wir gesehen haben, das Bit auch generalisieren: Licht an oder aus, Loch oder nicht, Rauch oder nicht, Ton oder nicht usw. Das Bit ist also nichts anderes als eine abstrakte Beschreibung für ein elementares Unterscheidungsmerkmal, ein physikalisches Merkmal, das genau zwei Zustände einnehmen kann. Beispielsweise bräuchte es zwei Bits, um die vier möglichen Zeichen an einer Position in einer DNA zu beschreiben.

Auf den ersten Blick scheint es leicht, die minimale Codewortlänge zu ermitteln. Es kommt einzig darauf an, wie viele Zeichen zu codieren sind. Da jedes Bit zwei Darstellungsmöglichkeiten bietet, lautet die Frage: Wie oft müssen wir Zwei mit sich selbst multiplizieren, sodass wir die Anzahl der zu repräsentierenden Zeichen erreichen bzw. übertreffen? Die Antwort dazu liefert der duale Logarithmus. Bei 26 Zeichen käme 4,7 heraus, und da es nur ganze Bits gibt, müssten wir minimal fünf Bits verwenden, um das Alphabet unserer Sprache abzubilden. Auf den zweiten Blick ist es

jedoch komplizierter, denn so ein Code wird in der Regel ja mehrfach und in Form von Zeichensequenzen eingesetzt, sei es nun bei der Übertragung von Textnachrichten oder der Speicherung von Rasterbildern. Jetzt spielt auch eine Rolle, wie oft die einzelnen Zeichen durchschnittlich verwendet werden. In deutschsprachigen Texten beispielsweise überwiegen Vokalbuchstaben gegenüber den anderen Buchstaben. Wenn wir nun häufiger auftretende Buchstaben mit weniger Bits codieren als die seltener auftretenden, dann lassen sich insgesamt Bits sparen, wie Bsp. 7.7 demonstriert.

Beispiel 7.7: Wie sich Bits sparen lassen

In Abb. 7.12 sehen Sie drei Zeichenfolgen mit jeweils acht Buchstabenvorkommen und darüber jeweils die binäre Codierung der einzelnen Zeichen. Im ersten Fall steht an jeder Position ein anderer Buchstabe, kein Buchstabe tritt mehrfach auf und wir müssen demgemäß jede Textstelle auf ein anderes Codewort abbilden. Insgesamt sind acht verschiedene Zeichen zu repräsentieren bzw. acht verschiedene Codewörter erforderlich, weshalb drei Bits in der binären Darstellung genügen. Das ist die einfache Situation. Was ändert sich nun, wenn die Zeichenfolge Wiederholungen enthält?

Im zweiten Fall kommen nur vier Buchstaben vor, die dafür aber doppelt. Wir könnten wie im ersten Fall jedem Zeichenvorkommen in der Sequenz ein eindeutiges dreistelliges Codewort zuordnen. Da wir aber nicht Vorkommen, sondern nur Zeichen unterscheiden müssen, benötigen wir auch nur vier und nicht acht verschiedene Codewörter. Das können wir beispielsweise erreichen, indem wir die Codewörter, die das gleiche Zeichen repräsentieren, zusammenfassen. So wird aus den Codewörtern 000 und 001 für das A das einzelne Codewort 00; das dritte Bit ist obsolet, weil es nur die Vorkommen unterscheidet.

Im dritten Fall beinhaltet die Sequenz ebenfalls nur vier Buchstaben, die jedoch unterschiedlich häufig vorkommen. Wir können jetzt dieselbe Argumentation wie im zweiten Fall anwenden und Codewörter für identische Zeichen zusammenfassen. Dann benötigen wir für das A nur ein Bit, für das B zwei Bits und für C und D jeweils drei Bits. Wir sehen, welchen Einfluss die Häufigkeit der Buchstaben auf die Länge der Codierung hat. Im ersten Fall benötigt die Darstellung insgesamt 24 Bits, im zweiten Fall 16 Bits und im dritten Fall 14 Bits.

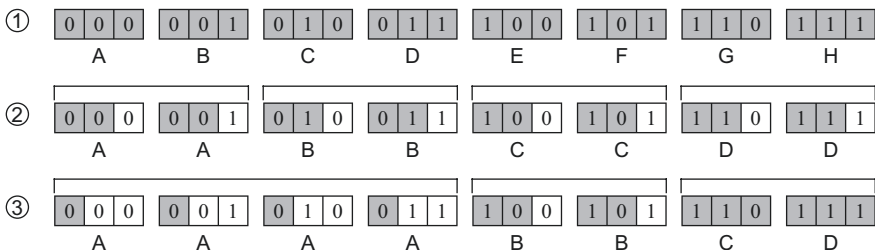


Abb. 7.12 Wie sich Folgen von acht Buchstaben auf kompakte Weise binär codieren lassen. (© Susanne Staubli, Eckart Zitzler)

Die in Bsp. 7.7 angestellten Überlegungen lassen sich verallgemeinern und führen schließlich zum Konzept der Entropie. Die Entropie ist ein künstliches Maß und bezeichnet die Anzahl der Bits, mit der wir im Mittel pro Zeichen rechnen müssen. Oder ganz genau ausgedrückt: Die Entropie gibt die minimale durchschnittliche Codewortlänge für einen Zeichenvorrat an, unter der Voraussetzung, dass die Auftretenswahrscheinlichkeiten aller Zeichen bekannt sind (Anm. 7.2). Damit ermittelt sie die gesuchte untere Grenze, denn mit weniger Bits, als die Entropie angibt, kommt kein Code aus. Die Entropie wird manchmal auch als Informationsgehalt bezeichnet. Dies ist vor einem Kommunikationshintergrund zu verstehen: Je größer die Entropie, desto größer ist die Unsicherheit, um welche Information es sich wohl handelt, wenn das nächste Bit gesendet wird. Sie ist maximal, wenn jedes Zeichen gleich wahrscheinlich ist. In natürlichsprachlichen Texten ist dies jedoch nicht der Fall: Wenn ich Ihnen beispielsweise ein Wort übermitteln möchte und den Anfangsbuchstaben sende, dann wissen Sie zwar noch nicht, um welches Wort es sich handelt, aber die Zahl der möglichen Wörter ist bei einem Y wesentlich kleiner als bei einem E. Das Y reduziert die Unsicherheit also in höherem Maße als das E.

Anmerkung 7.2: Claude Shannon und die Entropie

Die Definition der Entropie ist recht abstrakt und lässt sich am besten anhand eines konkreten Szenarios erklären. Nehmen wir also an, die Zeichenkette „DU BIST MEIN ICH BIN DEIN“ sei vorgegeben und wir möchten wissen, wie viele Bits durchschnittlich pro Zeichen benötigt werden. Die Idee ist nun: Wir berechnen zuerst, wie viele Bits notwendig wären, wenn alle Zeichen verschieden wären, und ziehen dann die Bits ab, die durch das Zusammenziehen von Codewörtern eingespart werden können, wenn Buchstaben mehrfach vorkommen.

Konkret gehen wir zeichenweise vor und ermitteln für jeden vorkommenden Buchstaben (das Leerzeichen zähle auch als eigenständiger Buchstabe), wie viele Bits das entsprechende Codewort mindestens umfassen muss. Dafür berechnen wir zunächst, wie viele Bits wir pro Codewort bräuchten, wären alle Zeichen unterschiedlich: Diese Zahl entspricht dem dualen Logarithmus von der Länge der Zeichenkette, bei 25 Zeichenvorkommen kommt da 4,64 heraus. Anschließend berechnen wir für jeden Buchstaben, wie viele Bits wir sparen können, weil er mehrfach vorkommt: Diese Zahl ergibt sich aus dem dualen Logarithmus von der Anzahl des Vorkommens des Buchstabens; beim N, das dreimal auftritt, ergibt dies 1,58. Ziehen wir letzteren Wert von ersterem ab, erhalten wir die minimale Länge des Codeworts für einen spezifischen Buchstaben; beim N wäre diese 3,06.

Auf dieser Basis können wir berechnen, wie viele Bits wir im Durchschnitt pro Zeichenvorkommen aufwenden müssen: Für jede Position in der Zeichenkette ermitteln wir die minimale Codewortlänge des entsprechenden Buchstabens, addieren diese Werte auf und teilen anschließend durch die

Länge der Zeichenkette. Dieser Mittelwert wird als Entropie bezeichnet und man kann zeigen, dass es keine Codierung gibt, die weniger Bits pro Zeichen benötigt, als die Entropie angibt. Bei der obigen Zeichenkette beträgt der Wert der Entropie 3,28, die gesamte Folge mit 25 Zeichenvorkommen benötigt also mindestens 82,12 Bits.

Die Entropie ist ein wichtiges Konzept im Kontext von Codierungen und Nachrichtenübertragungen. Ins Leben gerufen hat sie Claude Elwood Shannon (Abb. 7.13), ein Amerikaner, der mit seiner bahnbrechenden Schrift *A Mathematical Theory of Communication* bereits 1948 die Informationstheorie ins Leben rief; er hat übrigens in Mathematik promoviert, doch dabei die Biologie im Blick gehabt: Seine Doktorarbeit beschäftigte sich mit theoretischer Genetik.

Die Entropie ist also die Richtschnur dafür, ob ein Code heiße Luft enthält oder nicht. Der Fachterminus für überflüssige Bits ist Redundanz und man nennt einen Code, der mehr Bits verwendet, als effektiv notwendig wären, redundant. Je mehr Bits „verschwendet“ werden, desto höher die Redundanz. Ein Zahlencode, bei dem die Anzahl der Einsen die Wertigkeit einer Zahl repräsentiert – wie bei einer Strichliste –, ist hochredundant, während der Dualzahlencode keine Redundanz aufweist. Es gibt Situationen, in denen man Redundanz möglichst vermeiden möchte, z. B. um große Datenmengen zu speichern; das schauen wir uns als Nächstes an. Es gibt aber auch Situationen, in denen man mit Absicht die Redundanz erhöht, dazu kommen wir später noch.



Abb. 7.13 Claude Edwood Shannon, der Begründer der Informationstheorie. (© Camille Büsser)

Komprimierung: Wie man aus Daten die Luft rauslässt

Bei der Datenkomprimierung geht es darum, die Luft rauszulassen: Hier sollen Daten verdichtet, also die Länge einer 0-1-Sequenz reduziert werden. Warum ist das überhaupt nötig, warum konzipiert man nicht von vorneher ein kompakte Codes? Nun, zum einen ist die Kompaktheit eines Codes nur ein Aspekt, und die Geschwindigkeit, mit der er verarbeitet werden kann, ist ein weiterer, häufig widersprechender Gesichtspunkt. Zum anderen lässt sich ja nicht vorhersehen, welche Zeichenfolgen bzw. Daten mit diesem Code erzeugt werden. Selbst wenn wir einen für die deutsche Sprache optimal zugeschnittenen Buchstabencode einsetzen: Jeder Text ist anders und erforderte für eine minimale Codierung einen individuell zugeschnittenen Code. Aus diesem Grund gibt es eine Vielzahl von Komprimierungsverfahren, die Zeichenfolgen umcodieren und so die Darstellung verkürzen. Ein einfaches Verfahren, an dem sich die Idee gut illustrieren lässt, ist die Lauflängencodierung (Bsp. 7.8). Sie fasst direkt aufeinander folgende Wiederholungen von Zeichen zusammen und liefert z. B. bei großflächigen, monotonen Bildern sehr gute Resultate.

Beispiel 7.8: Wiederholungen zusammenfassen

Die Idee der sogenannten Lauflängencodierung ist sehr einfach: Wiederholungen des gleichen Zeichens werden zusammengefasst in einem Paar, das die Anzahl der Wiederholungen und das Zeichen selbst darstellt. In Abb. 7.14 sehen Sie, wie eine Sequenz von 16 Zeichen statt mit regulär 16 Bytes durch 8 Bytes dargestellt werden kann (die einzelnen Zeichen werden gemäß Unicode repräsentiert). Die Lauflängencodierung lässt sich noch mit anderen Verfahren kombinieren, die sich wiederholende Musterketten identifizieren.

Die Lauflängencodierung ist ein schnell durchführbares Verfahren, doch in der Regel ist die Datenkomprimierung mit Aufwand verbunden. Man setzt sie daher sehr gezielt ein, vor allem dann, wenn große Datenmengen anfallen und der Speicherbedarf oder die Übertragungsgeschwindigkeit im Vordergrund steht. Beispielsweise lässt sich tatsächlich für einen gegebenen Text der optimale Buchstabencode ermitteln – analog zu den Überlegungen zur Entropie. Allerdings müssen wir dafür den Text im Vorfeld analysieren und eine Datenstruktur aufbauen, wie es in Abb. 7.15 unten zu sehen ist. Die Datenstruktur hat die Form eines Baums und wird wie folgt benutzt: Ausgehend von der Wurzel folgen wir den Pfeilen bzw. Kanten, bis wir beim gewünschten Buchstaben angelangt sind; die an den gewählten Kanten angeschriebenen Dualziffern hängen wir hintereinander und erhalten so das

Weg vorgibt. Im Gegensatz zu einer herkömmlichen Unicode-Darstellung lässt sich die Datenmenge mit diesem speziell auf den vorliegenden Text zugeschnittenen Code um mehr als 50 % reduzieren.

Diese Art der Darstellung nennt sich Huffman-Code, die Datenstruktur – Bsp. 7.9 erläutert ihren Aufbau – wird demgemäß als Huffman-Baum bezeichnet. Das Verfahren ist nicht auf Buchstaben beschränkt, sondern kann beispielsweise auch auf Wörter angewendet werden. Dazu nehmen wir die Wörter als Einheiten, zählen ihre Häufigkeiten und weisen dann Wörtern Bitfolgen zu – so lassen sich noch höhere Komprimierungsraten erzielen. Darüber hinaus kann das Huffman-Verfahren zur Komprimierung beliebiger Binärdaten eingesetzt werden, indem wir nämlich Bytes oder Byteblöcke als elementare Einheiten betrachten, die separat zu codieren sind. So oder so nehmen Codierung und Decodierung Zeit in Anspruch, insbesondere wenn sie noch mit weiteren Komprimierungsverfahren, z. B. der Lauflängencodierung, kombiniert werden.

Beispiel 7.9: Ein optimaler Code für Buchstaben

Um einen Huffman-Baum für die Buchstabenfolge „DU BIST MEIN ICH BIN DEIN“ aufzubauen, zählen wir zunächst für jeden Buchstaben einschließlich des Leerzeichens dessen Vorkommen im Text. Für jeden Buchstaben erstellen wir einen unverbundenen Knoten und schreiben diesen entsprechend mit der Häufigkeit an (siehe Abb. 7.16 oben). Pro vorkommendem Zeichen erhalten wir also einen Baum, der zu Beginn nur aus einem Wurzelknoten besteht. Nun verbinden wir die Bäume schrittweise, bis schließlich alle in einem einzigen Baum vereinigt sind. In jedem Schritt wählen wir die zwei Bäume aus, deren Wurzeln mit den kleinsten Häufigkeiten angeschrieben sind. Im ersten Schritt könnten dies z. B. die C und H zugeordneten Bäume sein. Jetzt verbinden wir die beiden zu C und H gehörenden Knoten zu einem Baum, indem wir einen neuen Wurzelknoten einführen und die zwei vorherigen Wurzelknoten dort anhängen. Zudem summieren wir die Häufigkeiten und vermerken das Gesamtvorkommen in der neuen Wurzel, in diesem Fall 2. Diese Schritte wiederholen wir, bis es nur noch einen Baum gibt. Abb. 7.16 zeigt den Entstehungsprozess auf, der Übersichtlichkeit halber werden mehrere Schritte pro Zeile dargestellt.

Den vollständigen Huffman-Baum erhalten wir nun, indem wir pro Knoten die abgehenden Kanten jeweils mit 0 und 1 anschreiben und die Ausgangsknoten mit den entsprechenden Buchstaben markieren. Dieser Baum muss zusammen mit den komprimierten Daten abgespeichert werden, ansonsten können sie nicht decodiert werden. Übrigens ist der so konstruierte Huffman-Code im Gegensatz zum Morse-Code präfixfrei, d. h., ein Codewort ist nie der Anfang eines anderen; deswegen braucht man auch keine Pause oder spezielle Markierung zwischen den Codewörtern. Es lässt sich zudem zeigen, dass die mittlere Codewortlänge optimal ist und die Entropie höchstens um ein Bit übersteigt.

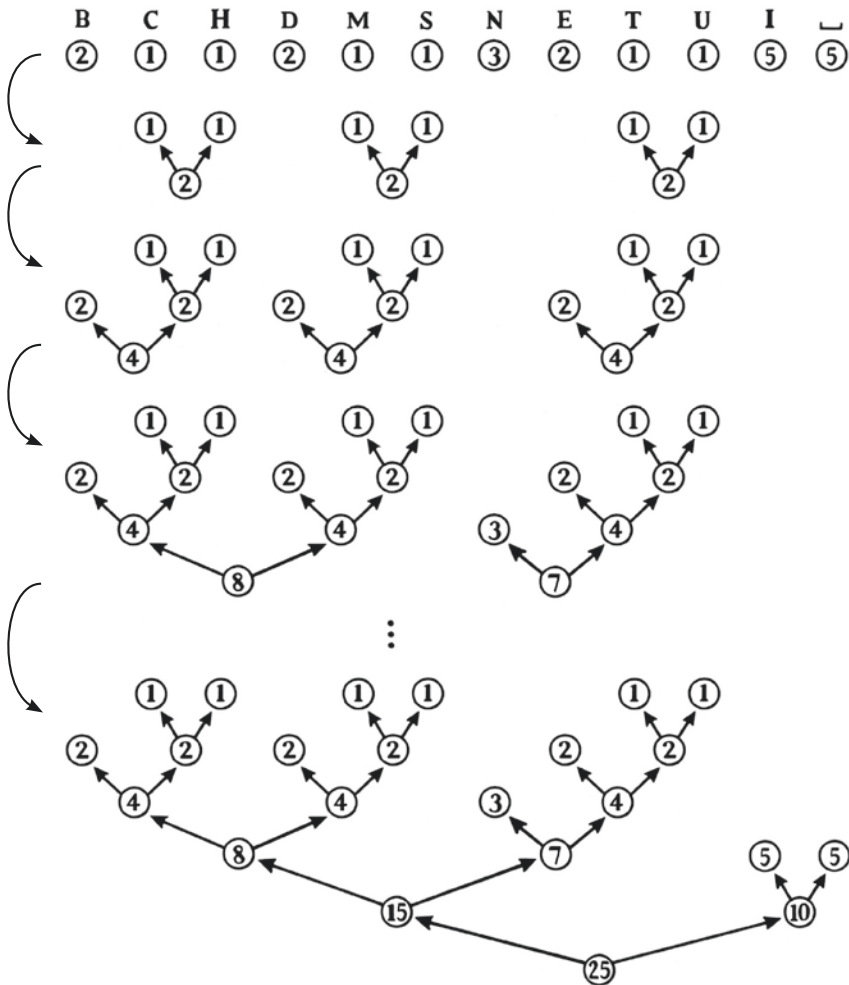


Abb. 7.16 Wie der Huffman-Code konstruiert wird. (© Susanne Staubli, Eckart Zitzler)

Die beiden soeben erläuterten Verfahren – es gibt natürlich noch viele weitere – sind Beispiele für verlustfreie Komprimierungsmethoden: Die Informationen bleiben erhalten, obwohl die Datenmenge verkleinert wurde; die Originaldaten lassen sich bei der Decodierung vollständig rekonstruieren. Häufig ist es jedoch nicht möglich, die Datenmenge zu reduzieren, ohne auch Informationen zu verlieren. Oder es ist nicht nötig, die Originaldaten vollständig zu erhalten. Bei Bildern z. B. müssen wir nicht die für einen Posterdruck notwendige Auflösung erhalten, wenn es nur um ein Foto auf einer Webseite geht. Indem wir die Rasterung vergrößern und Pixel durch



Abb. 7.17 Ein Bildausschnitt im Original (links) und in einer stark komprimierten Version mit Kompressionsartefakten (rechts). (© Eckart Zitzler)

Mittelung der Grau- bzw. Farbwerte zu größeren Bildpunkten zusammenfassen, können wir die Auflösungen von Bildern reduzieren. Klar ist: Da geht etwas verloren. Gnauso, wie wen wir in einem Satz in jedem grösseren Wort einen Buchstaben weglassen würden. Entscheidend ist: Der Satz ist trotzdem lesbar.

Gerade bei Fotos oder Audioaufnahmen wird mit unserer Wahrnehmungsgrenze gespielt, viele Reduktionen können wir gar nicht sehen oder hören. Bei Videos macht man sich das zunutze, indem sie nicht als Folge von Einzelbildern gespeichert, sondern nur die zentralen Bildinformationen codiert werden, die sich zum Vorgängerbild ändern. Erst wenn wir einen Bildausschnitt vergrößern, eine Musikkwiedergabe verstärken oder das Abspielen eines Films verlangsamen, bemerken wir den Informationsverlust. Abb. 7.17 zeigt Ihnen ein Beispiel für Artefakte, die bei der Bildkomprimierung entstehen können. Natürlich können wir die verlustbehaftete Komprimierung ins Extrem treiben und eine Landschaftsaufnahme auf ein Bit herunterkomprimieren, das dann nicht mehr aussagt als „Tag“ oder „Nacht“. Viel Sinn macht das nicht, es hängt vielmehr von der konkreten Anwendung ab, in welchem Maß Qualitätseinbußen verkraftbar sind. Und weil man eigentlich immer das Maximum herausholen möchte, gibt es sehr viele, auf spezifische Situationen zugeschnittene Komprimierungsmethoden.

So weit, so gut. Wenden wir uns jetzt der anderen Richtung zu: Anstatt Luft herauszulassen, werden wir nun Daten aufblasen.

Steganografie: Luft hinzufügen

Redundanz kann dann von Vorteil sein, wenn Teile eines Datenblocks verloren gehen können und die codierte Information trotzdem erhalten bleiben soll. Wir hatten in Kap. 5 ein Beispiel im Zusammenhang mit Festplatten

kennengelernt, bei dem pro Byte ein zusätzliches Bit für die Parität gespeichert wurde; wird ein Bit fehlerhaft abgelegt, so kann dies über das Paritätsbit festgestellt und das Originaldatum rekonstruiert werden. Man spricht in diesem Zusammenhang von fehlererkennenden bzw. fehlerkorrigierenden Codes. Dabei werden die Originaldaten mit Zusatzinformationen angereichert (siehe Anm. 7.3).

Anmerkung 7.3: Codes, die Fehler aufspüren

Fehlererkennende bzw. fehlerkorrigierende Codes zielen darauf ab, Übertragungs- oder Speicherungsfehler erkennen und evtl. sogar korrigieren zu können. Die einfachste Variante ist, die Daten mehrfach zu übertragen. Wenn wir einen Datenblock dreifach übertragen, dann können wir Veränderungen erkennen, selbst wenn zwei Kopien fehlerhaft sind. Korrekturen sind möglich, wenn nur eine der Kopien fehlerhaft ist; wir nehmen an jeder Bitposition jenen Wert, den die Mehrheit der Kopien enthält. Eine andere Variante ist, die Codewörter so zu wählen, dass sich diese genügend unterscheiden und Bitfehler nicht ins Gewicht fallen. Viele fehlererkennende Codes arbeiten mit mathematischen Formeln, bei denen aus den Daten ein Zahlenwert berechnet wird, der dann an die Daten angehängt wird. Zum Beispiel könnten wir die Quersumme einer Ziffernfolge nehmen und die letzte Ziffer der Quersumme anhängen; so ließen sich gewisse Fehler erkennen.

Ein anderer Grund, Daten heiße Luft zuzuführen, ist, Daten in Daten zu verstecken. Damit beschäftigt sich die Steganografie, die Wissenschaft von der verborgenen Speicherung oder Übermittlung von Informationen. Der Trick ist: Wir fügen eine Menge irrelevantes Material zu den geheim zu haltenden Daten hinzu, sodass dieses Material wie das Interessante wirkt und von den eigentlichen Informationen ablenkt. Früher nutzte man Geheimtinte, z. B. Zitronensaft, um in einem belanglosen Brief Nachrichten zu verstecken. Oder man füllte einen Text mit sinnlosem Text, und die eigentliche Nachricht ergab sich aus den Anfangsbuchstaben des Tarntextes. Mit der Computertechnologie haben sich die Möglichkeiten enorm erweitert. In jeglicher Art von Daten können andere Daten eingebettet werden, besonders leicht lassen sich Texte in Bild- oder Musikdaten verbergen. Die Idee besteht darin, die den Bild- bzw. Abtastpunkten zugeordneten Zahlenwerte leicht zu verändern, ohne dass es sichtbar oder hörbar wäre. Bsp. 7.10 erklärt im Detail, wie das geht.

Beispiel 7.10: Daten in einem Bild verstecken

Werfen Sie einen Blick auf Abb. 7.18: Wir wollen den Text „DU BIST MEIN...“ in einem Rasterbild verstecken. Wie geht das? Nehmen wir an, jedes Pixel ist

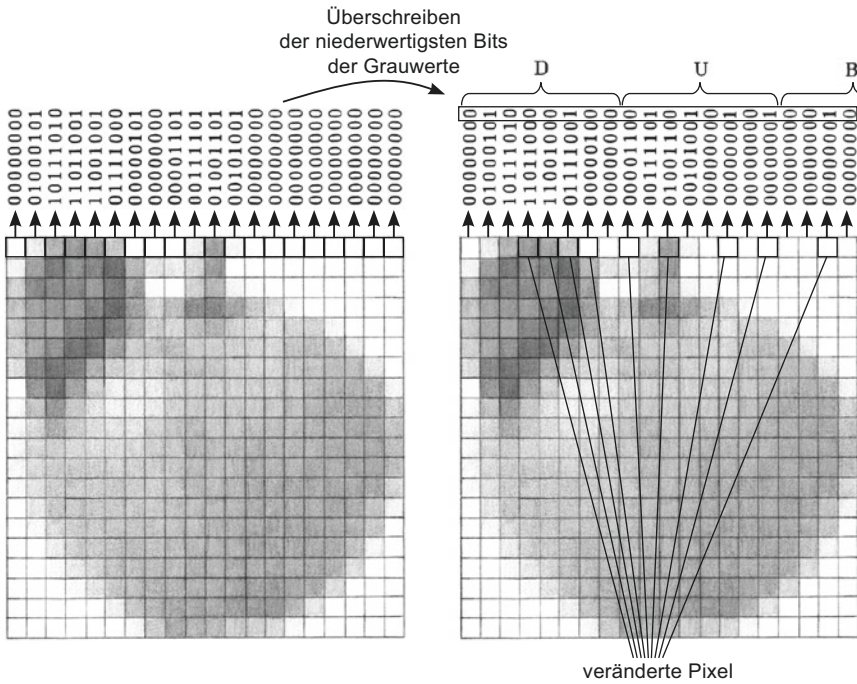


Abb. 7.18 Wie man einen Text in einem Graustufenbild verstecken kann. (© Susanne Staubli, Eckart Zitzler)

durch einen 8-Bit-Graustufenwert repräsentiert. Dann ist es an sich nicht wahrnehmbar, ob das letzte Bit im Graustufenbyte gesetzt ist oder nicht; der Grauwert unterscheidet sich ja nur um eins. Und das nutzen wir aus. Wir verwenden jetzt das niederwertigste Bit, um anderweitige Information zu codieren. Um also einen mit dem Unicode dargestellten Text unterzubringen, benötigen wir pro Buchstabe mindestens acht Pixel. Es braucht in diesem Fall mindestens die achtfache Menge an Bilddaten, um den eigentlichen Text zu speichern. Doch wenn wir es geschickt anstellen, z. B. die Reihenfolge der Pixel noch beliebig festlegen, dann wird es für jemand anderen schwer sein, die verborgene Information auszulesen.

Die Steganografie wird auch verwendet, um Medieninhalte mit Wasserzeichen zu versehen und so beim Kauf eines E-Books oder Digitalfotos die gelieferte Ware eindeutig zu markieren. Digitale Wasserzeichen können sichtbar oder unsichtbar sein und ähneln jenen, wie wir sie von Geldscheinen kennen. Sie sind zumeist darauf ausgelegt, dass sie nicht oder nur schwer entfernt werden können.

Genetische Datenakrobatik

Schließen wir das Thema der heißen Luft ab, indem wir nochmals einen Blick auf die DNA werfen. Wie Proteinbaupläne als A-C-G-T-Sequenzen durch Gene genannte Abschnitte auf der DNA codiert werden können, hatten wir bereits gesehen; der genetische Code definiert die Zuordnungsvorschrift von Nucleotiden zu Aminosäuren. Interessant ist jedoch auch die Verteilung und Platzierung der Gene auf der DNA, denn hier lassen sich Phänomene beobachten, die stark an die Methoden der Datenkomprimierung erinnern. Nicht nur, dass Gene sich teilweise überlagern können. Zudem werden die Abschriften noch manipuliert, sodass jeweils Ausschnitte von Abschriften zu verschiedenen Proteinbauplänen zusammengesetzt werden. Das heißt, ein und dasselbe Gen kann verschiedene Proteine hervorbringen. Bsp. 7.11 vermittelt Ihnen einen Einblick in die genetische Datenakrobatik.

Beispiel 7.11: Wie Gene verdichtet sind

Es gibt verschiedene Mechanismen, die die Natur nutzt, um die Daten auf der DNA zu komprimieren und den Platz optimal zu nutzen. Eine Möglichkeit besteht darin, Gene zu überlagern, wie es in Abb. 7.19 oben gezeigt ist; das lässt sich häufig bei Viren beobachten. Eine andere Möglichkeit besteht darin, aus ein und demselben Gen verschiedene Proteine zu produzieren (Abb. 7.19 unten). Das geschieht, indem die Abschrift – bevor sie vom Ribosom genutzt wird, um ein Protein herzustellen – noch manipuliert wird und Teile herausgeschnitten werden. Diesen Vorgang nennt man Spleißen. Und die Abschrift kann verschiedentlich zerschnitten werden mit dem Resultat, dass unterschiedliche Proteine entstehen – das nennt sich dann alternatives Spleißen. Man nimmt an, dass im menschlichen Genom über ein Drittel der Gene dem alternativen Spleißen unterliegen.

Dass Gene sich überlagern oder gleich mehrere Proteine codieren können, heißt nicht notwendigerweise, dass die DNA mit Genen vollgestopft ist. Bei Bakterien wie der Kolibakterie ist das häufig der Fall, die Daten können auf der DNA äußerst kompakt gespeichert sein. Bei uns Menschen hingegen ist nur ein sehr kleiner Anteil der DNA, weniger als 2 %, durch Proteinbaupläne belegt. Die gesamte DNA einer unserer Zellen ist zwar um die 700-mal länger als die einer Kolibakterie, aber wir besitzen nur ungefähr sechsmal mehr Gene als dieser Einzeller. Es kommt noch hinzu, dass bei uns Menschen die DNA-Stränge in Form von Chromosomen verdoppelt sind, wir quasi zwei DNA-Sätze besitzen – einen von der Mutter und einen vom Vater. Und schließlich ist das DNA-Molekül als Doppelhelix per se redun-

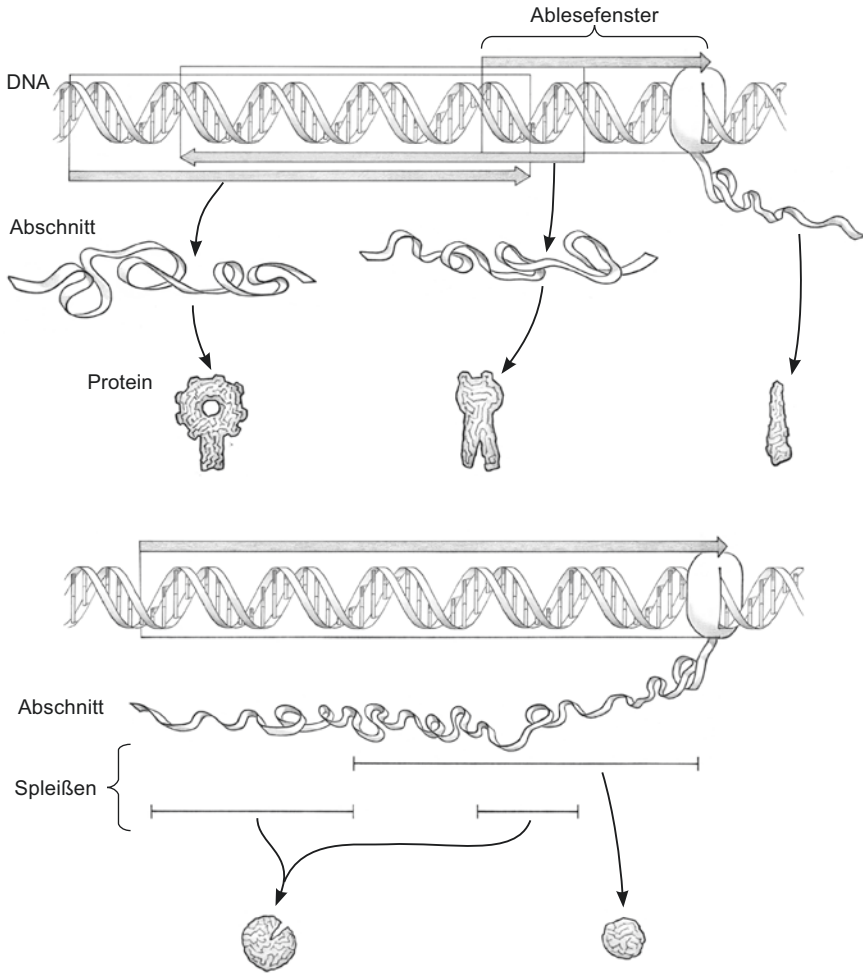


Abb. 7.19 Möglichkeiten der Datenkomprimierung auf der DNA. (© Susanne Staubli, Eckart Zitzler)

dant, denn die gleiche Information wird zweimal dargestellt. In unserem Genom scheint also viel heiße Luft enthalten zu sein. Die ist allerdings nicht einfach überflüssig, sondern hat verschiedene Funktionen und gibt den Forschern viele spannende Fragen auf. Sie dient u. a. dazu, Beschädigungen der DNA zu reparieren, d. h. durch Umwelteinflüsse oder Kopiervorgänge verursachte Fehler im Erbttext zu erkennen und zu beheben. Da gibt es äußerst vielfältige Korrekturmechanismen in den Zellen, denn fehlerhafte DNA kann in multizellulären Organismen äußerst problematisch werden.

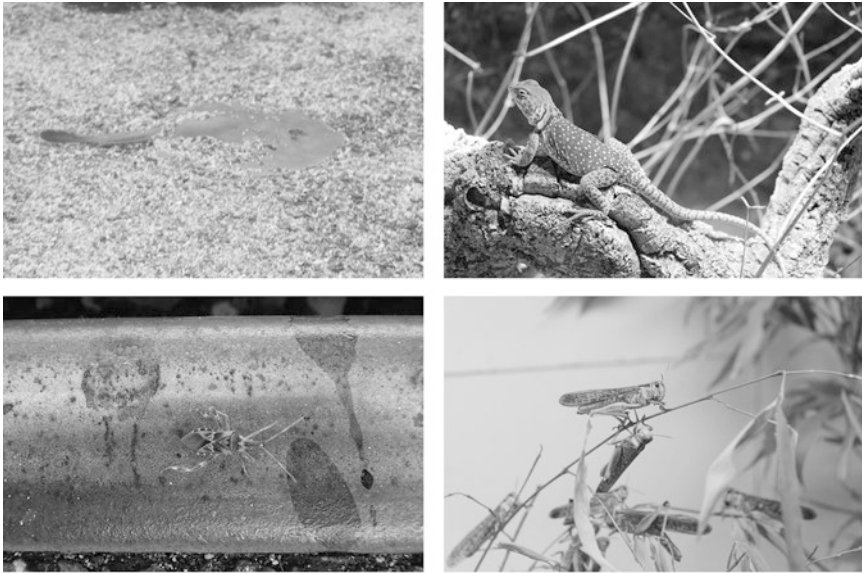


Abb. 7.20 Biologische Steganografie: Tarnung durch Verstecken und durch Vorgaukeln. (© Judith Büsser (oben, rechts unten), Eckart Zitzler (links unten))

Übrigens gibt es auch die Steganografie auf der DNA-Ebene: Viren verstecken und betten ihre DNA in die DNA ihres Wirts ein und nutzen so die ganze Produktionsmaschinerie der Wirtszelle aus. Das ist eine Art von Tarnung, genauso wie es in der Pflanzen- und Tierwelt gang und gäbe ist, Informationen zu verstecken oder gezielt andere Informationen zu suggerieren (Abb. 7.20).

Geheimniskrämereien

Im letzten Teil dieses Kapitels beschäftigen wir uns mit der Kryptografie, bei der es um die Entwicklung, die Analyse und möglicherweise das Brechen von Geheimcodes geht, aber auch um das Absichern von Daten gegen unerwünschte Manipulationen. Wann ist ein Code geheim? Nun, wenn nur wir und unsere Kommunikationspartner die konkrete Zeichenzuordnung kennen, während anderen dieses Wissen verwehrt ist. Weil der Mensch aber von Natur aus neugierig und Geheimhaltung schwierig ist, hat man sich Methoden überlegt, um andere entweder in die Irre zu führen – das versucht die Steganografie – oder ihnen den Zugriff per se zu verwehren – dieses Ziel verfolgt die Kryptografie. Während Erstere auf das Prinzip des

Versteckens setzt, so als würden wir einen kleinen Notizzettel in einem riesigen Papierstapel platzieren, geht die Kryptografie einen Schritt weiter und zielt darauf ab, schützenswerte Daten unzugänglich zu machen, so als wären sie in einem Tresor verschlossen. Kryptografische Verfahren können beispielsweise einen Text in einen unverständlichen, auf den ersten Blick sinnlosen Text übersetzen; das nennt man eine Chiffre. Wenn anderen das Wissen über die Chiffre fehlt, dann können sie – so die Hoffnung – nicht die originalen Daten und somit die codierte Information entschlüsseln.

Alter Wein in neuen Schläuchen: Verschlüsseln mit Schlüssel

Geheimcodes und Chiffren haben natürlich eine lange Tradition, und die Anfänge der Kryptografie reichen weit in die Geschichte der Menschheit zurück. Die Griechen beispielsweise nutzten ab dem 5. Jahrhundert einen Verschlüsselungsstab, Skytale genannt, mit dem die Buchstaben einer Nachricht auf einfache Weise neu angeordnet und zu einer scheinbar unsinnigen Zeichenfolge zusammengesetzt werden können; mit dem gleichen Mechanismus lassen sich verschlüsselte Texte wieder in den Originaltext zurückverwandeln. Auch Julius Cäsar soll eine Verschlüsselungsmethode eingesetzt haben, bei der Buchstaben nach einem fixen Schema durch andere ersetzt werden. Diese beiden altertümlichen Chiffriermethoden – Bsp. 7.12 erläutert die Details – illustrieren bereits die zwei Prinzipien, die bei der Verschlüsselung von Nachrichten zum Einsatz kommen: Transposition und Substitution. Bei der Transposition werden die Buchstaben des Originaltextes neu angeordnet, d. h., die Reihenfolge wird verändert, während bei der Substitution Buchstaben durch andere Zeichen ersetzt werden, ohne dass dadurch die Reihenfolge tangiert ist. Häufig werden diese zwei Prinzipien kombiniert und zudem noch mehrfach angewendet; und das kann dann recht kompliziert werden, wie wir später sehen werden.

Beispiel 7.12: Alte Chiffren

Die Idee der Skytale ist einfach: Der Geheimtext wird auf ein Pergament- oder Lederband vertikal aufgetragen, anschließend wird das Band um einen mit Kanten versehenen Holzstab mit vordefiniertem Durchmesser, die Skytale, gewickelt (siehe Abb. 7.21). Der verschlüsselte Text ergibt sich, indem die aufgewickelten Buchstaben nun horizontal vom Stab abgelesen werden. Hier wird also die Reihenfolge der Buchstaben verändert, die Originalbuchstaben bleiben bestehen.

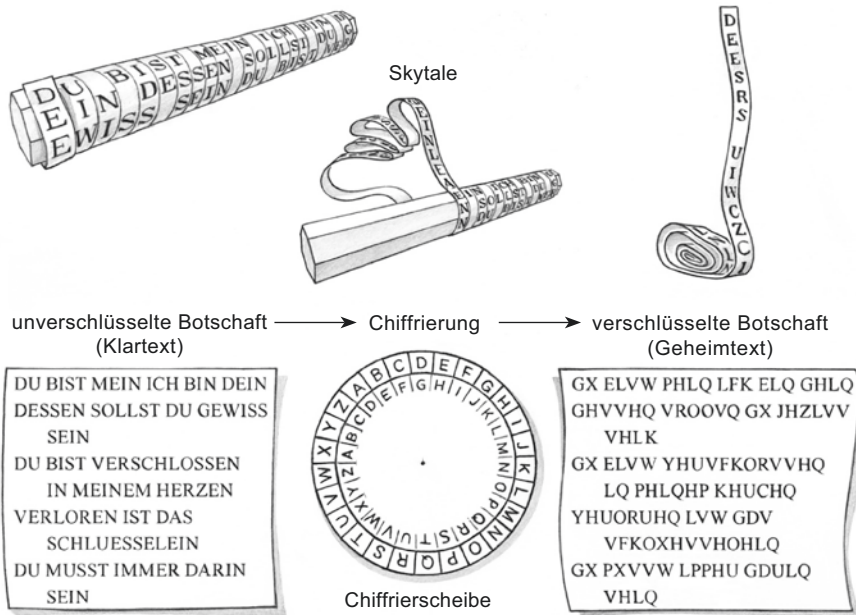


Abb. 7.21 Alte Chiffriermethoden: Die Skytale und die Chiffrierscheibe. (© Susanne Staubli, Eckart Zitzler)

Eine andere Art von Chiffre wird Julius Cäsar zugeschrieben, bei dem jeder Buchstabe ersetzt wird durch den überüberrnächsten Nachfolger im Alphabet – dabei ergibt sich quasi eine Verschiebung des Gesamtalphabets, die nicht nur drei Positionen, sondern beliebige Distanzen umfassen kann. Im 15. Jahrhundert wurde dieses Prinzip verallgemeinert mittels Chiffrierscheiben, bei denen die Buchstaben des Alphabets auf andere Zeichen, z. B. wiederum Buchstaben oder willkürliche Symbole, abgebildet werden (siehe Abb. 7.21).

Die Skytale und Cäsars Chiffriermethode können vertrauliche Informationen vor Dritten nur schützen, wenn die zugrunde liegenden Mechanismen vor Unbefugten geheim gehalten werden können. Mit der Zeit stellte sich heraus, dass dies unrealistisch war und es einer zusätzlichen Stufe der Sicherheit bedurfte. Schon bald kam die Idee auf, die Verschlüsselungsmethoden zu flexibilisieren und ihnen Zusatzinformationen an die Seite zu stellen, die als Zugangsschlüssel fungieren. Die Chiffrierscheibe ist z. B. eine Generalisierung der Cäsar-Chiffre: Anstatt das Alphabet jeweils fix um drei Positionen zu verschieben, können die Kommunikationspartner auch eine beliebige Verschiebung, z. B. um fünf oder zehn Positionen, vereinbaren. Die Verschiebungsinformationen,

die beide Kommunikationspartner im Vorfeld austauschen und geheim halten müssen, stellen den Schlüssel dar. Auf diese Art und Weise lassen sich wesentlich sicherere Verschlüsselungsmethoden wie die Vigenère-Chiffre konstruieren (siehe Bsp. 7.13). Der Vorteil an dieser Aufteilung in einen variablen und einen starren Anteil ist, dass nur der Schlüssel und nicht das gesamte Verfahren vor der Öffentlichkeit verborgen werden muss. Das ist ja bei einem regulären Haustürschloss auch so: Wir wissen, wie es funktioniert, aber ohne Schlüssel können wir es nicht öffnen.

Beispiel 7.13: Die Vigenère-Chiffre

Die Chiffrierscheibe hat den Nachteil, dass die Verschlüsselung leicht zu knacken ist: Einer der am häufigsten vorkommenden Buchstaben ist vermutlich das E, und schon haben wir ein paar wahrscheinliche Hypothesen für die Verschiebung. Schwieriger wird es, wenn gleich mehrere Scheiben parallel eingesetzt werden, jede mit einer spezifischen Einstellung. Wenn wir beispielsweise zwei Chiffrierscheiben einsetzen, dann werden die Buchstaben des Originaltextes abwechselnd mal mit der einen und mal mit der anderen Chiffrierscheibe codiert. Die Einstellungen der beiden Chiffrierscheiben lassen sich über zwei Buchstaben beschreiben, die die Verschiebung spezifizieren, d. h. durch welchen Buchstaben das A jeweils ersetzt wird (siehe Abb. 7.22). Die zwei Buchstaben stellen also den Schlüssel dar. Im Prinzip lassen sich auch beliebig viele Chiffrierscheiben verwenden, die durch ein entsprechendes langes Schlüsselwort repräsentiert werden. Diese Verschlüsselungsmethode stammt aus dem 16. Jahrhundert und nennt sich Vigenère-Chiffre.

Die simple Chiffrierscheibe lässt sich sogar zu einem Verschlüsselungsverfahren ausbauen, das nicht gebrochen werden kann. Indem wir für jede Position im Text zufällig eine individuelle Verschiebung auswählen, wird jeder Buchstabe in der Originalnachricht anders verschlüsselt – das entspricht einer Vigenère-Chiffre, bei der der Schlüssel mindestens genauso lang ist wie der zu verschlüsselnde Text. Verwenden wir diese Verschiebungsinformationen nur einmal, dann ist hundertprozentige Sicherheit garantiert. Das Verfahren ist als One-Time-Pad bekannt und wurde schon Ende des 19. Jahrhunderts beschrieben. Es lässt sich nicht nur

Klartext	D	U		B	I	S	T		M	E	I	N
Schlüsselfolge	B	Z		B	Z	B	Z		B	Z	B	Z
Geheimtext	E	T		C	H	T	S		N	D	J	M

Abb. 7.22 Die Vigenère-Chiffre mit einem Schlüssel der Länge 2. (© Susanne Staubli, Eckart Zitzler)

für die Verschlüsselung von Texten verwenden, sondern auch für visuelle Darstellungen umsetzen, wie Anm. 7.4 aufzeigt.

Anmerkung 7.4: Ein visuelles One-Time-Pad

Eine ganze andere Möglichkeit, Informationen zu verschlüsseln, bietet die visuelle Kryptografie. Die Idee ist hier folgende (Abb. 7.23): Ein zu verschlüsselndes Bild wird in zwei Bilder aufgeteilt, jedes für sich stellt einzig Rauschen dar und erlaubt keine Rückschlüsse auf die gespeicherte Information. Nur durch die Überlagerung beider Bilder wird die Geheimbotschaft sichtbar. Das Prinzip dahinter ist einfach: Jedes Pixel der Original-Bitmap wird durch einen 4-Pixel-Block ersetzt, bei dem nur zwei gegenüberliegende Pixel schwarz sind, die anderen beiden sind weiß. Ist nun das Original-Pixel weiß, so wird auf beiden Teilbildern der gleiche 4-Pixel-Block verwendet; ist das Original-Pixel schwarz, so wird auf einem Teilbild der komplementäre 4-Pixel-Block verwendet. Werden nun die Teilbilder übereinandergelegt, so ergeben sich dort schwarze 4-Pixel-Blöcke, wo im Originalbild schwarze Pixel waren.

Als One-Time-Pad lässt es sich wie folgt nutzen: Die eine Folie ist die verschlüsselte Nachricht, die andere ist der einmalige Schlüssel. Das lässt sich auch gut so handhaben, dass die zweite Folie im Voraus verteilt und die erste Folie dann gemäß dem Zielbild angefertigt wird.

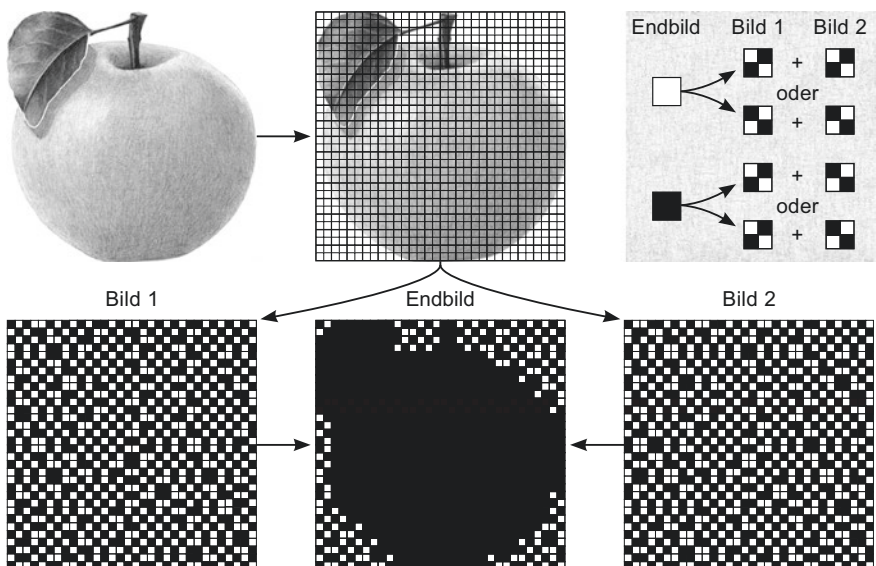


Abb. 7.23 Wie ein Bild in zwei Folien versteckt werden kann. (© Susanne Staubli, Eckart Zitzler)

So sicher das One-Time-Pad auch ist, sein Einsatzbereich ist beschränkt. Nicht nur, dass sich beide Kommunikationspartner zuerst treffen müssen oder absolut verlässliche Boten brauchen, um mehrere One-Time-Pads für später auszutauschen. Es kommt noch hinzu, dass die Schlüssellänge mindestens der Länge des Originaltextes entsprechen muss, was im alltäglichen Datenverkehr Limitierungen nach sich zieht. Das Schlüssel-Prinzip ist dann ideal umgesetzt, wenn der Schlüssel einerseits so klein ist, dass er noch handhabbar bleibt, er andererseits aber so essenziell für das Verfahren ist, dass die Dechiffrierung ohne ihn unmöglich oder zumindest extrem schwierig wird. Denn genau das ist die Idee: Zentrale Zusatzinformationen, die kurz darstellbar und einfach geheim auszutauschen sind, helfen, große Datenmengen effektiv zu verschlüsseln, ohne das Verschlüsselungsverfahren geheim halten zu müssen.

Moderne Chiffriermethoden, bei denen die Verschlüsselung von Computern vorgenommen wird, arbeiten auf der Basis von 0-1-Sequenzen und können so beliebige Daten verschlüsseln. Die Daten werden in mehreren Durchläufen durch Transpositionen und Substitutionen so stark verändert, dass es bei entsprechend großen Schlüsselräumen heutzutage nicht möglich ist, die Rücktransformation ohne Schlüsselkenntnis herbeizuführen. In Bsp. 7.14 können Sie nachlesen, wie so ein Verfahren arbeitet; es skizziert den *Data Encryption Standard* (DES), der 1977 offizieller Verschlüsselungsstandard für die US-Regierung und als erstes Verschlüsselungsverfahren des Computerzeitalters in allen Details veröffentlicht wurde.

Beispiel 7.14: Der Data Encryption Standard

DES verschlüsselt jeweils 64-Bit-Blöcke – die Originaldaten sind entsprechend zu portionieren – und operiert mit einem gleich großen Schlüssel, von dem allerdings nur 56 Bits effektiv gesetzt werden können, die restlichen 8 Bits sind Paritätsbits. Jeder Datenblock wird nun in einem mehrstufigen Verfahren mit dem Schlüssel vermischt und umgewandelt. Der Ablauf ist in Abb. 7.24 skizziert, dabei symbolisieren Kochtöpfe Transpositionen und Substitutionen, Rührbesen die Verknüpfung verschiedener 0-1-Sequenzen und Messer die Aufteilung einer Bitsequenz. Die nachfolgende Beschreibung soll Ihnen eine Idee des Vorgehens vermitteln, für Details verweise ich auf weiterführende Literatur (z. B. Beutelspacher, Neumann und Schwarzpaul 2010).

Als Erstes werden die Bits im zu verschlüsselnden Block neu angeordnet, anschließend wird der Block in zwei Hälften zerteilt. Des Weiteren wird aus dem Schlüssel durch Bitverschiebung und -ersetzung ein 48 Bit breiter Teilschlüssel generiert. Die rechte Hälfte des Datenblocks wird nun mit dem generierten Teilschlüssel mittels einer XOR-Operation verknüpft, die Bits werden gemäß vorgegebenen Tabellen substituiert, anschließend wieder umsortiert und zum Schluss mit der linken Hälfte des Datenblocks kombiniert. Das Ergebnis ist nun die neue rechte Hälfte, die vorherige rechte Hälfte ist die neue linke Hälfte und das Verfahren wird wiederholt, insgesamt 16-mal. Parallel

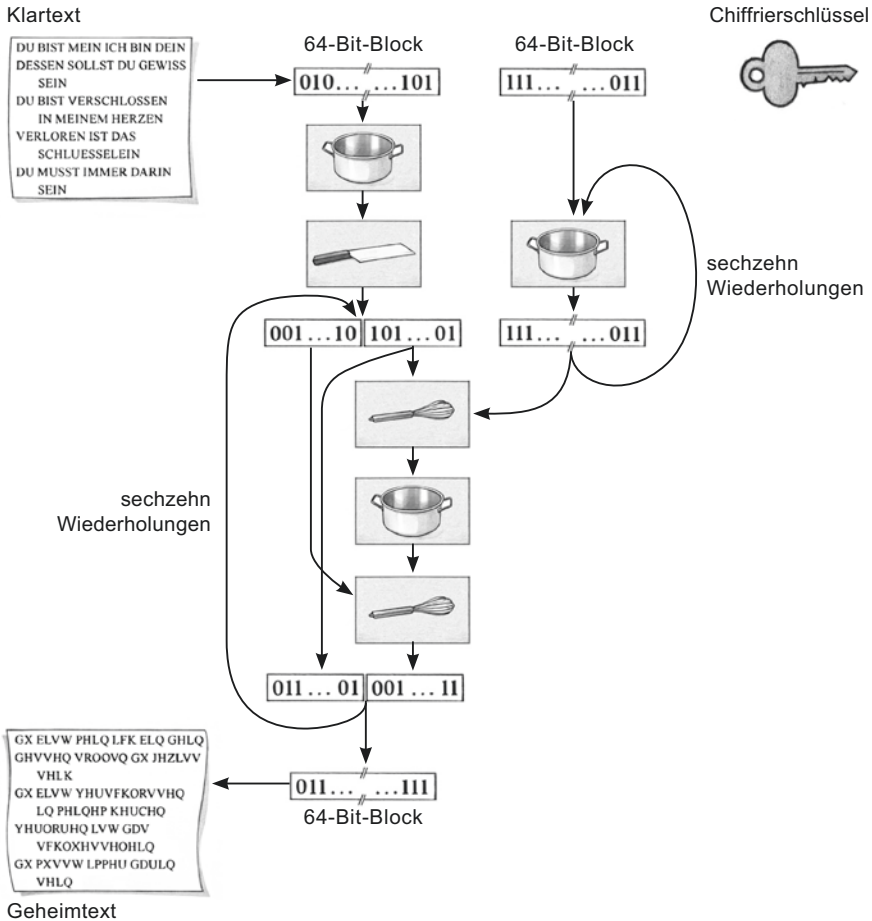


Abb. 7.24 Der Ablauf beim DES-Verschlüsselungsalgorithmus. (© Susanne Staubli, Eckart Zitzler)

dazu wird aus dem aktuellen Teilschlüssel jeweils ein neuer Teilschlüssel generiert, insgesamt auch 16-mal. Die Dechiffrierung läuft nach demselben Schema ab, nur dass die Teilschlüssel in umgekehrter Reihenfolge generiert werden.

Da DES auf einfachen Bitverknüpfungs- und -ersetzungsmethoden beruht, ist das Verfahren leicht in Hardware zu implementieren. Allerdings gilt der Originalalgorithmus heutzutage als nicht mehr sicher wegen der zu kurzen Schlüssellänge von 56 Bit. Durch eine dreifache Hintereinanderausführung mit zwei verschiedenen Schlüsseln lässt sich die Schlüssellänge verdoppeln und die Sicherheit erhöhen: Erst wird der Klartext mit dem ersten Schlüssel chiffriert, dann mittels des zweiten Schlüssels dechiffriert und schließlich nochmals mit dem ersten Schlüssel chiffriert. Dieses Verfahren nennt sich *Triple DES* (3DES).

Natürlich stellt sich die Frage: Wie sicher sind die kryptografischen Verfahren, wie sehr kann man sich auf sie verlassen? Je mehr die Kommunikationsmittel wie Funk und Telegrafie den Globus eroberten, desto mehr spielte auch die Kryptografie eine zentrale Rolle, z. B. während des Zweiten Weltkriegs. Die Deutschen setzten eine elektromechanische Chiffriermaschine namens Enigma ein, die auf komplizierte Art und Weise – mittels rotierender Chiffrierscheiben, die miteinander verbunden waren – den Originaltext durchschüttelte (Abb. 7.25). Die Enigma galt als unknackbar, es gab über 150 Billionen (Millionen Millionen Millionen) Möglichkeiten, einen Schlüssel zu wählen. Die Zahl ist so groß, dass selbst heutige Computer nicht in der Lage wären, alle möglichen Schlüssel auszuprobieren, um den Originaltext zu ermitteln. Die Anzahl der Verschlüsselungsmöglichkeiten sagt jedoch nichts aus über die Sicherheit des Verfahrens. Tatsächlich gelang es den Briten unter der Führung von Alan Turing (da haben wir ihn schon wieder), Enigma-verschlüsselte Funksprüche zu decodieren und somit den Alliierten entscheidende Vorteile zu verschaffen. Verschlüsselung ist auch immer ein Wettlauf, und zwar zwischen Jägern und Gejagten. Die Vigenère-Chiffre aus dem 16. Jahrhundert konnte über 300 Jahrhunderte nicht gebrochen werden, beim DES-Verfahren dauerte es etwas mehr als 20 Jahre. Der Nachfolger von DES, der *Advanced Encryption Standard* (AES), wurde 2001 eingeführt und bislang nicht gebrochen. Doch auch hier könnte sich das zu gegebener Zeit ändern, nämlich u. a. mit dem Quantencomputer.

Nichtsdestotrotz, wenn wir nicht gerade die Ewigkeit im Sinn haben, gelten die heutigen Verschlüsselungsverfahren als sehr sicher – insofern beide Kommunikationspartner einen gemeinsamen, geheimen Schlüssel besitzen. Gemeinsam heißt, der Schlüssel muss beiden bekannt sein. Doch wie

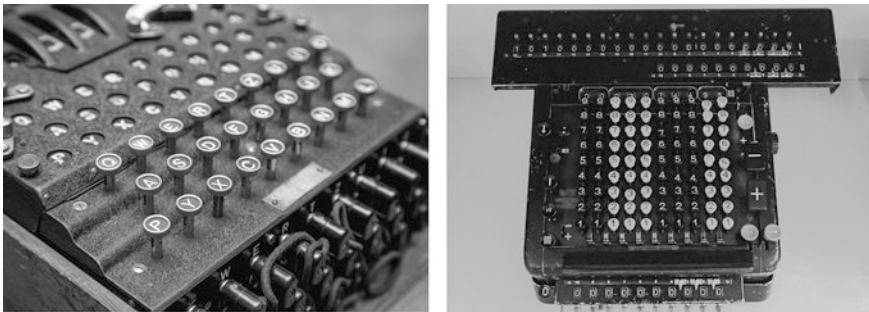


Abb. 7.25 Die Enigma-Chiffriermaschine aus dem 2. Weltkrieg (links), die äußerlich einer mechanischen Rechenmaschine (rechts) ähnelt. (© Boca/stock.adobe.com [links], Eckart Zitzler (rechts, mit freundlicher Genehmigung des Museums ENTER Solothurn))

tauschen sie diesen Schlüssel aus? Dafür müssen sie sich treffen, oder es braucht vertrauenswürdige, zuverlässige Boten oder Kanäle, die den Partnern exklusiv zur Verfügung stehen – im großen Stil, sei es nun für das Online-Banking oder den verschlüsselten Nachrichtenaustausch zwischen beliebigen Empfängern, ist das nicht praktikabel. Und tatsächlich wurde der Schlüsselaustausch mit dem Aufkommen von Computernetzen zum Problem, das man längere Zeit für nicht lösbar hielt.

Öffentlich Geheimnisse teilen

Die zwei Amerikaner Whitfield Diffie und Martin Hellman waren es schließlich, die Mitte der 1970er-Jahre eine Lösung präsentierten. Die Idee ist, dass beide Kommunikationspartner jeweils über eine geheime Zahl verfügen und aus diesen zwei Zahlen einen gemeinsamen Schlüssel berechnen. Zu diesem Zweck führen beide Partner parallel dieselbe Rechnung durch – der eine jedoch in umgekehrter Reihenfolge –, tauschen in der Mitte der Berechnung die Zwischenergebnisse aus und verfügen am Ende über dasselbe Resultat, das dann den gesuchten Schlüssel darstellt. Die Berechnung ist so ausgelegt, dass das Abfangen der Zwischenergebnisse nicht ausreicht, um den Schlüssel zu ermitteln; es braucht immer noch eine der zwei geheimen Zahlen. Das Prinzip lässt sich veranschaulichen an einem Beispiel, das zwar konstruiert und rein hypothetisch ist, aber zugleich die abstrakte Mathematik in anschauliche Bilder übersetzt. Wir benutzen den Duft als Analogie (siehe Bsp. 7.15).

Beispiel 7.15: Die Idee der Schlüsselgenerierung in Analogie zu Düften

Nehmen wir an, Alice und Bob wollen zusammen einen Duft kreieren, entweder indem sie ein Parfüm herstellen oder eine Grundsubstanz wie eine Suppe würzen. Jeder von ihnen hat geheime Duft- bzw. Gewürzessenzen, die sonst kein anderer kennt, nicht einmal gegenseitig wissen sie davon. Sie sollen nun zusammen einen Duft erstellen, sodass Dritte, z. B. Eva und Eveline, nicht in der Lage sind, den Duft zu reproduzieren. Wie soll das gehen?

Nun, zunächst einmal einigen sich Alice und Bob auf eine Grundsubstanz; sagen wir, Alice wählt sie aus und schickt einen Becher davon an Bob – Eva und Eveline können diese Grundsubstanz ggf. abfangen, d. h., die Übertragung ist nicht geheim. Jetzt mischen Alice und Bob, jeder für sich, ihre geheimen Essenzen in die gleiche Menge an Grundsubstanz. Anschließend tauschen sie die hergestellten duftenden Substanzen aus, Eva und Eveline haben wiederum die Möglichkeit, beide abzufangen. Und zum Schluss nehmen Alice und Bob die duftende Grundsubstanz vom Gegenüber und mischen sie ihren Geheimessenzen hinzu. Unter der Annahme, dass die Reihenfolge der Beimischung keine Rolle spielt, haben nun beide den gleichen Duft.

Doch was ist mit Eva und Eveline? Nehmen wir den schlimmsten Fall an, dass sie alles abgefangen haben. Sie könnten ja auch einfach Alices und Bobs Substanz nehmen und mischen. Aber halt: Dann hätten sie ein anderes Mischverhältnis und einen anderen Duft. Die einzige Möglichkeit besteht darin, dass sie für eine der duftenden Substanzen die Zutaten herausbekommen. Wenn die Duftmischung komplex genug ist, wird das sehr schwierig bis zu unmöglich sein. Das ist die Idee.

Entscheidend bei diesem Beispiel ist, dass die Produktion eines Duftgemisches einfach funktioniert, während die Umkehrung dieses Prozesses, d. h. die Zerlegung eines Geruchs in einzelne Duftessenzen, extrem aufwändig bis unmöglich ist. Diese Situation lässt sich auch in der Informatik erzeugen, nur dass hier anstatt mit Düften mit Zahlen hantiert und aus dem Mischen ein Berechnen wird. Es ist tatsächlich möglich, Rechenwege zu konstruieren, auf denen sich das Zurücklaufen als schwierig bis unmöglich erweist. Ein Verschlüsselungsverfahren ist dafür ein Beispiel, wenn der Schlüssel nicht bekannt ist. Vertraut ist uns natürlich eher das Gegenteil: Wir können ja Plus mit Minus, Mal mit Geteilt, Quadrieren mit Wurzelziehen und Potenzieren mit Logarithmieren rückgängig machen. Doch schauen wir uns zunächst einmal an, wie zwei Partner überhaupt über einen Kommunikationskanal einen gemeinsamen Schlüssel generieren, und blenden den Aspekt, dass Dritte mithören können, für den Moment aus. Bsp. 7.16 zeigt eine sehr einfache Möglichkeit auf: Eine vorgegebene Zahl wird mehrfach mit sich selbst multipliziert, die Häufigkeit ist bestimmt durch die zwei Geheimzahlen der Kommunikationspartner; das Resultat ist der gesuchte Schlüssel.

Beispiel 7.16: Schlüsselaustausch im Detail

Die Ausgangslage für die Berechnung bildet eine Zahl, über die sich Alice und Bob zu Beginn verständigen. Diese Zahl, im Folgenden mit g bezeichnet, entspricht der Grundsubstanz beim Duftbeispiel und kann beispielsweise von Alice vorgeschlagen (Schritt 1) und an Bob übermittelt werden (Schritt 2), wie Abb. 7.26 illustriert. Des Weiteren verfügen Alice und Bob jeweils über eine geheime Zahl, sagen wir a und b (Schritt 3). Nun multipliziert Alice die Zahl g genau a -mal, während Bob g insgesamt b -mal multipliziert (Schritt 4). Ihre Zwischenergebnisse tauschen sie aus (Schritt 5), und jeder potenziert das Zwischenergebnis des Gegenübers wiederum mit der eigenen Geheimzahl (Schritt 6). Das Endresultat ist schließlich auf beiden Seiten das gleiche, nämlich g hoch a mal b ; es stellt den gesuchten Schlüssel dar, der üblicherweise als Dualzahl dargestellt wird.

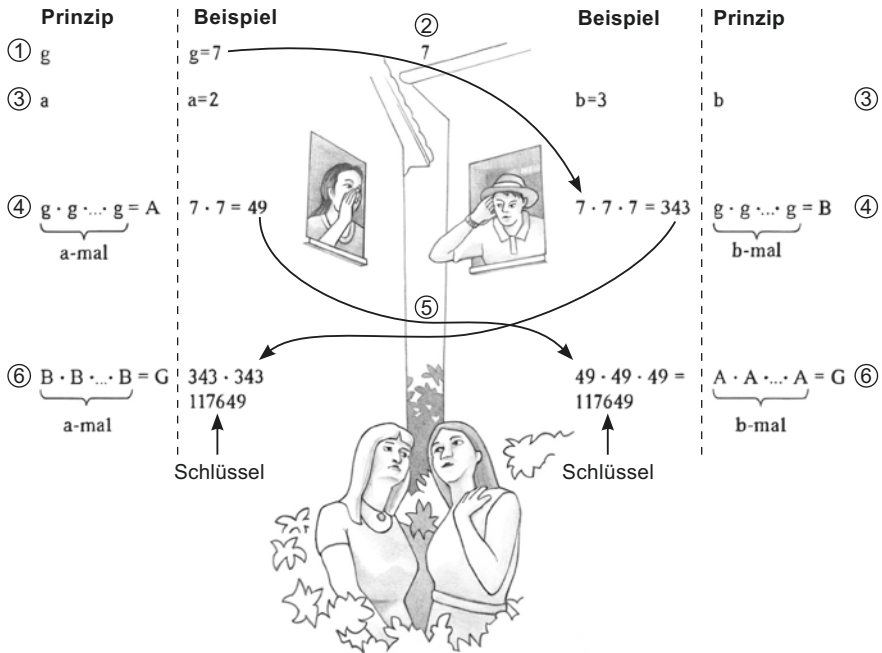


Abb. 7.26 Das Prinzip der Schlüsselgenerierung. (© Susanne Staubli, Eckart Zitzler)

Sie könnten an dieser Stelle zu Recht fragen, warum ausgerechnet diese Funktion (g hoch a mal b) zum Einsatz kommt und Alice nicht gleich einen selbst gewählten Schlüssel übermittelt. Denn falls Dritte, sagen wir Eva und Eveline, die Übertragung abhören, können sie auf einfachem Weg die Geheimzahlen und so auch den gemeinsamen Schlüssel ermitteln: Der Logarithmus stellt die gesuchte Umkehrfunktion dar und gibt an, wie häufig eine Zahl mit sich selbst multipliziert werden muss, um ein vorgegebenes Resultat zu erzielen. Die vorgestellte Berechnung lässt sich jedoch so abändern, dass Eva und Eveline keine Chance mehr haben. Der Trick besteht darin, eine sogenannte Einwegfunktion einzubauen, eine nicht umkehrbare Berechnungsvorschrift. Sie wandelt eine Zahl in eine andere Zahl um, sodass sich erstere aus letzterer nicht mehr ermitteln lässt. Einwegfunktionen können Sie sich als Geheimtüren in Wänden vorstellen, die von der einen Seite leicht zu finden, von der anderen Seite aber nicht sichtbar und schwer zu identifizieren sind. Deswegen heißen sie im Englischen auch *trap door functions*. Einwegfunktionen werden zumeist nach einem speziellen Schema zusammengebaut, bei dem Zahlen kurzerhand umbenannt werden.

Bsp. 7.17 erläutert ein Schema, das Ihnen aus dem Alltag bestens vertraut ist: die Uhrenarithmetik.

Beispiel 7.17: Die Uhrenfunktion oder wenn nur der Rest zählt

Die Uhrenarithmetik besagt nichts anderes, als dass das Zählen wie bei einer alten Turmuhr funktioniert: Nach Mitternacht beginnt es wieder von vorne. Vereinfacht dargestellt ist so ein Szenario in Abb. 7.27 links unten, in dem die Zahlen 0 bis 11 kreisförmig auf dem Ziffernblatt angeordnet sind. Letztlich wird dabei der gesamte Zahlenraum auf den Abschnitt von 0 bis 11 abgebildet, so als würden wir den Zahlenstrahl um das Ziffernblatt herumwickeln. Wenn es also 7 Uhr ist, dann ist es 20 Stunden später nicht 27 Uhr, sondern 3 Uhr bzw. $7 + 20 = 3$ in diesem Zählsystem. Diese Idee lässt sich nun verallgemeinern, indem wir die „Uhrengröße“ beliebig wählen können, z. B. mit 17 Unterteilungen bzw. den Zahlen von 0 bis 16 wie in Abb. 7.27 rechts. Das Ganze nennt sich dann Uhrenfunktion oder in der Fachsprache Modulo-Funktion: Ausgehend von der 0 wird der Zeiger so oft um eins weitergeschoben, wie es eine vorgegebene Zahl angibt; das Ergebnis ist die Zahl, auf die der Zeiger am Schluss zeigt. Bei einer Uhrengröße von 17 wird so aus einer 53 eine 2 und aus einer 66 eine 15.

Die Uhrenfunktion bietet ein gutes Fundament, um Einwegfunktionen zu konstruieren. Zum einen durchmischt die Modulo-Funktion eine Zahlenmenge, aus dem Ergebnis lässt sich nur schwer auf die Eingabe schließen. Zum anderen kann die Uhrenfunktion gut in Rechnungen eingebaut werden, ohne den Ablauf zu verändern. Beispielsweise spielt es keine Rolle, ob wir zwei Zahlen multiplizieren und anschließend die Uhrenfunktion anwenden oder ob wir erst die Uhrenfunktion auf beide Zahlen anwenden, die Ergebnisse multipli-

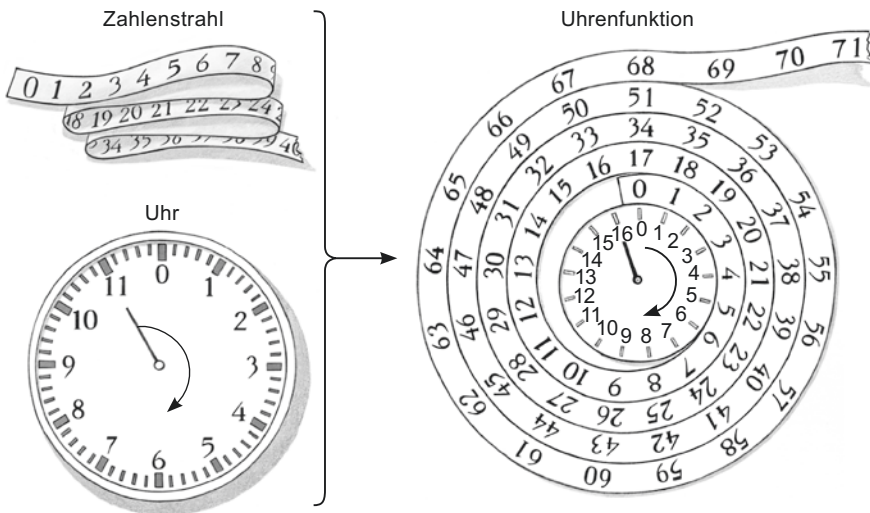


Abb. 7.27 Die Uhren- bzw. Modulo-Funktion. (© Susanne Staubli, Eckart Zitzler)

zieren und dann wiederum die Uhrenfunktion anwenden. Möchten Sie es selbst für eine Uhrengröße von 17 überprüfen? Wenn wir die zwei Zahlen 40 und 20 direkt miteinander multiplizieren und dann die Uhrenfunktion auf das Produkt 800 anwenden, erhalten wir 1 als Ergebnis; wenn wir stattdessen die zwei Zahlen erst durch die Uhrenfunktion schleusen, anschließend die zwei Resultate 6 und 3 miteinander multiplizieren und dann das Produkt 18 wieder mittels der Uhrenfunktion umwandeln, führt das ebenfalls zum Endergebnis 1.

Die Uhrenarithmetik ermöglicht es, den kritischen Moment abzusichern, wenn die Zwischenergebnisse ausgetauscht werden. Indem Alice und Bob vor der Übermittlung die Uhrenfunktion anwenden, realisieren sie eine Einwegfunktion und verschleiern die Zwischenergebnisse. Eva und Eveline können jetzt nicht mehr einfach über den Logarithmus ermitteln, wie häufig die Grundzahl mit sich selbst multipliziert wurde. Nur über einen extremen, praktisch nicht umsetzbaren Aufwand könnten sie theoretisch die Geheimzahlen ermitteln, doch dieses Quäntchen Unsicherheit ist in der Realität vernachlässigbar.

Der Einsatz der Uhrenfunktion erfordert allerdings weitere Anpassungen im Ablauf, damit die gemeinsame Schlüsselgenerierung funktioniert (siehe Abb. 7.28). Zum einen müssen Alice und Bob im Voraus die

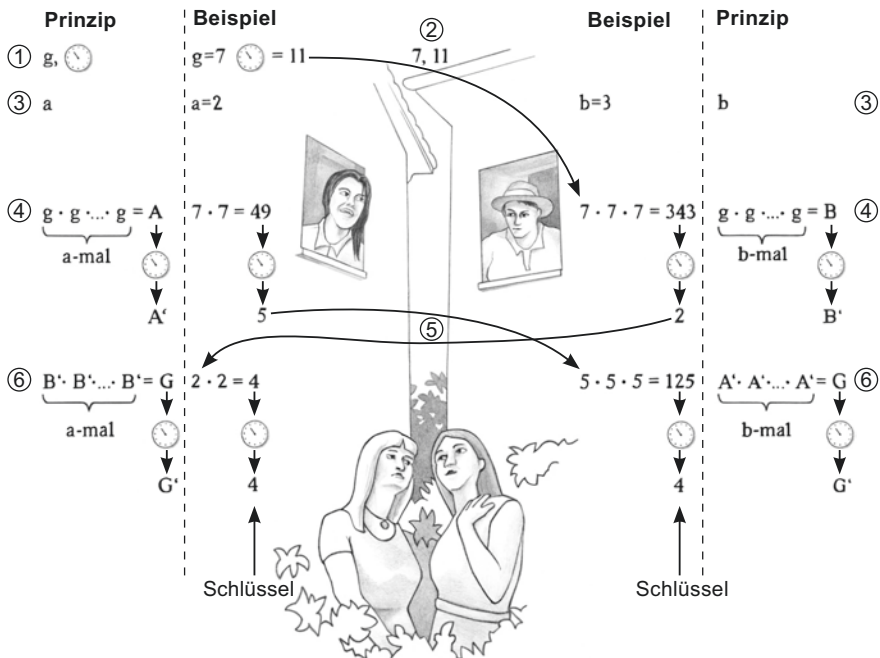


Abb. 7.28 Sichere Schlüsselgenerierung mittels Uhrenfunktion. (© Susanne Staubli, Eckart Zitzler)

Uhrengröße festlegen, die bei der Uhrenarithmetik zum Einsatz kommen soll. Zum anderen müssen beide das Schlussergebnis noch einmal mit der Uhrenfunktion umwandeln; nur so ist – aufgrund der besonderen Eigenschaften der Uhrenfunktion – gewährleistet, dass auf jeder Seite derselbe Schlüssel erzeugt wird. Im Anschluss kann dann ein bewährtes Verschlüsselungsverfahren eingesetzt werden, um den eigentlichen Datenaustausch sicher abzuwickeln.

Wenn der eine nicht ohne den anderen kann: zweiteilige Schlüssel

An sich haben wir nun das Rüstzeug, um sicher in einem Computernetzwerk zu kommunizieren – wenn es da nicht eine lästige Einschränkung gäbe. Um einen gemeinsamen Schlüssel zu erzeugen, müssen beide Kommunikationspartner im direkten Austausch miteinander stehen, quasi telefonieren. Danach können verschlüsselte Nachrichten zwar wie beim Briefverkehr auch in Abwesenheit des anderen deponiert werden, doch ganz am Anfang braucht es ein virtuelles Treffen. Aus diesem Grund haben Forschende weitergetüftelt und nach einer Methode gesucht, die den ersten Schritt umgeht und eine sichere Briefpost ohne vorheriges Telefonat ermöglicht.

Herausgekommen ist ein Verschlüsselungskonzept, das mit zwei anstatt einem Schlüssel operiert: Der eine dient zum Verschlüsseln, der andere zum Entschlüsseln. Nur Ersterer wird von der Besitzerin geheim gehalten, Letzterer wird veröffentlicht und ist für jeden frei verfügbar. Will Bob an Alice eine Nachricht schicken, so kann er ihren öffentlichen Schlüssel nehmen und damit die zu übermittelnden Daten verschlüsseln; sie wiederum kann die Nachricht mit den geheimen Zusatzinformationen, ihrem privaten Schlüssel, entschlüsseln. Das klingt fast zu schön, um wahr zu sein. Doch weil die öffentlichen Schlüsselinformationen keinerlei Hinweise für die Entschlüsselung liefern, funktioniert dieses Prinzip der Zweiteilung, das auch unter dem Schlagwort Public-Key-Kryptografie bekannt ist. Schauen wir uns diese Zauberei doch mal näher an und studieren ein konkretes Verfahren. Die Verschlüsselungsmethode, die ich Ihnen im Folgenden vorstelle, basiert auf dem Rucksack-Problem. Dieses Problem spiegelt eine uns vertraute Situation wider, mit der wir vor Ausflügen oder

Reisen konfrontiert sind: Was packen wir ein, wenn wir doch nur begrenzt viel mitnehmen können? Auch wenn das Szenario so rein gar nichts mit Kryptografie zu tun haben scheint, lässt es sich doch dafür nutzen, wie Bsp. 7.18 aufzeigt.

Beispiel 7.18: Verschlüsseln mit dem Rucksack-Problem

Wir betrachten hier eine abstrahierte Form des uns vertrauten Rucksack-Problems. Gegeben sind eine Menge von Gewichten und ein zulässiges Gesamtgewicht des Rucksacks. Das Ziel besteht darin herauszubekommen, welche Gewichte ausgewählt werden müssen, damit das zulässige Gesamtgewicht genau erreicht wird; wir setzen voraus, dass dies effektiv möglich ist. Auf dieser Grundlage lässt sich ein kryptografisches Protokoll definieren, wie es in Abb. 7.29 illustriert ist. Der Ablauf besteht aus fünf Schritten.

Alice wählt eine Menge von unterschiedlichen Gewichten, die zur Verfügung stehen (Schritt 1). Sie übermittelt diese in einer fest eingestellten Reihenfolge an Bob (Schritt 2). Bob kann nun auf dieser Basis eine 0-1-Sequenz verschlüsseln, deren Länge der Anzahl der Gewichte entspricht.

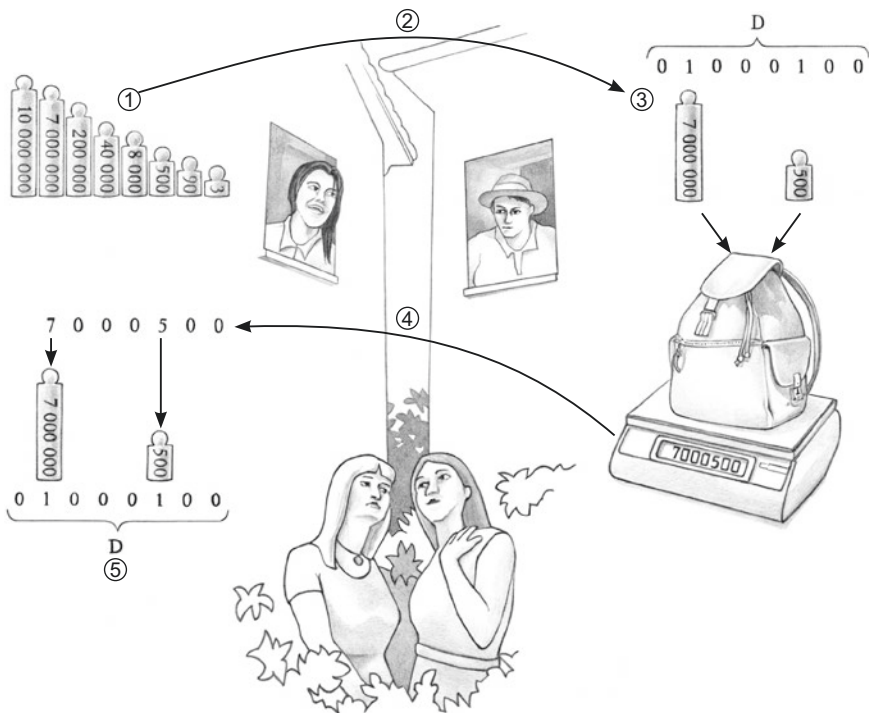


Abb. 7.29 Wie Nachrichten auf der Grundlage des Rucksack-Problems ver- und entschlüsselt werden können. (© Susanne Staubli, Eckart Zitzler)

Die Wertigkeit der einzelnen Bits bestimmt, welche Gewichte selektiert werden. Steht beispielsweise an der ersten Bitposition eine Eins, so wird das erste Gewicht ausgewählt, andernfalls nicht. Die Gewichte der ausgewählten Rucksackelemente werden aufsummiert (Schritt 3), und diese Summe stellt die Chiffre dar, die an Alice gesendet wird (Schritt 4). Alice wiederum kann die originale Bitsequenz aus der Chiffre ermitteln, wenn sie das korrespondierende Rucksack-Problem löst: Welche Elemente müssen ausgewählt werden, damit sich genau das gesendete Gesamtgewicht ergibt? Das ist die Idee. Sollen längere 0-1-Sequenzen verschlüsselt werden, unterteilt Bob die Daten in Blöcke, verschlüsselt jeden Block nach dem obigen Schema und sendet Alice anschließend eine Folge von Gesamtgewichten.

Zugegeben, die Verwendung des Rucksack-Problems wirkt etwas mysteriös – warum so umständlich? Der Clou ist, dass über die Wahl der Gewichte gesteuert werden kann, wie leicht bzw. schwer das Rucksack-Problem zu lösen ist. Sind die Gewichte so gewählt wie in Abb. 7.29, mit einem Gewicht aus der Menge 1 bis 9, einem aus der Menge 10, 20, 30, ..., 90, einem weiteren aus der Menge 100, 200, 300, ..., 900 usw., dann ist die Entschlüsselung ein Leichtes. An den Dezimalstellen des Gesamtgewichts kann Alice direkt ablesen, welche Auswahl Bob aufgrund der originalen 0-1-Nachricht getroffen hat. Sind die Gewichte hingegen anders verteilt, kann es sehr schwierig bis unmöglich sein, die gewählten Rucksackelemente aus dem Gesamtgewicht zu ermitteln. Genau dieser Umstand lässt sich nutzen und für die Sicherung der Daten einsetzen. Wir müssen nur sicherstellen, dass Alice bei der Entschlüsselung lediglich ein einfaches Rucksack-Problem zu lösen hat, während sich alle unerwünschten Zuhörer an einem schwierigen Rucksack-Problem die Zähne ausbeißen. Hier hilft uns wieder die Uhrenfunktion.

Abb. 7.30 skizziert den vollständigen Verschlüsselungs- und Entschlüsselungsvorgang. Alice wählt die Gewichte wie oben beschrieben, sodass sich die ausgewählten Rucksackelemente anhand der Dezimalstellen des Gesamtgewichts leicht identifizieren lassen. Daraus erzeugt sie eine schwierige Rucksack-Probleminstanz, indem sie die Gewichtswerte erst mit einer bestimmten Zahl multipliziert und anschließend über die Uhrenfunktion ummodellt. Die angepassten Gewichte übermittelt sie schließlich Bob. Bob verschlüsselt seine Daten wie vorher und sendet Alice das Gesamtgewicht. Doch wie bekommt Alice jetzt die ausgewählten Rucksackelemente heraus? Die Gewichte sind ja nicht mehr so leicht aus der Dezimalzahl abzulesen nach der Anwendung der Uhrenfunktion.

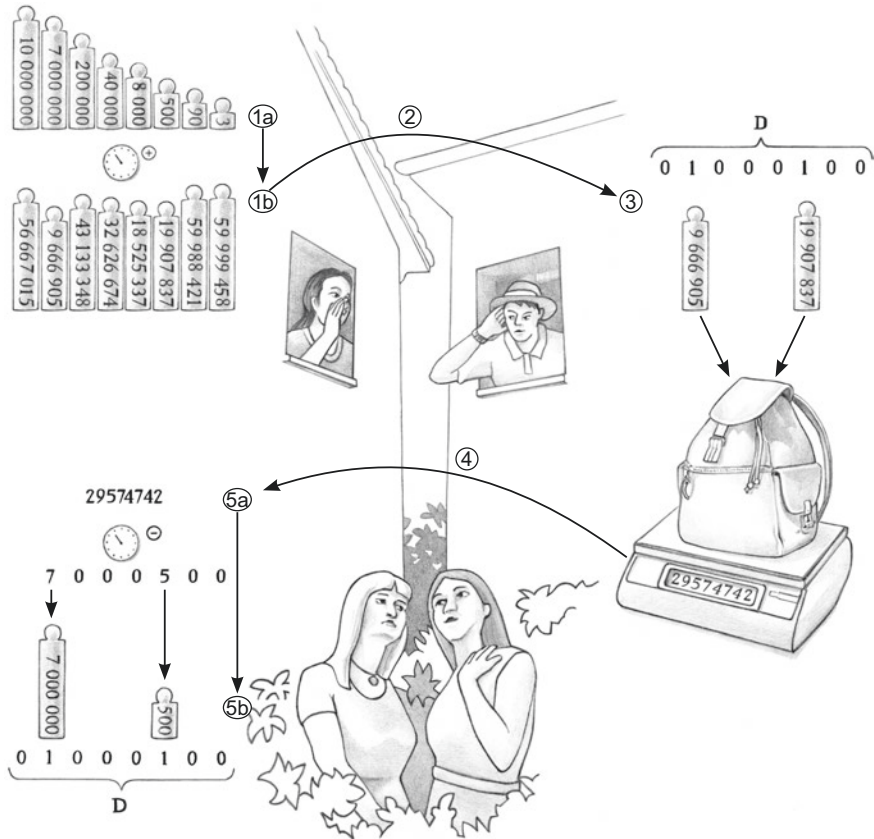


Abb. 7.30 Ein asymmetrisches Verschlüsselungsverfahren auf der Basis des Rucksack-Problems. (© Susanne Staubli, Eckart Zitzler)

Nun, Alice nutzt eine bestimmte Eigenschaft der Uhrenfunktion aus und kann so deren Anwendung umkehren (Anm. 7.5). Mit anderen Worten: Alice kann das Gesamtgewicht ermitteln, das sich ergeben würde, wenn Bob die ursprüngliche einfache Rucksack-Probleminstanz verwendet hätte. Allen anderen ist die Rücktransformation nicht möglich, weil ihnen Alices geheime Informationen fehlen. Alices privater Schlüssel besteht in diesem Szenario aus den gewählten einfachen Rucksackgewichten, der Uhrenfunktion und den Zusatzinformationen zur Umkehrung derselben; ihr öffentlicher Schlüssel ist die Folge der mit der Uhrenfunktion zerhackselten Rucksackgewichten.

Anmerkung 7.5: Uhrenfunktion umkehren

Der Trick hinter der Umkehrung der Uhrenfunktion beruht auf der Idee des Kehrwerts: Multiplizieren wir eine Zahl mit, sagen wir 5, dann können wir das Ergebnis mit dem Kehrwert von 5, also $1/5$, multiplizieren und erhalten die Ausgangszahl zurück. Das funktioniert, weil 5 multipliziert mit seinem Kehrwert 1 ergibt, logisch. Das Gleiche lässt sich auch bei Anwendung der Uhrenfunktion erzielen, nur dass der Kehrwert kein Bruch sein muss, sondern eine ganze Zahl sein kann. Dazu wählen wir zwei Zahlen aus, deren Produkt, eingespeist in die Uhrenfunktion, den Wert 1 liefert. Wenn wir nun eine beliebige Zahl mit der ersten gewählten Zahl multiplizieren, die Uhrenfunktion anwenden, das Ergebnis mit der zweiten gewählten Zahl multiplizieren und wieder die Uhrenfunktion anwenden, so erhalten wir wiederum die Ausgangszahl. Probieren wir es aus. Bei einer Uhrengroße von 17 bilden die Zahlen 20 und 40 ein Paar, dessen Produkt nach Anwendung der Uhrenfunktion 1 ergibt. Multiplizieren wir nun 7 mit 20 und wenden die Uhrenfunktion an, so kommt 4 heraus; 4 multipliziert mit 40 und dann eingesetzt in die Uhrenfunktion liefert wieder 7. Das klappt immer, solange die Ausgangszahl kleiner ist als die gewählte Uhrengroße. Denn wie bei der Schlüsselgenerierung ist es egal, ob wir erst multiplizieren und dann die Uhrenfunktion anwenden oder umgekehrt.

Dieses Phänomen macht sich Alice beim Rucksack-Problem zunutze, auch sie wählt so ein Zahlenpaar. Die ursprünglichen Gewichte werden mit der ersten gewählten Zahl multipliziert und durch die Uhrenfunktion gespeist (Schritt 1b in Abb. 7.30). Wenn Bob nun für diese modifizierte Gewichtemenge eine Auswahl trifft und deren Gesamtgewicht zurücksendet, multipliziert Alice den erhaltenen Wert mit der zweiten Zahl und wendet erneut die Uhrenfunktion an (Schritt 5a in Abb. 7.30). Auf diesem Weg erhält Alice die Summe der ausgewählten Rucksackelemente mit Hinblick auf ihr ursprüngliches Gewicht.

Das Verfahren, das Sie soeben kennengelernt haben, wurde 1978 von Ralph Merkle und Martin Hellman vorgeschlagen. Es realisiert das Prinzip der asymmetrischen Verschlüsselung, weil zum Ver- und Entschlüsseln verschiedene Schlüssel zum Einsatz kommen; das Gegenstück verkörpern die symmetrischen Verfahren wie DES und AES, bei denen derselbe Schlüssel für beide Zwecke eingesetzt wird. Die Sicherheit von Merkles und Hellmans Methode beruht auf der Annahme, dass es im Allgemeinen schwierig ist, aus dem Rucksackgesamtwert auf die Auswahl der Rucksackelemente zu schließen. Es stellte sich später heraus, dass das Rucksack-Problem in der Tat nicht schwierig genug ist und geknackt werden kann. Deswegen wird es heute nicht mehr eingesetzt. Der Grund, dass ich Sie trotzdem durch dieses Verfahren gelotst habe, hat mit der Verständlichkeit zu tun. Die Idee der Public-Key-Kryptografie lässt sich an ihm leichter nachvollziehen als an den mathematisch elaborierten Algorithmen, die letztlich das Rennen gemacht haben.

Yin und Yang im Alltag: Public-Key-Kryptografie

Heutige Public-Key-Kryptosysteme beruhen in der Regel auf mathematischen Problemen. Das weit verbreitete RSA-Verfahren setzt beispielsweise auf die Schwierigkeit der Primfaktorzerlegung: Es ist kein effizienter Algorithmus bekannt, der eine beliebige große Zahl in vernünftiger Zeit als ein Produkt von Primzahlen darstellen kann. Denn RSA verwendet das Produkt zweier großer Primzahlen in dem Schlüsselpaar; können Dritte die Primzahlen ermitteln, ist es mit der Sicherheit vorbei. Doch obwohl das Konzept bereits einige Jahre auf dem Buckel hat, gilt es nach wie vor als sehr sicher. Zudem bietet es gegenüber dem Rucksack-Ansatz den Vorteil, dass sowohl mit dem privaten als auch mit dem öffentlichen Schlüssel ver- und entschlüsselt werden kann. Dabei verhalten sich die Schlüssel komplementär zueinander, d. h., verschlüsselt man mit dem privaten Schlüssel, kann man das Ergebnis mit dem öffentlichen Schlüssel entschlüsseln und umgekehrt. Wenn Sie möchten, können Sie die Hintergründe zur RSA in Anm. 7.6 nachlesen.

Anmerkung 7.6: Das RSA-Verschlüsselungsverfahren

Der Name RSA steht für die Anfangsbuchstaben der Nachnamen seiner Erfinder Ron Rivest, Adi Shamir und Leonard Adleman (ja, es ist derselbe Leonard Adleman, den Sie im Zusammenhang mit dem DNA-Computing kennengelernt haben). Der Verschlüsselungsalgorithmus wurde 1977 vorgestellt und ähnelt im Ablauf dem Schlüsselgenerierungsverfahren von Diffie und Hellman. Dabei kommen drei Arten von Schlüsselinformationen zum Einsatz: eine öffentliche Zahl n , aus der die Uhrengröße für die Uhrenfunktion abgeleitet wird und die das Produkt zweier großer Primzahlen ist, eine private Zahl e , die nur Alice kennt, und eine Zahl d , die wiederum öffentlich ist.

Will Bob eine Nachricht an Alice verschlüsseln, dann unterteilt er sie zunächst in Blöcke einer fixen Länge (einige hundert Bit). Jeder Block wird dann als Dualzahl interpretiert, mit der Zahl d potenziert und anschließend durch die Uhrenfunktion geschleust; die resultierende Zahl wird als Dualzahl dargestellt, die korrespondierende 0-1-Sequenz ist der verschlüsselte Block. Alice empfängt den verschlüsselten Block, interpretiert ihn wieder als Dualzahl, potenziert diese mit ihrer geheimen Zahl e , wendet die Uhrenfunktion an und erhält so wieder den Originalblock. Letztlich wird also eine Dualzahl mit dem Produkt aus d und e potenziert. Weil es keine Rolle spielt, ob zuerst mit d oder mit e potenziert wird, können die Schlüssel in beliebiger Reihenfolge eingesetzt werden, d. h., mit beiden Schlüsseln kann entweder ver- oder entschlüsselt werden.

Mit dem RSA-Verfahren wurde 1977 die Public-Key-Kryptografie ins praktische Leben gerufen; die theoretischen Vorarbeiten hatten ein Jahr zuvor Whitfield Diffie und Martin Hellman veröffentlicht, als sie die Idee

eines asymmetrischen Public-Key-Verfahrens skizzierten. Die Public-Key-Kryptografie erlaubt nicht nur, dass Nachrichten verschlüsselt verschickt werden können, ohne dass die Partner direkt miteinander kommunizieren müssen. Sie ermöglicht auch zu belegen, dass jemand tatsächlich der ist, für den er sich ausgibt. Denn ein Problem kann sich ganz klar ergeben: Wenn Eva und Eveline sich gegenüber Alice als Bob und gegenüber Bob als Alice ausgeben, dann können sie sich unbemerkt zwischen diese beiden schalten und den gesamten Nachrichtenaustausch mitverfolgen sowie verändern (siehe Abb. 7.31). Es braucht also noch etwas, um zu wissen, dass der öffentliche Schlüssel, den Bob verwendet, tatsächlich auch Alice gehört und nicht jemand anderem. Wie lässt sich das Problem lösen?

Die einfache Lösung liegt auf der Hand: Alice und Bob müssen sich mal treffen, d. h. über einen vertrauenswürdigen Kanal ihre öffentlichen Schlüssel austauschen. Aber hatten wir nicht genau das mit den asymmetrischen Verschlüsselungsverfahren vermeiden wollen? Richtig. Es geht auch eleganter, und zwar über eine Zertifizierungsstelle. Eine Zertifizierungsstelle ist eine vertrauenswürdige Instanz, die die Identität von Teilnehmern im Internet bestätigt und einer Passstelle gleichkommt. Jeder, der an einer verschlüsselten Kommunikation teilnehmen möchte, muss einmal bei dieser Stelle vorbeigehen und sich ausweisen – dieser Aufwand ist vertretbar, schließlich ist er einmalig und wiederholt sich nicht für jede neue Bekanntschaft. Er bekommt dort ein Zertifikat – quasi einen Pass –, das ihn mit seiner Identität ausweist und das er bei jeder Kommunikation vorweist. Gleichzeitig erhält er von der Zertifizierungsstelle deren öffentlichen Schlüssel, um selbst auch die

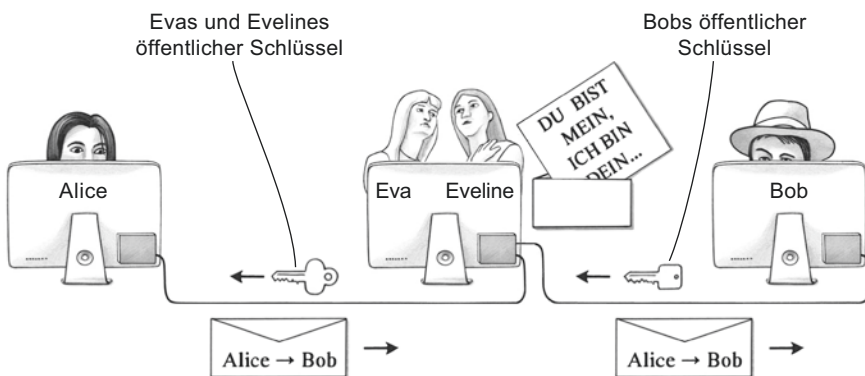


Abb. 7.31 Das „Women in the Middle“-Szenario: Zwei Angreiferinnen klinken sich unbemerkt in den Datenverkehr zwischen Alice und Bob. (© Susanne Staubli, Eckart Zitzler)

Authentizität anderer Kommunikationspartner überprüfen zu können. Wie das im Detail funktioniert, verrät Ihnen Bsp. 7.19.

Beispiel 7.19: Bist Du der, der Du bist?

Abb. 7.32 visualisiert die Schritte, die bei einer zertifikatsgestützten Nachrichtenübermittlung erforderlich sind; neben Alice und Bob sehen Sie Zeus, der die vertrauenswürdige Zertifizierungsstelle repräsentiert. In einem ersten Schritt gehen beide, Alice und Bob, einzeln bei Zeus vorbei, weisen sich aus und lassen ihre öffentlichen Schlüssel beglaubigen. Zeus stellt beiden ein eigenes Zertifikat aus, indem er den öffentlichen Schlüssel beider mit seinem privaten Schlüssel verschlüsselt; darüber hinaus gibt er ihnen seinen eigenen öffentlichen Schlüssel mit (Schritt 2). Dieser Vorgang ist nur einmalig nötig, von da an können Alice und Bob beliebig miteinander und mit anderen ausgewiesenen Teilnehmern kommunizieren.

Nun möchte Alice an Bob eine Mitteilung schicken. Dazu sendet Bob Alice sein Zertifikat (Schritt 3), um ihr zu zeigen: Ich bin tatsächlich Bob! Alice entschlüsselt Bobs Zertifikat mit Zeus' öffentlichem Schlüssel und erhält so Bobs öffentlichen Schlüssel (Schritt 4). Nun baut sie eine Nachricht zusammen, die aus ihrem Geheimtext, der mittels ihres privaten Schlüssels verschlüsselt wird, und ihrem Zertifikat besteht; beides verschlüsselt sie mit Bobs öffentlichem Schlüssel (Schritt 5) und sendet die Chiffre an Bob (Schritt 6). Bob wiederum nimmt die Doppelnachricht entgegen und entschlüsselt sie mit seinem privaten Schlüssel (Schritt 7a). Anschließend packt er die weiteren Bestandteile aus: Zuerst ermittelt er über Alices Zertifikat und Zeus' öffentlichen Schlüssel den öffentlichen Schlüssel von Alice (Schritt 7b); danach nimmt er Alices öffentlichen Schlüssel, um den Geheimtext zu dechiffrieren (Schritt 7c).

Etwas umständlich das Ganze? Ja, Sie haben recht, doch entscheidend ist: Es funktioniert.

Asymmetrische Verschlüsselungsverfahren bieten gegenüber ihren symmetrischen Pendanten viele Vorteile: Sie benötigen wenige Schlüssel – nur einen pro Teilnehmer anstatt einen pro Teilnehmerpaar –, vermeiden das Problem des Schlüsselaustauschs vor dem eigentlichen Datenaustausch und erlauben verschiedene Arten von Authentifizierungen, beispielsweise digitale Unterschriften. Ihr Sicherheitskonzept setzt auf die praktische Unlösbarkeit bestimmter Berechnungsprobleme, die aber theoretisch bislang nicht bewiesen werden konnte. Dennoch gelten die Verfahren als sicher, insofern die Schlüssel ausreichend lang sind – und es bei der praktischen Umsetzung keine Schwachstellen gibt. Der einzige Nachteil im Vergleich zu den symmetrischen Methoden ist, dass sie sehr rechenintensiv sind und wesentlich langsamer arbeiten. Aus diesem Grund führen beide Konzepte eine friedliche Koexistenz und werden häufig miteinander kombiniert.

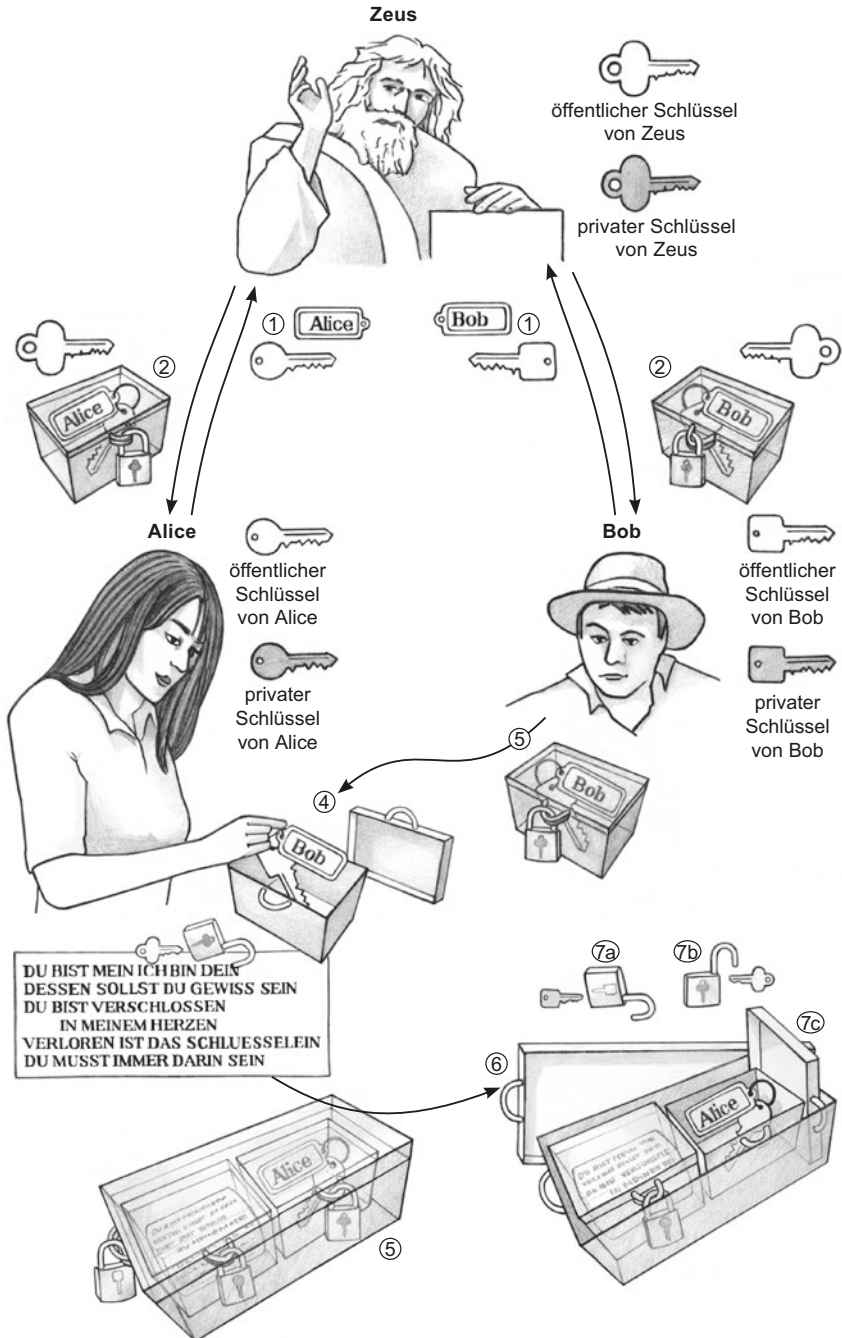


Abb. 7.32 Sich mittels Zertifikaten ausweisen. (© Susanne Staubli, Eckart Zitzler)

Duftcodes – und wie sie gehackt werden

Schließen wir das Thema der Geheimcodes ab mit einem Blick auf die Biologie, in der sich unzählige Anknüpfungspunkte zur Kryptografie finden lassen. Das fängt schon beim Schlüssel-Schloss-Prinzip an. Dieses finden wir sehr häufig in der Natur, und immer wieder dient es als Zugangskontrolle. Das Immunsystem erkennt z. B. an der Oberfläche, ob ein Stoff zum eigenen Körper gehört oder nicht. Zellen lassen nur bestimmte Stoffe mit gewissen Strukturen durch – und Viren nutzen das aus, indem sie die Strukturen imitieren. Enzyme können mit Substraten ebenfalls nach dem Schlüssel-Schloss-Prinzip interagieren und reagieren. Und auch beim Riechen ist das so, wie wir in Anm. 7.7 sehen – beim Duft gibt es unzählige Schlüssel und Schlösser, die einen Geruch ausmachen, ja es gibt eine ganz eigene Sprache des Dufts.

Anmerkung 7.7: Die Sprache des Dufts

Ein Apfel duftet. Wie jeder Gegenstand sondert er kleinste Moleküle – Duftstoffe – ab, die charakteristisch für ihn sind und seinen Duft ausmachen. Einige Gegenstände sondern viele dieser Duftmoleküle ab, andere weniger, und Steine beispielsweise sondern an sich gar keine Moleküle ab, sie nehmen eher den Duft von Materialien an, mit denen sie in Kontakt waren. Je mehr Moleküle – Odorants – vorhanden sind, desto intensiver der Geruch und desto einfacher die Zuordnung. Diese Moleküle nehmen wir beim Einatmen auf. Doch wie erkennen wir einen Duft? Steht jedes Molekül für einen bestimmten Duft? Und wie ergibt sich dann die Vielfalt an Düften, die wir bei Blumen, beim Olivenöl oder beim Wein vorfinden?

Es hat sich gezeigt, dass auch hier das Schlüssel-Schloss-Prinzip zur Anwendung kommt: Es gibt Geruchsrezeptoren in der Nase (siehe Abb. 7.33), die eine bestimmte Form aufweisen. Diese Form ist so beschaffen, dass nur Geruchsmoleküle mit einer bestimmten Struktur an einen Rezeptor binden können und diesen somit reizen. Das heißt nicht, dass ein Rezeptor nur auf einen bestimmten Dufttyp reagiert, sondern auf bestimmte Eigenschaften eines Duftstoffes, die in der Regel mehrere Duftstoffe teilen. Erst die Kombination aller Eigenschaften des Dufts macht das Geruchsbild aus. Wenn wir den Duft mit der Schriftsprache vergleichen wollten, dann könnten wir sagen, jeder Typ eines Geruchsrezeptors erkennt einen bestimmten Buchstaben. Der Mensch hat ungefähr 400 Arten von Geruchsrezeptoren, während ein Hund knapp 1000 besitzt. Darüber kann ein Eisbär allerdings nur lachen: Mit seinen 10.000 verschiedenartigen Geruchsrezeptoren kann er sogar unter Wasser riechen.

Ein Duft ist also eine Kombination vieler Schlüssel, und jedes Duftmolekül ist ein Schlüssel, um bestimmte Sinneszellen zu erregen. Duftmoleküle verschlüsseln sozusagen die Information über den Gegenstand, von dem sie

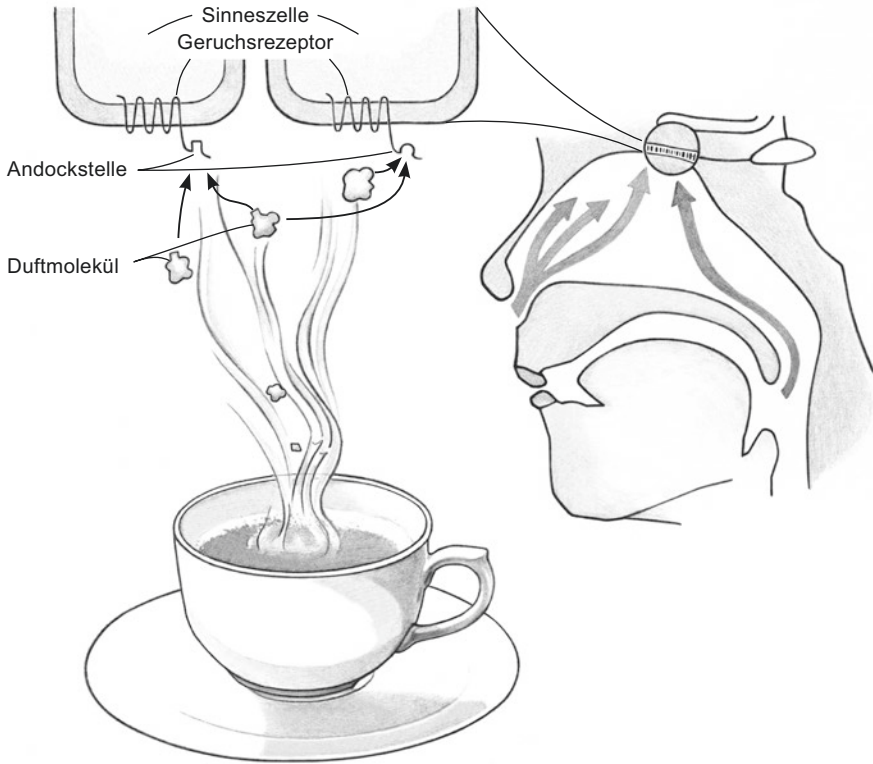


Abb. 7.33 Wie wir Duftmoleküle wahrnehmen. (© Susanne Staubli, Eckart Zitzler)

ausgesendet werden. Düfte werden auch von Tieren benutzt, um sich untereinander zu verständigen. Ameisen beispielsweise benutzen verschiedenste Arten von chemischen Substanzen, um sich zu warnen, anzulocken, sich gegenseitig zu erkennen, um Territorien zu markieren usw. In diesem Fall spricht man von Pheromonen, bei denen es sich wie bei den Duftstoffen um flüchtige Substanzen handelt, die allerdings über ein spezielles Organ wahrgenommen werden, nämlich das Jacobson-Organ. Pheromone dienen der Kommunikation zwischen Artgenossen und lösen stereotype Verhaltensweisen aus. Duftcodes sind also chemische Geheimcodes zwischen Tieren der gleichen Art und essenziell für das Funktionieren der Gemeinschaft. Wenn es jedoch artfremden Tieren gelingt, diesen Code zu knacken, dann kann es heikel werden. Einige Ameisenarten betätigen sich als Codeknacker anderer Ameisenarten und lassen sich parasitär herumtragen und bewirten. Oder mit etwas mehr Dramatik: Bola-Spinnen imitieren den Lockduft bestimmter

Nachfalterweibchen, was mitunter tödliche Konsequenzen für die Männchen der Gattung hat (Anm. 7.8).

Anmerkung 7.8: Chemische Maskerade

Bola-Spinnen sind Hacker. Sie haben es geschafft, den sexuellen Duftcode bestimmter Nachfalterweibchen zu knacken und die Duftstoffe selbst herzustellen. Und sie haben eine darauf abgestimmte, sehr eigene Fangtechnik entwickelt: Sie benutzen kein Radnetz, sondern einzig einen herunterhängenden Faden, an dessen Ende sich ein Leimtropfen befindet. Der Faden fungiert als Lasso und erinnert zusammen mit dem Leimtropfen an eine Bola, eine WurfwaFFE, die aus zusammengeknuteten Leinen besteht, an deren Ende Kugeln angebracht sind – daher der Name der Spinne (Abb. 7.34).

Der Beutefang läuft nun folgendermaßen ab: Zunächst sondern die Bola-Spinnen ihr Imitat des Sexualpheromons der Nachfalterweibchen ab. Die Männchen derselben Nachfalterart werden daraufhin angelockt und umkreisen die Spinne. Nun wirft die Bola-Spinne ihr Lasso aus und versucht die Beute mit dem schleudernden Lasso zu fangen. Wird der Nachfalter getroffen, so bleibt er am klebrigen Sekret des Spinnfadens hängen – aus der Traum vom Rendezvous. Diese Vortäuschung falscher Tatsachen, also des Dufts, nennt man Mimikry, in dem Fall handelt es sich um chemische Mimikry.

Es macht also Sinn, Codes geheim zu halten und Informationen vor anderen zu verstecken. Den Nachfaltern fehlt offenbar noch eine Zertifizierungsstelle, die ihnen bescheinigt, dass es sich tatsächlich um ein Nachfalterweibchen handelt, das seinen betörenden Duft versprüht. Bei uns Menschen übernimmt das Immunsystem diese Funktion, doch auch es kann getäuscht werden. Kryptografie ist ein ewiger Wettlauf, auch in der Biologie.



Abb. 7.34 Eine Bola-Spinne mit ihrem Fangfaden (links) und ein Bola-Lasso (rechts). (© Richard A. Bradley (links), jasantiso/Getty Images/iStock (rechts))

Fazit

Die Welt im Computer darstellen heißt weitaus mehr, als Informationen in 0-1-Muster zu übersetzen. Die Art der Codierung wirkt sich auf alle Aspekte der Informationsverarbeitung aus, sie steht an ihrem Anfang. Entsprechend sorgfältig muss das Datenformat gewählt werden. Je nach Zweck können ganz andere Eigenschaften gefragt sein. Geht es darum, die Informationen möglichst einfach und effizient zu bearbeiten? Sollen die Informationen zwischen verschiedenen Computersystemen ausgetauscht werden oder für den Menschen lesbar sein? Zielen wir darauf ab, die Daten gegen Zerstörungen zu schützen? Oder wollen wir die Informationen möglichst kompakt darstellen, sie verstecken oder gar vor dem Zugriff Dritter schützen? All diese Überlegungen spielen eine Rolle und können das Finden eines geeigneten Codes zu einer eigenen Wissenschaft – oder eben Kunst – machen.

Und auch in der Natur sind Codes allgegenwärtig. Am Anfang des Lebens steht der genetische Code, die universelle Sprache, in der alle Lebewesen die Baupläne ihrer Proteine formulieren. Unzählige weitere Codes sind ihm gefolgt, vom Frequenzcode im Nervensystem über Duftcodes bis zur Körpersprache und schließlich zur menschlichen Schriftsprache. Der Computer hat nun unsere Möglichkeiten, Informationen außerhalb unseres Körpers festzuhalten, erweitert um das Feld der binären Darstellungen. Unser Bedürfnis, Informationen darzustellen und immer bessere Formen zu finden, wird die Entwicklung immer wieder antreiben. Wir können gespannt sein, was die Kunst der Darstellung noch alles hervorbringt.

Zum Nach- und Weiterlesen

- Alberts, B., Johnson, A., Lewis, J., Morgan, D., Raff, M., Roberts, K., & Walter, P. (2017). *Lehrbuch der Molekularen Zellbiologie* (6. Aufl.). Weinheim: Wiley-VCH.
- Bartels, K. (2012). Stichwort „Information“. *Neue Zürcher Zeitung*, 176,45.
- Beutelspacher, A., Neumann, H. B., & Schwarzpaul, T. (2010). *Kryptografie in Theorie und Praxis: Mathematische Grundlagen für Internetsicherheit, Mobilfunk und Elektronisches Geld* (2. Aufl.). Wiesbaden: Vieweg + Teubner.
- Bradley, R. A. (2012). *Common spiders of North America*. Oakland: University of California Press.
- Dewdney, A. K. (1993). *The new turing Omnibus: Sixty-six excursions in computer science*. New York: Holt Paperbacks.
- Foelix, R. F. (2015). *Biologie der Spinnen*. Frankfurt a. M.: Chimaira.

- Frings, S., & Müller, F. (2014). *Biologie der Sinne: Vom Molekül zur Wahrnehmung*. Berlin: Springer.
- Gleick, J. (2011). *Die Information: Geschichte, Theorie, Flut*. München: Redline Verlag.
- Singh, S. (2001). *Geheime Botschaften. Die Kunst der Verschlüsselung von der Antike bis in die Zeiten des Internet*. München: Deutscher Taschenbuch Verlag.

Bildquellen

- Abb. 7.25: Boca, Bildnummer 101996899, <https://stock.adobe.com>. (links, abgerufen am 30.05.2017)
- Abb. 7.34: Richard A. Bradley, <http://spidersinohio.net/> (links, abgerufen am 28.05.2017); jasantiso/Getty Images/iStock, Bildnummer 482881739, <http://www.istockphoto.com>. (rechts, abgerufen am 30.05.2017).

8

Denk!

Überblick

Probleme lösen – um nichts Geringeres geht es in diesem Kapitel. Damit ist die Ausgestaltung eines Verarbeitungsvorgangs bzw. eines Algorithmus gemeint, der für eine gegebene Funktion den Rechenweg von der Eingabe zur gewünschten Ausgabe definiert. Zunächst beschäftigen wir uns mit den Grundlagen des Algorithmenentwurfs. Sie lernen bewährte Strategien kennen, mit denen sich schwierige Probleme anpacken und effiziente Programme konstruieren lassen. Anschließend schauen wir uns intelligente Algorithmen an, d. h. Verarbeitungsprozesse, die Aspekte des menschlichen Verhaltens nachbilden. Es geht darum, wie Computer lernen, spielen und Sprachen erkennen können. Zum Schluss widmen wir uns der Frage, was eigentlich ein guter Algorithmus ist bzw. wann ein Problem gut gelöst ist. Wir werden sehen, dass es große Unterschiede geben kann zwischen geschickten und naiven Lösungen und dass manche Probleme gar nicht lösbar sind. Algorithmik, Künstliche Intelligenz und Komplexitätstheorie heißen die Zutaten dieses Kapitels im Informatikjargon.

Die Motivation, Rechenmaschinen zu entwickeln, ist wohl gleichermaßen in der Faulheit und der Neugierde des Menschen begründet. Einerseits wollte er stupide Aufgaben wie das Aufaddieren von Zahlenkolonnen an einen maschinellen Rechenknecht delegieren, andererseits aufwändige Berechnungen, wie sie zur Vorhersage der Flugbahnen von Himmelskörpern erforderlich sind, in kurzer Zeit und fehlerfrei durchführen können. Der Rechenvorgang als solcher war bekannt, die Frage lautete vielmehr, wie sich die einzelnen Rechenschritte mechanisch und elektronisch umsetzen ließen. Als der Computer schließlich Realität und immer leistungsfähiger, erschwinglicher und kleiner wurde, trat eine andere Frage in den

Vordergrund: Wie sieht ein adäquater Rechengvorgang aus, um eine vorgegebene Aufgabe automatisiert zu bearbeiten? Der Verarbeitungsprozess muss nicht nur auf einem Computer umsetzbar sein, sondern auch die zur Verfügung stehenden Ressourcen wie Prozessor und Speicher bestmöglich nutzen.

Der Fokus verschob sich also immer mehr von der Hardware zur Software, und die Konzeption passender Programme wurde eine Wissenschaft für sich. Mit der Zeit bildeten sich Schemata für Verarbeitungsabläufe heraus, die in vielen Anwendungsfeldern erfolgreich eingesetzt werden konnten. Und es wurden die theoretischen Grundlagen erarbeitet, um den Schwierigkeitsgrad von Aufgaben klassifizieren und die Güte von Lösungsverfahren beurteilen zu können. Von Anfang an ging es dabei auch darum, die Grenzen der automatisierten Informationsverarbeitung auszuloten: Was lässt sich berechnen und was nicht? Vor allem das Thema der Intelligenz – inwiefern lässt sich mit einem Computersystem ein Verhalten erzeugen, das wir mit natürlicher Intelligenz in Verbindung bringen? – hat sowohl die Forschenden als auch die Allgemeinheit seit jeher beschäftigt.

Damit haben Sie einen ersten Überblick über die Themen, die Sie im Folgenden erwarten. Und weil Sie jetzt das längste Kapitel in diesem Buch angehen, möchte ich Sie an dieser Stelle noch einmal explizit ermuntern, sich von Ihren Interessen durch das Dickicht leiten zu lassen. Gut, und jetzt steigen wir ein bei den Grundlagen und der Ausgangsfrage: Wie lässt sich für eine vorgegebene Aufgabenstellung ein Berechnungsablauf entwickeln?

Knifflige Probleme – bewährte Strategien

Die Informationsverarbeitung hat zum Ziel, aus einer Eingabe bzw. Wahrnehmung schrittweise eine gewünschte Ausgabe bzw. Reaktion abzuleiten. Die Umwandlung der eingehenden Daten zu den ausgehenden Daten kann dabei auf verschiedenen Wegen erfolgen, und für die Ausgestaltung dieses Weges interessieren wir uns nun – unter der Annahme, dass die elementaren Verarbeitungsoperationen durch die Computer-Hardware vorgegeben sind. Allerdings soll es weniger um die detaillierte Formulierung eines Programms gehen, sondern vielmehr um die Lösungsstrategien, die dem Verarbeitungsprozess zugrunde liegen. Fangen wir an, indem wir einen Blick auf die Lebenswelt werfen, wie die Informationsverarbeitung dort zielgerichtete Reaktionen „berechnet“.

Von Verhaltensprogrammen und Verhaltensmustern

Lebewesen müssen eigentlich ständig Aufgaben bewältigen und Probleme lösen: Nahrung beschaffen, bedrohlichen Situationen entkommen, sich fortpflanzen usw. Dafür haben sie Verhaltensweisen herausgebildet, das sind Verknüpfungen von inneren und äußeren Signalen mit Reaktionen, die dem Organismus zur Verfügung stehen. Die Umsetzung dieser Verknüpfungen ist Aufgabe der Informationsverarbeitung. Wenn wir beispielsweise Hunger haben und den Apfel vor uns greifen wollen – das sind die inneren und äußeren Signale –, dann muss unser Gehirn daraus ein Verhalten „berechnen“, also eine Folge einfacher Schritte ermitteln, um den Apfel zu ergreifen. Letztlich setzt es dabei eingehende Sequenzen von Nervenimpulsmustern, die unsere Sinneszellen generieren, in ausgehende Sequenzen von Nervenimpulsmustern um, die die entsprechenden Muskeln ansteuern (siehe Abb. 8.1). Natürlich werden da nicht einfach „Inputs“ in „Outputs“ übersetzt, vielmehr handelt es sich beim gesamten Greifvorgang um einen rückgekoppelten Prozess, bei dem die Bewegung permanent über unsere Wahrnehmung kontrolliert und ggf. angepasst wird. Auch wenn es sich für uns um eine selbstverständliche Verhaltensweise handelt, unser Gehirn hat hier einiges zu rechnen.

Das ist nur ein Beispiel für eine Verhaltensweise, wir verfügen über ein riesiges Repertoire an Reaktionsmöglichkeiten. Ein Großteil davon läuft unbewusst ab und kann nur eingeschränkt oder gar nicht willentlich beeinflusst werden. Die Regulation unseres Körpermilieus beispielsweise repräsentiert ein umfassendes Verhaltensprogramm, das durch unser vegetatives

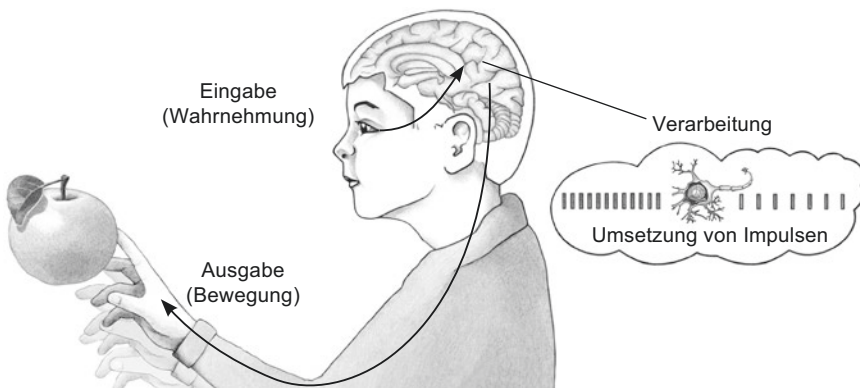


Abb. 8.1 Ein komplexer Berechnungsvorgang für unser Gehirn: einen Gegenstand greifen. (© Susanne Staubli, Eckart Zitzler)

Nervensystem realisiert wird und autonom abläuft. Bei Gefahr werden Herz- und Atemfrequenz erhöht, ohne dass wir bewusst etwas dafür tun müssten. Das ist bei Tieren nicht anders, auch hier gibt es Verhaltensprogramme, die teilweise oder vollständig genetisch fixiert sind. Beim Fadenwurm *C. elegans* ist das gesamte Verhalten durch ein starres Nervensystem festgelegt, Spinnen tragen die Methode zum Weben eines Netzes in sich und bei Ameisen werden durch Duftstoffe stereotype Verhaltensweisen ausgelöst, z. B. bei der Verteidigung des Nestes oder beim Sammeln von Nahrung. Es gibt aber auch Verhaltensweisen, die durch Prägung – ein bekanntes Beispiel sind die Graugänse, die in den ersten Minuten nach dem Schlüpfen das Erscheinungsbild ihrer Mutter irreversibel verinnerlichen – sowie Lernen allgemein erworben oder durch Überlegungen abgeleitet werden. Verhaltensprogramme können also situativ zusammengestellt werden, fix vorgegeben sein oder als Mischform vorliegen. All diese Varianten stehen für verschiedene Freiheitsgrade im Verhaltensrepertoire, eine höhere Flexibilität erfordert in der Regel erweiterte geistige Fähigkeiten.

Je komplexer die Umwelt ist, in der sich Lebewesen aufhalten, desto wichtiger sind Verhaltensmuster, um auch in unbekannten Situationen schnell reagieren zu können. Verhaltensmuster sind quasi Schablonen für Verhalten, die je nach Gegebenheit eingesetzt und angepasst werden. Das konkrete Verhalten kann im Detail variieren, das Verhaltensmuster ist dagegen ein Schema, das sich bewährt oder eingespielt hat. Jeder von uns hat seine Verhaltensmuster, genauso wie jede Art ihre Verhaltensmuster besitzt. Verhalten wir uns in einer unangenehmen Lage still oder gehen wir in die Offensive? Das sind Beispiele für individuelle Verhaltensmuster. Spinnennetze stehen für ein übergeordnetes Verhaltensmuster (siehe Abb. 8.2): Spinnen setzen sie ein, um ihre Nahrung zu fangen, doch das konkrete Aussehen eines Netzes hängt von der spezifischen Spinnenart ab. Es gibt aber auch Verhaltensmuster, die sich über Arten hinweg entwickelt haben, man könnte sagen, bewährte Strategien der Natur. Denken Sie z. B. an das Prinzip „Versuch und Irrtum“: Wie die Stubenfliege am Fenster entlangfliegt, bis sie schließlich doch noch die Öffnung findet, oder wenn wir vor einem mysteriösen Automaten mit vielen Knöpfen stehen, wild herumprobieren und schauen, was passiert. Genauso das Prinzip der Kooperation, des gemeinsamen zielgerichteten Handelns, das sich bei verschiedenen Organismen entwickelt hat. Oder bei Raubkatzen das Prinzip, Beutetiere von der Herde zu isolieren und dann gemeinsam zu jagen, welches unter der Redewendung „Teile und herrsche“ bekannt ist.

Ebenso wie in der Natur haben sich in der Informatik Muster herausgebildet, wie Probleme gelöst werden können. Ich werde ihnen gleich



Abb. 8.2 Verhaltensmuster in der Natur. (© Judith Büsser (links), Frithjof Nolting (Mitte und rechts))

verschiedene Problemlösungsstrategien vorstellen, aber vorher müssen wir noch klären, was in der Informatik unter einem Problem verstanden wird und was es mit dem Begriff des Algorithmus auf sich hat.

Wenn Informatiker Probleme haben

Auch beim Computer geht es darum, ein Verhalten zu erzeugen, d. h. eine Eingabe in eine Ausgabe zu übersetzen. Allerdings lösen Computer nicht das Problem ihres Überlebens, sondern Probleme, die wir ihnen gestellt haben. Tatsächlich verwenden Informatikerinnen den Ausdruck „Problem“, wenn sie sich mit einer Fragestellung beschäftigen, die von einem Computer bearbeitet werden soll. Da schwingt das Wörtchen „schwierig“ mit, und in der Tat geht es häufig um schwierige Fragestellungen. Das schließt jedoch nicht aus, dass ein Problem ganz einfach sein kann. Dabei abstrahiert man von den Interaktionen mit dem Benutzer und allen nichtrelevanten Details, denn es geht vor allem um die Frage des Rechenwegs: Wie lässt sich der Kern der Fragestellung automatisiert lösen? Aus diesem Grund legt man auch das universelle Modell der Turing-Maschine zugrunde, selbst wenn das Szenario heutzutage altertümlich wirkt: Der Computer wird mit Daten gefüttert, rechnet vor sich hin und spuckt schließlich das Ergebnis aus. Natürlich gibt es in der Realität genauso Rückkopplungen, wie es beim Verhalten von Lebewesen der Fall ist; das Betriebssystem beispielsweise läuft

kontinuierlich im Hintergrund und ist im ständigen Wechselspiel mit anderen Programmen und dem Benutzer. Doch für viele Fragestellungen ist es vollkommen ausreichend, das in Abb. 8.3 skizzierte Szenario zu betrachten.

Ein Problem ist demnach beschrieben durch die Art der Eingabe und das gewünschte Resultat – alles in computerlesbarer Form, doch da das Thema der Codierung ausführlich in Kap. 7 behandelt ist, lassen wir diesen Aspekt im Folgenden außer Acht. Ein paar Probleme sind uns bereits begegnet: das Hamilton-Pfad-Problem (Kap. 1) oder das Rucksack-Problem (Kap. 7), ja selbst die Addition zweier Dualzahlen (vgl. Kap. 2), lässt sich als Problem betrachten. Beim Hamilton-Pfad-Problem beispielsweise besteht die Eingabe aus einem Stadtplan, einem Start- und einem Zielpunkt, und die Ausgabe ist eine Folge von Fahrbahnen, die vom Start zum Ziel führt und dabei alle Kreuzungen auf dem Stadtplan genau einmal besucht – oder die Antwort, dass eine solche Lösung nicht existiert. Wichtig an dieser Stelle ist der Unterschied zwischen einem Problem und einer Probleminstance, bei der eine konkrete Eingabe betrachtet wird. Eine einzelne Probleminstance kann durchaus einfach zu lösen sein, z. B. einen Hamilton-Pfad auf dem regelmäßigen Stadtplan von Manhattan zu finden. Bei dem Problem hingegen geht es um ein allgemeines Lösungsverfahren, das auf alle möglichen Probleminstance, also Stadtpläne jeglicher Art, anwendbar ist.

Des Weiteren unterscheiden Informatiker verschiedene Arten von Problemen. Klassisch sind Rechenprobleme, bei denen die Eingabe wie in einem Produktionsprozess schrittweise in die Ausgabe umgewandelt wird. Dabei müssen nicht notwendigerweise nur Zahlen verarbeitet werden; auch bei der Bearbeitung von Texten, Bildern oder Tonaufnahmen ergeben sich Probleme, für die Lösungen berechnet werden müssen. So ein Problem würde z. B. vorliegen, wenn wir die Anzahl der Kreuzungen auf einem Stadtplan berechnen lassen wollten. Entscheidend ist: Pro Eingabe gibt es eine eindeutig zugeordnete Ausgabe. Bei anderen Problemen sind

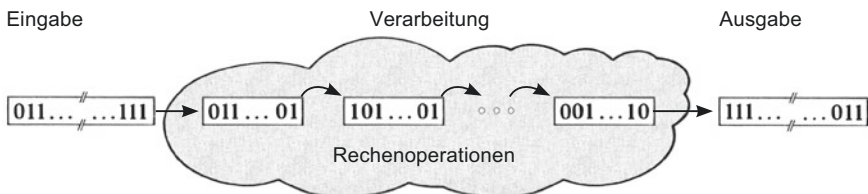


Abb. 8.3 Bei einem Problem in der Informatik geht es darum, ein Eingabemuster schrittweise in ein Ausgabemuster zu transformieren. (© Susanne Staubli, Eckart Zitzler)

jedoch mehrere verschiedene Ausgaben bei derselben Eingabe möglich. Das Hamilton-Pfad-Problem ist so eine Aufgabenstellung, denn unter allen möglichen Wegen auf dem Stadtplan kann es durchaus mehrere verschiedene Hamilton-Pfade geben. Und hier können sich jetzt verschiedene Problemvarianten ergeben, wie Bsp. 8.1 aufzeigt. Letztlich kommt es dabei auf die Frage an, die im Vordergrund steht.

Beispiel 8.1: Arten von Problemen

Das Hamilton-Pfad-Problem ist ein Beispiel für ein kombinatorisches Problem, bei dem Grundelemente – in diesem Fall die Fahrbahnen der Straßen – zu kombinieren sind und ein bestimmtes Arrangement – ein Hamilton-Pfad – gesucht ist. Der gesamte Raum der Kombinationsmöglichkeiten wird als Suchraum bezeichnet, dieser ist für jede Probleminstanz anders. Ein Element des Suchraums wird als Suchpunkt bezeichnet und die Suchpunkte, die gewünschte Kombinationen bzw. Hamilton-Pfade repräsentieren, werden als Lösungen bezeichnet.

Je nach Zielsetzung können wir nun verschiedene Problemvarianten formulieren (Abb. 8.4). Stellen wir nur die Frage danach, ob eine Lösung existiert, ob es also einen Hamilton-Pfad gibt, dann kann die Antwort 0 (nein) oder 1 (ja) sein. In diesem Fall und allgemein, wenn die Antwort lediglich ja oder nein ist, spricht man von einem Entscheidungsproblem. Eine Verallgemeinerung davon ist das Zählproblem, hier möchte man die Anzahl der möglichen Lösungen wissen: Wie viele verschiedene Hamilton-Pfade gibt es? Sind wir an einer konkreten Lösung, einem Hamilton-Pfad – das ist ja naheliegend – interessiert, dann sprechen wir von einem Suchproblem. Und häufig sucht man nicht nur irgendeine Lösung, sondern die beste. Das setzt voraus, dass man ein oder mehrere Kriterien hat, anhand derer man Lösungen bewerten und vergleichen kann. Beim Hamilton-Pfad-Problem könnte die Länge der Route so ein Kriterium sein, und dann haben wir es mit einem Optimierungsproblem zu tun.

Das Entscheidungsproblem lässt sich übrigens indirekt über die anderen Problemvarianten beantworten, genauso wie man das dazugehörige Suchproblem gelöst hat, wenn man das Optimierungsproblem bewältigen kann.

Die Unterscheidung der Problemvarianten ist insofern wichtig, als dass sie verschiedene Schwierigkeitsstufen repräsentieren. Wenn man sich also an einem Problem die Zähne ausbeißt, sollte man zuerst einmal die einfachste Variante in Angriff nehmen, bevor man gleich das Optimierungsproblem in Betracht zieht. Im Folgenden werden wir uns vor allem mit der Frage auseinandersetzen, wie man Berechnungsverfahren zur Lösung vorgegebener Probleme entwerfen kann. Wie wir gleich sehen werden, gibt es da verschiedene Strategien, die man bei der Konstruktion eines Programms einsetzen kann. Doch ein Problem lösen heißt viel mehr,

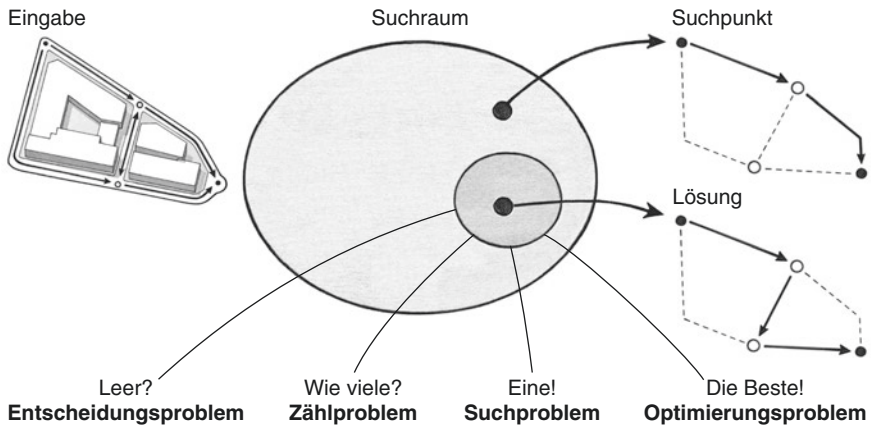


Abb. 8.4 Arten von Problemen in der Informatik. (© Susanne Staubli, Eckart Zitzler)

als ein Verarbeitungsverfahren zu entwerfen. Bevor man knobeln und brüten kann, muss man sich zunächst einmal überlegen: Was ist eigentlich das Berechnungsproblem, für das ich ein Programm entwickeln will? Wie sehen die Eingabedaten aus, wie die Ausgabedaten? Was genau ist zu tun? Das heißt, das Problem muss formal präzise beschrieben werden, und dieser Schritt – von physikalischen Größen in der realen Welt zu abstrahieren und die relevanten Aspekte in ein computerlesbares Format zu pressen – steht am Anfang des Problemlösens mittels Computern. Der Vorgang wird als Modellierung bezeichnet und die präzise Problemformulierung als Modell.

Der Weg ist das Ziel: Was es mit Algorithmen auf sich hat

Haben wir nun ein Problem und dieses entsprechend modelliert, so besteht die nächste Herausforderung darin, einen passenden Verarbeitungsvorgang zu definieren. Ja klar, ein Programm schreiben! Stimmt, darauf läuft es letztlich hinaus, doch zunächst einmal geht es darum, eine Idee davon zu bekommen, wie die Verarbeitung ablaufen könnte, und das Programm in groben Zügen zu skizzieren. Denn es ist häufig gar nicht so einfach, sich einen pfiffigen Informationsverarbeitungsvorgang auszudenken. Ich möchte Ihnen aufzeigen, welche Überlegungen bei einer solchen Ablaufkonstruktion angestellt werden, und beginne mit einem Beispiel, das Ihnen sicherlich schon während Ihrer Schulzeit – vielleicht in anderer Form – begegnet ist.

Nehmen wir an, Sie haben zwei verschieden lange Kasten Kuchen gebacken, die Sie nun restlos in möglichst wenige gleich große Stücke zerteilen wollen (siehe Abb. 8.5.) Vielleicht haben Sie penible Gäste eingeladen oder die Kuchen sind für einen Kindergeburtstag vorgesehen – und da darf auf keinen Fall ein Stück größer als das andere sein! Das Modell für dieses Problem sieht wie folgt aus: Die Eingabe besteht aus den Längen beider Kuchen – in Zentimetern angegeben – und die Ausgabe besteht aus der gesuchten Kuchenstückgröße. Wie ist hier der Weg von der Eingabe zur Ausgabe, wie berechnet man das? Die primitive, oder sagen wir besser brachiale Variante lautet: Ausgehend von der Länge des kürzeren Kuchens probieren wir systematisch in 1 cm-Schritten immer kleinere Stückgrößen aus, bis sich beide Kuchen restlos unterteilen lassen. Das funktioniert zwar, aber es geht auch wesentlich schneller, wie bereits Euklid, der als Vater der Geometrie bezeichnete griechische Mathematiker, vor über 2000 Jahren aufzeigte (Bsp. 8.2).

Beispiel 8.2: Den größten gemeinsamen Teiler finden

Wie ermitteln wir also für zwei in der Breite und Höhe identische, aber verschieden lange Kasten Kuchen die größte Stückgröße, sodass keine Reste übrig bleiben? Mathematisch gesehen handelt es sich hier um das Problem, den größten gemeinsamen Teiler zweier Zahlen zu finden.

Das Prinzip, welches dafür wiederholt angewendet wird, basiert auf einer einfachen Idee und lautet wie folgt: Zunächst legen wir beide Kuchen linksbündig nebeneinander, wie es in Abb. 8.6 oben dargestellt ist. Wenn beide Kuchenstücke gleich lang sind, dann ist die Kuchenlänge auch gleichzeitig die größte Stücklänge. Nehmen wir also an, ein Kuchen ist länger als der andere. In diesem Fall können wir den größeren Kuchen in zwei Teile zerschneiden, sodass der linke Teil der Größe des kleineren Kuchens entspricht, der rechte Teil ist quasi der Rest. Beide entstandenen Teile müssen restlos gemäß der Stückgröße zerlegt werden können. Den linken Teil des zerteilten größeren

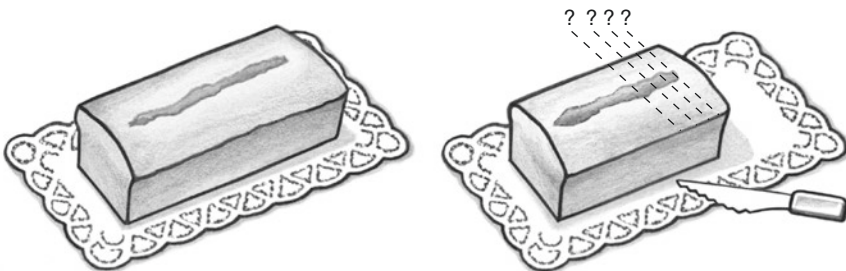
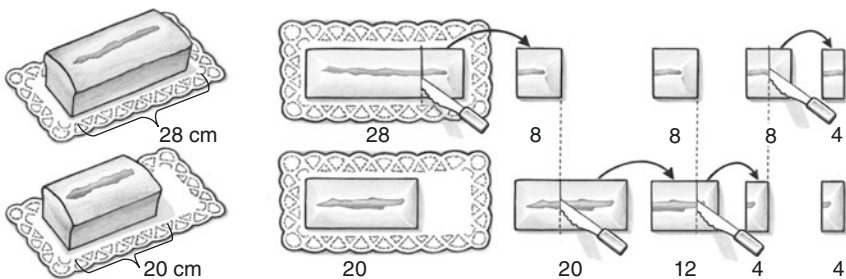


Abb. 8.5 Das Kuchenproblem: Zwei Kuchen sollen in möglichst wenige gleich große Stücke zerteilt werden. (© Susanne Staubli, Eckart Zitzler)

Kuchens legen wir nun zur Seite, übrig bleiben der rechte Teil des größeren Kuchens und der komplette kleinere Kuchen. Das soeben beschriebene Prinzip wenden wir nun erneut auf die zwei verbleibenden Kuchenstücke an, bis beide Kuchen die gleiche Länge aufweisen. Die Länge dieser Schlusstücke stellt die gesuchte maximale Stückgröße dar. Anschließend müssen wir noch alle in den Zwischenschritten beiseitegelegten Kuchenstücke gemäß der maximalen Stückgröße unterteilen. Fertig sind wir! Im konkreten Beispiel, in dem der kürzere Kuchen 20 cm lang ist und die gesuchte Stückgröße 4 cm beträgt, kommt Euklids Methode mit fünf Prüfschritten aus. Beim primitiven Verfahren hätten wir alle Zahlen von 20 bis 4 prüfen müssen.

Eine mögliche Umsetzung dieses Algorithmus in ein Programm sehen Sie in Abb. 8.6 rechts unten (formuliert in der Programmiersprache C). Dieses Programm berechnet die Funktion, die auf der linken Seite tabellarisch angedeutet ist.

Es kann also unterschiedliche Wege geben, um ein Problem zu lösen. Wir haben zwei möglichen Lösungsverfahren für das Kuchenproblem kennengelernt und dabei handelt es jeweils um einen Algorithmus. Dieser Begriff



Funktion

$c = \text{ggT}(a, b)$

Eingabe		Ausgabe
a	b	c
1	1	1
1	2	1
1	3	1
⋮	⋮	⋮
28	19	1
28	20	4
28	21	7
⋮	⋮	⋮

Programm

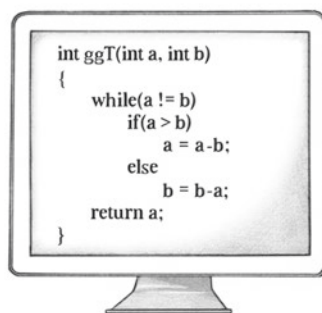


Abb. 8.6 Klassischer Euklidischer Algorithmus zur Identifizierung des größten gemeinsamen Teilers. (© Susanne Staubli, Eckart Zitzler)

hat nichts mit Logarithmus oder Rhythmus zu tun, sondern leitet sich vom Namen eines berühmten arabischen Gelehrten ab (Anm. 8.1). Ein Algorithmus definiert den Weg, wie aus den Eingabewerten Ausgabewerte berechnet werden können, und Weg heißt eine Folge von einzelnen, elementaren Schritten. In diesem Sinne bezeichnet er eine unmissverständliche Handlungsvorschrift, wie man vorgehen soll – zunächst einmal ganz unabhängig davon, ob wir die Berechnung selber vornehmen oder von einem Computer durchführen lassen wollen. In der Praxis muss er zudem bestimmte Eigenschaften erfüllen, z. B. sollte seine Beschreibung eine endliche Länge besitzen (man spricht von statischer Finitheit), er sollte in der Ausführung nur endlich viele Daten produzieren (diese Eigenschaft wird als dynamische Finitheit bezeichnet) und er sollte, wenn er mehrfach mit derselben Eingabe ausgeführt wird, stets das gleiche Ergebnis liefern (das nennt man Determiniertheit). Zumeist setzt man auch voraus, dass zu jedem Zeitpunkt der Ausführung eindeutig bestimmt ist, was die nächste elementare Rechenoperation ist (das nennt man Determinismus); allerdings gibt es auch Ausnahmen, beispielsweise die randomisierten Algorithmen, bei denen gezielt der Zufall eingesetzt wird.

Anmerkung 8.1: Woher der Algorithmus seinen Namen hat

Der Begriff des Algorithmus ist viel älter als die Computer. Das Wort ist abgeleitet vom Namen des Universalgelehrten Abu Dscha'far Muhammad ibn Musa al-Chwarizmi, der während der ersten Hälfte des 10. Jahrhunderts in Bagdad wirkte und als einer der bedeutendsten Mathematiker gilt. Sein Beiname al-Chwarizmi heißt so viel wie „der Choresmier“ und weist auf seine Herkunft aus dem gleichnamigen iranischen Volk hin. Der Weg vom Namen al-Chwarizmi zum Wort Algorithmus hat über die lateinische Übersetzung seines einflussreichen Algebra-Werks geführt – mit den Vorläufern des Wortes Algorithmus bezeichnete man die Kunst des Rechnens mit den arabischen Ziffern.

Algorithmen gab es schon weit vor dem 20. Jahrhundert, Euklids Methode ist nur ein Beispiel. Allerdings war der Begriff des Algorithmus mathematisch nicht klar gefasst, es fehlte eine genaue Definition. Das war der Grund, warum Alan Turing die Turing-Maschine erfand. Turings fiktives Gerät erlaubt es, den Begriff des Algorithmus zu formalisieren: Eine Berechnungsvorschrift heißt Algorithmus, wenn sie als Turing-Programm formulierbar ist und eine mit diesem Programm gefütterte Turing-Maschine für jeden Eingabewert den korrekten Ausgabewert berechnet. Erst später wurde also der Algorithmus-Begriff an Maschinen gekoppelt. Allerdings meinen wir mit Algorithmus eigentlich nur das Verfahren. Denn ein

Programm für einen bestimmten Computer ist nicht der Algorithmus, es stellt nur dessen spezifische Repräsentation dar. Dieses Programm sieht ggf. ganz anders aus als das Programm für die Turing-Maschine und doch ist der zugrunde liegende Algorithmus, sprich das Berechnungsverfahren, dasselbe. Der Algorithmus stellt also die Idee dar, das Konzept der Berechnung. Man könnte auch sagen: Ein Algorithmus ist eine Abstraktion von konkreten Programmen für verschiedene Computer, die alle auf die gleiche Art und Weise die gleiche Funktion berechnen.

Wenn wir also ein Informatik-Problem mit dem Computer lösen wollen, so müssen wir einerseits einen Lösungsweg, einen Algorithmus zur Berechnung finden, und andererseits müssen wir den Algorithmus anschließend noch in ein ausführbares Programm umsetzen. Wie wir beim Kuchen-Problem gesehen haben, können verschiedene Wege zum Ziel führen, nicht alle sind gleich geeignet. Wir werden uns später noch damit beschäftigen, was eigentlich gute Algorithmen sind und wodurch sich ein weniger geeigneter auszeichnet. Davor schauen wir uns Lösungsmuster an, die sich im Laufe der Zeit herausgeschält haben und von Informatikerinnen eingesetzt werden, wenn sie einen Algorithmus entwerfen. Es handelt sich wie bei den Verhaltensmustern um Schablonen, die auf das jeweilige Problem angepasst werden. Natürlich ist nicht jede Schablone für jedes Problem geeignet. Wir werden die prominentesten Schablonen anschauen und sie jeweils an konkreten Problemen illustrieren.

Probieren geht über Studieren

Wenn wir überhaupt keine Idee haben, wie sich aus der Eingabe die gesuchte Ausgabe konstruieren lässt, hilft eigentlich immer die Strategie des Ausprobierens. Die erste Lösungsvariante für das Kuchenproblem beruht auch auf diesem Prinzip. Dazu legen wir zunächst eine geeignete Menge von Lösungskandidaten fest; beim Kuchenproblem waren es die Zahlen von 1 bis zur Länge des kürzeren Kuchens. Anschließend probieren wir alle Lösungsmöglichkeiten systematisch durch, bis wir schließlich eine gültige Lösung gefunden haben. Diese Lösungsstrategie wird daher auch als „Versuch und Irrtum“ bezeichnet. Im Englischen wird sie auch *trial and error* oder *brute force* genannt, weil man im schlimmsten Fall alle Möglichkeiten durchprobieren muss und die stumpfe, rohe Gewalt Vorrang hat vor dem klugen, gewitzten Vorgehen.

Natürlich muss man sich hierbei überlegen, wie die einzelnen Lösungskandidaten systematisch erzeugt werden können, und bereits das

kann manchmal schwierig sein. Gleichzeitig sollte die Kandidatenmenge möglichst klein sein. Schauen wir uns das Ganze mal beim bereits bekannten Hamilton-Pfad-Problem an, dann wird klarer, auf was ich hinauswill. In Abb. 8.7 sehen Sie oben den bereits bekannten Stadtplan in leicht abgewandelter Form. Die einfachste Variante, das Prinzip „Versuch und Irrtum“ hierbei umzusetzen, besteht darin, alle Permutationen bzw. Kombinationen der Fahrbahnen durchzuprobieren, d. h. die Fahrbahnen durchzunummerieren, hintereinander als Zahlenkette zu schreiben und die Reihenfolge innerhalb der Zahlenkette durch das Vertauschen von Nummern zu verändern. In diesem Fall werden sehr viele Lösungskandidaten generiert, die wir eigentlich von vornherein ausschließen können: wenn sie nicht beim Startpunkt beginnen oder wenn zwei Fahrbahnen aufeinander folgen, die gar nicht miteinander über eine Kreuzung verbunden sind. Geschickter ist es, vom Start ausgehend möglichen Fahrbahnen zu folgen, wie Bsp. 8.3 erläutert. In diesem Fall ist die Kandidatenmenge kleiner, denn unsinnige Routen werden erst gar nicht betrachtet.

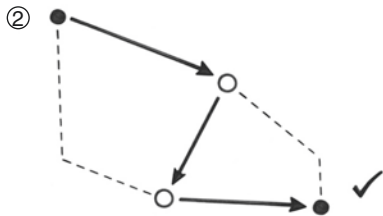
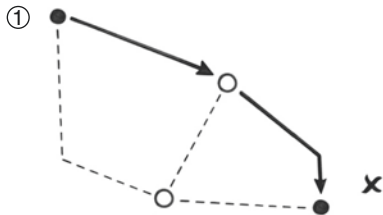
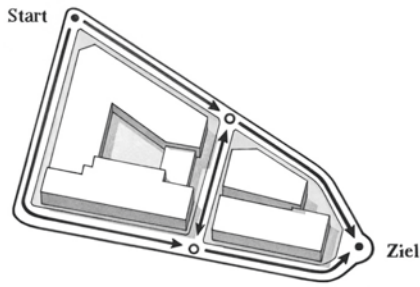
Beispiel 8.3: Versuch und Irrtum

In Abb. 8.7 ist auf der linken Seite illustriert, wie vom Startpunkt ausgehende Routen systematisch durchprobiert werden können. Beginnend beim Ausgangspunkt wählen wir die erstbeste Fahrbahn, folgen ihr bis zur nächsten Kreuzung, treffen auch dort wieder eine Wahl usw. Treffen wir auf den Zielpunkt, gilt es zu prüfen, ob alle Kreuzungen besucht wurden. Bei dem ersten Lösungskandidaten in Abb. 8.7 ist das nicht der Fall, deswegen muss die Route angepasst werden. Wir gehen zur vorherigen Kreuzung zurück und probieren von dort aus eine andere Fahrbahn; falls wir jedoch bereits alle Abzweigungen an dieser Kreuzung durchprobiert haben, laufen wir noch weiter zurück zur vorletzten Kreuzung und probieren dort alternative Wege. Das nennt man Rückverfolgung, im Englischen *backtracking*. Falls es einen Hamilton-Pfad gibt, so wird man diesen nach einer gewissen Zeit auch finden (Schritt 2 in Abb. 8.7).

Im Detail ist es natürlich schon noch etwas komplizierter, als es eben skizziert wurde. Wir müssen uns merken, welche Route wir gerade genommen haben, und die entsprechenden Fahrbahnen und Kreuzungen markieren – sonst laufen wir evtl. im Kreis herum. Darüber hinaus müssen wir Buch darüber führen, welche Routen wir bereits ausprobiert haben – auch das lässt sich über geschicktes Markieren lösen.

Diese Methode lässt sich auch bei Optimierungsproblemen anwenden, wenn wir nicht nur eine, sondern die beste Lösung suchen, z. B. den kürzesten Hamilton-Pfad. Das Vorgehen ist im Prinzip das gleiche, nur können

Versuch und Irrtum



Verzweige und begrenze

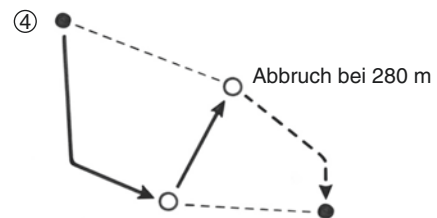
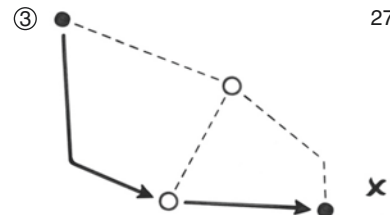
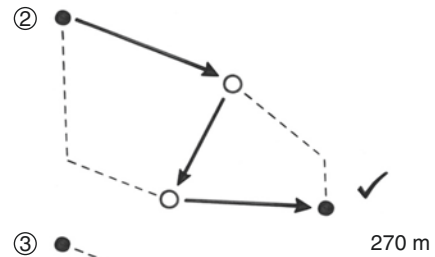
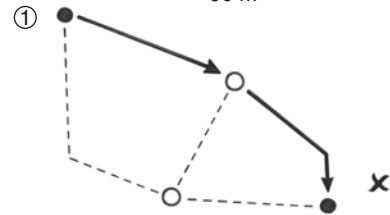
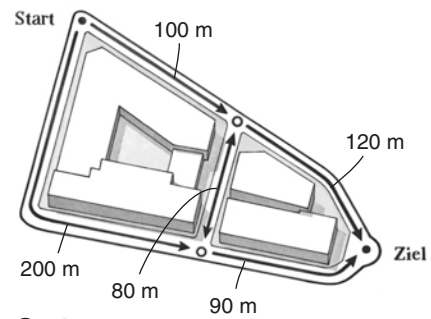


Abb. 8.7 Die Lösungsstrategien „Versuch und Irrtum“ (rechts) und „Verzweige und begrenze“ anhand des Hamilton-Pfad-Problems illustriert. (© Susanne Staubli, Eckart Zitzler)

wir beim Auffinden einer gültigen Lösung nicht aufhören – vielleicht gibt es ja noch eine bessere! In diesem Fall bleibt uns nicht anderes übrig, als effektiv alle Kombinationsmöglichkeiten auszuprobieren, und davon kann es sehr viele geben. Aber auch hier gibt es Möglichkeiten, unnötige Lösungswege gar nicht erst weiterzuverfolgen. Haben wir bereits eine oder mehrere Rundreisen gefunden, so liefert das einen Anhaltspunkt, welche Weglänge maximal notwendig ist. Daher können wir beim weiteren Ausprobieren aufhören, sobald eine Teilstrecke bereits die Länge der besten bereits gefundenen Lösung erreicht. Wir können unsere Suche hier beschneiden, deswegen heißt dieses Prinzip auch „Verzweige und begrenze“ oder im Englischen *branch and bound*. Bsp. 8.4 erläutert diese Strategie.

Beispiel 8.4: Verzweige und begrenze

Nehmen wir nun an, dass wir noch die Längen der Fahrbahnen in Betracht ziehen, die Distanzen sind auf der rechten Seite in Abb. 8.7 angeschrieben. In diesem Fall gehen wir gleich vor wie auf der linken Seite, um gültige Routen zu finden, doch nach dem Finden des ersten Hamilton-Pfads hören wir nicht auf, sondern suchen weiter. Wir gehen zurück zum Start und probieren letztlich dort die untere Verzweigung, die wir weiter ergänzen, bis wir am Ziel sind. Die dritte Routenvariante verwerfen wir direkt, weil nicht alle Kreuzungen besucht wurden. Schließlich wählen wir bei der unteren Kreuzung vor dem Ziel die Fahrbahn, die auf dem Stadtplan nach oben führt. Diese Route können wir an dieser Stelle bereits abbrechen, weil die vom Start aus zurückgelegte Teilstrecke 270 m beträgt, die vorher gefundene Lösung jedoch nur 250 m Wegstrecke benötigt; d. h., diese Teilstrecke kann unmöglich zu einer kürzeren Route ergänzt werden. Die Wahl der Fahrbahnen an jeder Kreuzung entspricht dem „Verzweige“, der Abbruch, wenn eine Teilstrecke bereits zu lang ist, dem „Begrenze“.

Den Strategien „Versuch und Irrtum“ sowie „Verzweige und begrenze“ ist gemeinsam, dass der gesamte Suchraum systematisch durchkämmt wird. Je nach Problem und Problemistanz kann dieser Raum riesig sein – so immens groß, dass selbst die leistungsfähigsten Computersysteme Jahre brauchen würden, um ihn komplett zu durchsuchen, wie es z. B. beim Ausprobieren von kryptografischen Schlüsseln der Fall sein kann. Der Vorteil der Methoden ist ihre generelle Einsetzbarkeit, und tatsächlich bleibt einem manchmal nichts anderes übrig. Doch wenn irgend möglich, versucht man Algorithmen zu entwickeln, die nicht Lösungskandidaten ausprobieren, sondern Lösungen direkt zusammenbauen.

Aus kleinen Lösungen werden große

Konstruktionsstrategien eignen sich dann, wenn zulässige Lösungen leicht zu generieren sind – anders als beim Hamilton-Pfad-Problem, bei dem die Hauptschwierigkeit darin besteht, überhaupt eine gültige Route zu finden. Ich werde Ihnen zwei Ansätze vorstellen und sie Ihnen anhand einer Spielart des Rucksackproblems erläutern. Bei dieser Variante geht es darum, Goldgewichte in dem Rucksack zu platzieren – zugegeben eine etwas unrealistische Situation –, sodass ein bestimmtes Gewicht nicht überschritten wird und gleichzeitig der Wert des Inhalts maximal ist. In Abb. 8.8 sehen Sie eine Probleminstance mit vier Goldklumpen, jeder wiegt ein oder mehrere Kilogramm, die maximale Zuladung des Rucksacks beträgt 10 kg.

Die erste Lösungsstrategie ist eine für Eilige. Wir gehen die Goldgewichte in einer bestimmten Reihenfolge durch und fügen das entsprechende Gewicht hinzu, wenn dadurch nicht das zulässige Gesamtgewicht überschritten wird (Bsp. 8.5). Die Gier kommt quasi vor dem Nachdenken: Wir packen ein, was uns in die Hände kommt. Deswegen wird dieses Vorgehen auch als gierige Strategie – im Englischen *greedy strategy* – bezeichnet. Der Vorteil liegt in der Geschwindigkeit, denn in wenigen Schritten ist die Lösung zusammengestellt, und das funktioniert auch, wenn aus einer riesigen Menge unterschiedlicher Gewichte auszuwählen ist. Der Nachteil ist jedoch: Die so konstruierte Lösung ist zumeist nicht optimal, sondern nur eine Annäherung an die beste Auswahl. Es handelt sich hierbei um eine Heuristik – ein Verfahren, das auf Annahmen und Vermutungen beruht und sich in der Praxis bewährt hat, aber nicht notwendigerweise stets gute Lösungen generiert.

Beispiel 8.5: Die gierige Strategie

Die gierige Strategie setzt eine Lösung schrittweise zusammen, und in jedem Konstruktionsschritt wird eine Entscheidung gefällt, die in späteren Schritten nicht revidiert werden kann. Beim Rucksackproblem lässt sie sich relativ einfach einsetzen, z. B. kann es geschickt sein, immer das größte Gewicht zu wählen, das noch hineinpasst. So kommen wir bei der in Abb. 8.8 illustrierten Probleminstance zu einer Auswahl von 8 kg; die optimale Auswahl bestände jedoch aus dem 6 kg- und dem 4 kg-Goldgewicht. Das Vorgehen ist häufig anwendbar, jedoch nicht bei allen Problemen. Beim Hamilton-Pfad-Problem z. B. ist es nicht möglich, vom Startpunkt aus die nächste Fahrbahn zu wählen, ohne die nachfolgenden Entscheidungen mit einzubeziehen.

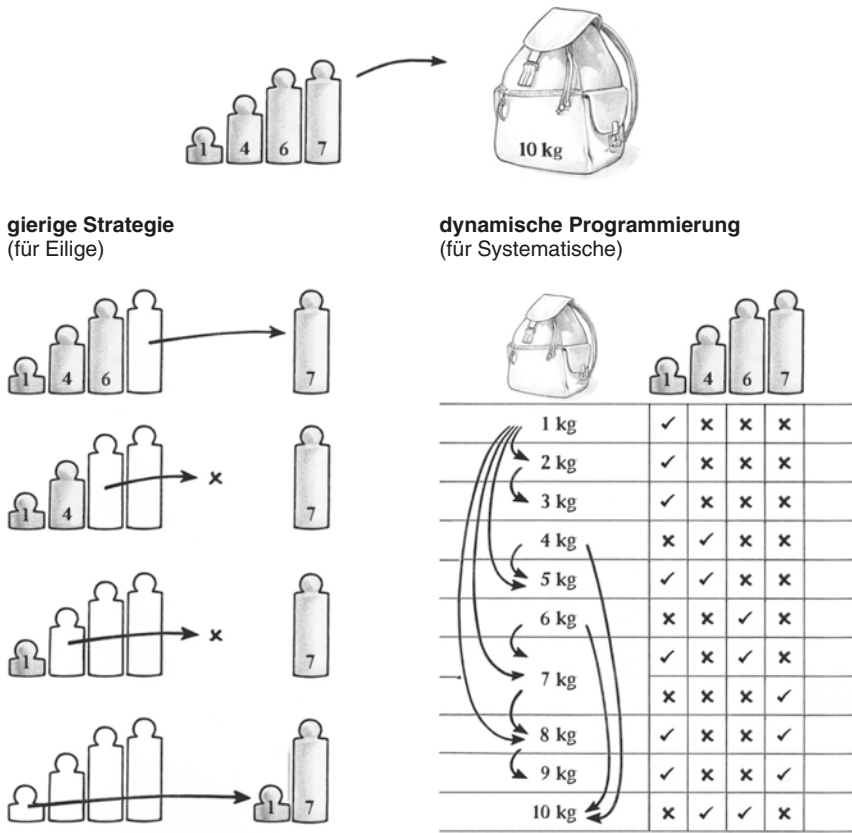


Abb. 8.8 Das Rucksackproblem: Links eine gierige Strategie, rechts die Methode der dynamischen Programmierung. (© Susanne Staubli, Eckart Zitzler)

Die zweite Konstruktionsmethode ist aufwändiger, erzeugt dafür aber optimale Lösungen. Sie ist der Ansatz für Systematische, die vor dem Einpacken eine regelrechte Auslegeordnung vornehmen, alles genau durchplanen und zum Schluss effizient zusammenpacken. Die Idee ist, eine optimale Lösung schrittweise aus optimalen Teillösungen zusammenzusetzen. Beim Rucksackproblem heißt das: Wir schrauben die Gewichtsgrenze sukzessive hoch, bis wir bei der effektiv vorgegebenen Gewichtsgrenze sind. Für jede Gewichtsgrenze können wir die beste Lösung bestimmen, indem wir auf die besten Lösungen der vorher betrachteten Gewichtsgrenzen zurückgreifen. Und so kommen wir schließlich zur optimalen Lösung, die ein Gesamtgewicht von 10 kg erreicht. Im Detail können Sie den Ablauf in Bsp. 8.6 nachlesen. Die Methode nennt sich übrigens dynamische

Programmierung; die Begrifflichkeit ist etwas verwirrend und hat historische Gründe, doch das ist nun mal der Fachterminus.

Beispiel 8.6: Die systematische Strategie

Betrachten Sie in Abb. 8.8 die rechte Seite. Wir beginnen zunächst mit der kleinsten Gewichtsgrenze von einem Kilogramm. In diesem Fall ist die Auswahl des Goldstücks mit einem Gewicht von 1 kg die optimale Lösung; dies merken wir uns in einer Tabelle. Jetzt nehmen wir die nächsthöhere Gewichtsgrenze von 2 kg: Die könnten wir erreichen, wenn es entweder ein 2 kg-Goldstück gäbe (gibt es aber nicht) oder zwei verschiedene optimale Lösungen für die Gewichtsgrenzen mit einem Kilogramm (es gibt aber nur eine). Aus diesem Grund übernehmen wir für die 2 kg-Grenze einfach die beste Lösung von der 1 kg-Grenze. Bei der 3 kg-Gewichtsgrenze prüfen wir wiederum für alle Möglichkeiten, wie diese vollständig erreicht werden kann: durch ein 3 kg-Goldstück oder durch die Kombination der besten Lösungen für die 1 kg- und die 2 kg-Gewichtsgrenze (in diesem Fall sind diese identisch und nicht kombinierbar). Die beste Lösung ergibt sich als die beste Lösung bei der 1 kg-Gewichtsgrenze.

Das geht nun immer so weiter. Schauen wir uns noch die 7 kg-Grenze an: Wiederum prüfen wir alle Möglichkeiten, eine 7 kg-Lösung zusammenzusetzen, mithilfe der optimalen Lösungen zu den kleineren Gewichtsgrenzen, also 1 kg zusammen mit 6 kg, 2 kg zusammen mit 5 kg und 3 kg zusammen mit 4 kg. Tatsächlich gibt es eine solche Lösung, wenn wir die 3 kg- und 4 kg-Gewichtsgrenzlösungen miteinander kombinieren; das tragen wir in der Tabelle ein. Darüber hinaus gibt es noch eine zweite optimale Lösung, nämlich das Einzelstück mit 7 kg Gewicht, auch das tragen wir in die Tabelle. Wenn wir schließlich zur 10 kg-Gewichtsgrenze kommen, können wir die optimale Lösung (das 6 kg- und das 4 kg-Goldstück) leicht identifizieren.

Der Aufwand, der bei der dynamischen Programmierung betrieben werden muss, ist in der Regel wesentlich kleiner als bei der Methode des „Verzweige und begrenze“. Beim Rucksackproblem hängt dieser Aufwand vom Gewichtsspektrum ab: Je größer und unterschiedlicher die Gewichte sind, desto mehr Schritte muss das Verfahren durchlaufen. In der Fachsprache handelt es sich um einen pseudopolynomiellen Algorithmus, der an sich effizient arbeitet, doch ineffizient werden kann, wenn die zu verarbeitenden Zahlen – in diesem Fall die Gewichtsgrenze – zu groß werden.

Große Probleme in kleine zerlegen

Die Idee der dynamischen Programmierung, nämlich vom Kleinen zum Großen zu schreiten und größere Lösungen aus kleineren (Teil)Lösungen zusammenzusetzen, wird auch bei einer weiteren Lösungsstrategie umgesetzt,

dem „Teile und herrsche“-Prinzip. Hier wird ein Problem in mehrere handhabbare Teilprobleme zerlegt, diese werden separat gelöst und die Teilergebnisse schließlich zu einer Gesamtlösung zusammengesetzt. Das Paradebeispiel für „Teile und herrsche“ in der Informatik ist das Sortieren: Große Kartenstapel lassen sich einfacher sortieren, wenn man sie zunächst in kleinere Stapel aufteilt – beim Mischen ist das ja genauso. Bsp. 8.7 erläutert Ihnen ein Verfahren, das als Mergesort bekannt ist.

Beispiel 8.7: Teile und herrsche

In Kap. 5 war davon die Rede gewesen, die im Speicher abgelegten Tierkarten zu sortieren, damit die binäre Suche durchgeführt werden kann. Jetzt zeige ich Ihnen, wie das geht.

Zunächst teilen wir den Kartenstapel beliebig in zwei Hälften auf. Die Hälften werden wiederum zerlegt in zwei Stapel usw., bis nur noch eine Karte pro Teilstapel übrig bleibt – jeder Teilstapel ist für sich sortiert. Jetzt fügen wir die Teilstapel wieder paarweise zusammen. Dabei können wir ausnutzen, dass die beiden Teilstapel bereits sortiert sind, wir müssen immer nur die beiden linken Enden der Teilstapel betrachten und die jeweils alphabetisch kleinere Karte rechts an die Gesamtfolge anhängen. In Abb. 8.9 ist das Prinzip visualisiert, allerdings wird nur ein Zerlegungsschritt dargestellt.

Bei diesem Beispiel haben wir die Problemlösungsstrategie, das Zerlegen und das Zusammenfügen, ebenso bei den entstandenen Teilproblemen eingesetzt. Dieses Vorgehen – ein Prinzip wiederholt auf sich selbst anzuwenden – nennt man übrigens Rekursion. Im Alltag tritt sie uns in Form verschiedener Phänomene entgegen, z. B. wenn zwei Spiegel frontal zueinander aufgestellt sind und sich das Spiegelbild im Spiegelbild ins Unendliche fortpflanzt; oder bei der akustischen Rückkopplung, wenn ein Mikrofon aufnimmt und verstärkt, was es selbst in einen Lautsprecher einspeist. In der Informatik meint man damit die Rückführung eines Problems oder Verhaltens auf sich selbst, und diese Lösungsstrategie kommt häufig zur Anwendung. Sie wird in Bsp. 8.8 an einem weiteren Kuchenproblem erläutert, das auch als Knobelspiel unter dem Namen „Türme von Hanoi“ bekannt ist. Es handelt sich insofern um ein besonderes Problem, als hier ein Ablauf die Lösung ist: In welcher Reihenfolge müssen Kuchen verschoben werden?

Beispiel 8.8: Rekursion

Betrachten Sie das folgende, in Abb. 8.10 illustrierte Szenario: Eine vierstöckige Hochzeitstorte ist auf eine andere Kuchenplatte zu befördern. Weil sie sehr filigran ist, muss jede Schicht einzeln abgetragen und versetzt werden. Kein Problem, werden Sie sagen, wir nehmen die Schichten einzeln ab, legen

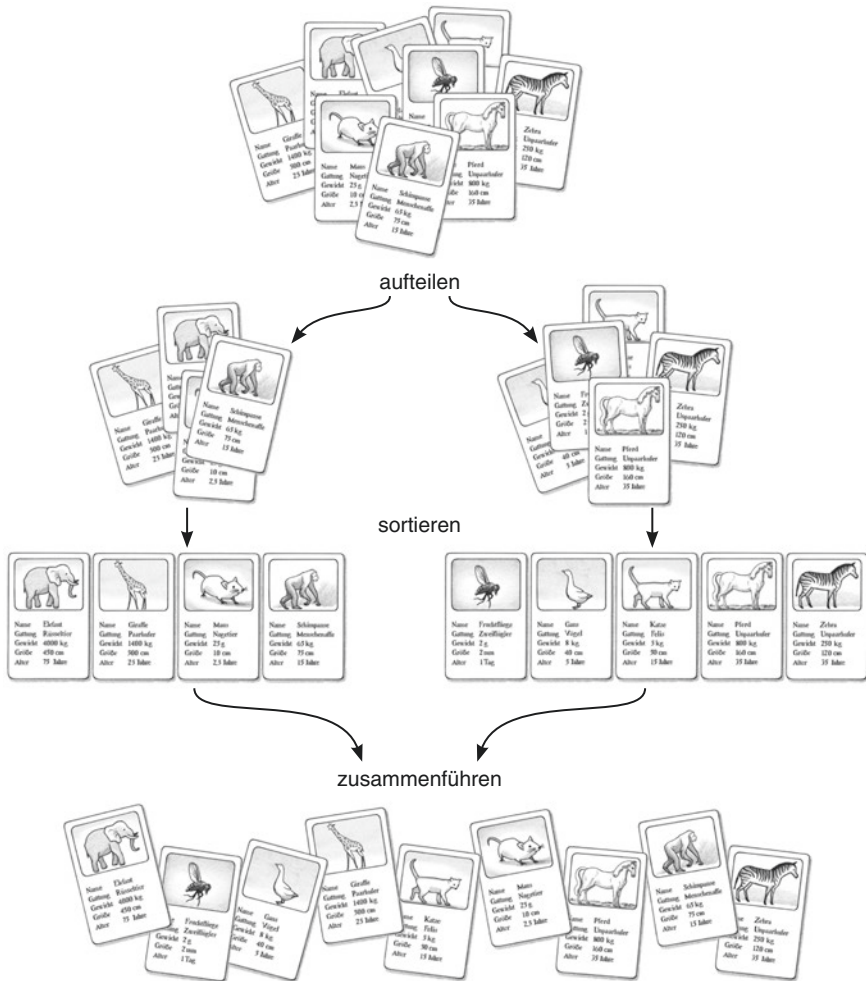


Abb. 8.9 Der Mergesort-Algorithmus, der das Prinzip „Teile und herrsche“ nutzt. (© Susanne Staubli, Eckart Zitzler)

sie zur Seite und bauen die Torte auf der anderen Seite wieder auf. Die zusätzliche Schwierigkeit besteht nun darin, dass es nur eine zusätzliche Kuchenplatte zum Zwischenlagern gibt, nicht drei. Das heißt, wir müssen die Schichten auf den Zwischenlagern übereinanderstapeln. Da kommt die nächste Schwierigkeit hinzu: Weil sonst die Tortenschichten zerdrückt würden, darf eine Tortenschicht immer nur auf einer größeren Schicht oder einer Kuchenplatte abgelegt werden. Und auf einmal ist das Problem nicht mehr so einfach zu lösen.

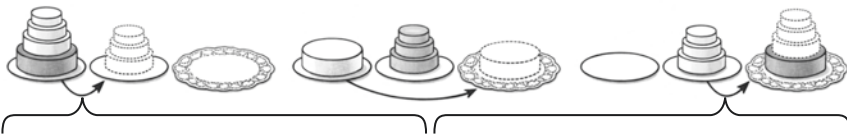
Wenn wir jedoch das Prinzip der Rekursion benutzen, dann wird es einfach. Wir müssen ja nur die drei oberen Schichten auf die mittlere Kuchenplatte platzieren, anschließend die unterste Schicht auf die rechteste

Kuchenplatte umverlagern und zum Schluss die drei restlichen Schichten von der Zwischenablage wieder auf die unterste Schicht hieven. Ja, aber wie bekommen wir die drei Schichten in die Mitte? Nach dem gleichen (rekursiven) Prinzip. Wir müssen nur die zwei obersten Schichten nach ganz rechts verlagern, die zweitunterste Schicht in die Mitte legen und anschließend die zwei obersten Schichten wieder aufsetzen. Und die obersten zwei Schichten bewegen wir nach dem gleichen Prinzip. Wenn Sie sich die Idee in Abb. 8.10 vor Augen führen, dann sehen Sie, dass die konkrete Idee simpel ist, aber die konkrete Ausführung natürlich ganz schon kompliziert, weil die Rekursion immer wieder eingesetzt wird.

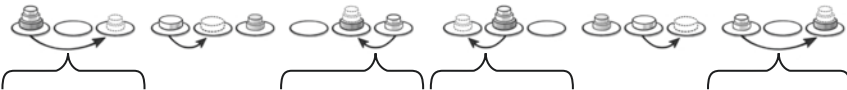
Es gibt noch viele weitere Problemlösungsstrategien, die Informatiker entwickelt haben und beim Entwurf von Algorithmen einsetzen. Vielfach werden verschiedene Entwurfsschemata auch miteinander kombiniert. Wir können das Thema hier nicht weiter vertiefen, es genügt, wenn Sie einen Einblick haben, wie Algorithmen konzipiert werden und welche Grundlagen dafür zur Verfügung stehen. Und nur für den Fall, dass Sie meinen, Kuchenprobleme seien Spielprobleme, die nur bei Kindergeburtstagen oder Hochzeiten eine Rolle spielen: Sie können so vertrackt sein und für die Informatik spannende Fragestellungen liefern, dass sich Wissenschaftler weiterhin ausführlich mit Kuchenschneide-Algorithmen auseinandersetzen (Robertson und Webb 1998).

obere Rekursionsstufe

- ① oberen Turm wegnehmen ② Fundament verschieben ③ oberen Turm aufsetzen



mittlere Rekursionsstufe



untere Rekursionsstufe

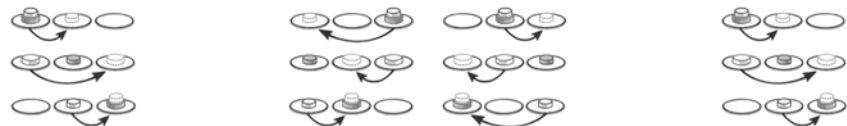


Abb. 8.10 Ein weiteres Kuchenproblem: Hochzeitstorten verschieben. (© Susanne Staubli, Eckart Zitzler)

Vom Rechenknecht zum Schlaumeier

Bislang haben wir den Computer als Rechenknecht kennengelernt, der stupide eine Sequenz von Rechenoperationen abarbeitet. Zwar lassen sich so selbst komplexe Probleme lösen, doch die durch das Programm codierte Lösungsstrategie ist der Intelligenz eines Menschen entsprungen, der den Ablauf erdacht und formuliert hat. Die Idee, auch Computer mit Intelligenz auszustatten – was immer das im Einzelnen heißt, denn eine allgemeingültige Definition der Intelligenz gibt es nicht –, hatte schon die Pioniere beschäftigt. Alan Turing beispielsweise ersann 1950 einen nach ihm benannten Test: Können wir in einem Chat herausfinden, ob auf der Gegenseite ein Computer Antworten generiert oder ein Mensch? Falls nein, wäre die Intelligenz des Computers bewiesen, so Turings Ansicht. So umstritten der Turing-Test auch ist, er spiegelt eine Frage wider, die die Menschheit schon lange beschäftigt, nämlich ob sich Maschinen intelligent verhalten können. Mit der Zeit etablierte sich ein eigenes Forschungsgebiet – die Künstliche Intelligenz –, das dieser Frage nachgeht und den Computer als Schlaumeier ins Blickfeld rückt. Wir werden dieses Gebiet im Folgenden in Augenschein nehmen und dabei drei Themenfelder streifen, nämlich Lernen, Spielen und Sprache.

Vom stereotypen zum intelligenten Verhalten

Doch versuchen wir zunächst, diese Themenfelder biologisch zu verorten. Wenn wir von Intelligenz sprechen, dann steht die Anpassungsfähigkeit des Verhaltens im Vordergrund. Es geht um die Fähigkeit, auch in neuen Problemsituationen adäquate Lösungen zu finden, und das möglichst effizient. Das setzt nicht zwingend ein Nervensystem voraus, auch Pflanzen können auf veränderte Umweltbedingungen reagieren, doch die Entwicklung von Gehirnen hat immer komplexere Anpassungsleistungen ermöglicht. Bei den ersten tierischen Mehrzellern spielte das Nervensystem zunächst einmal die Rolle des Mittlers: Von den spezialisierten Sinneszellen mussten Informationen zu den spezialisierten Bewegungszellen transportiert werden, damit sich das Lebewesen koordiniert als Ganzes verhalten konnte, in Abhängigkeit von der Umwelt. Später bildeten sich größere Ansammlungen von Nervenzellen – ein erster Schritt Richtung Gehirn. Allerdings repräsentierten diese ersten Gehirne unveränderliche Verhaltensprogramme, das Verhaltensrepertoire war fest vorgegeben. Mit den Wirbeltieren erreichte die Anpassungsfähigkeit eine neue Stufe. Mit ihren durch eine Schädelkapsel

geschützten Gehirnen entwickelten sie die Fähigkeit, aus Erfahrungen zu lernen und Eindrücke blitzschnell mit Erinnerungen abzugleichen. Gedächtnis und Lernen flexibilisierten das Verhalten und vergrößerten den Handlungsspielraum. Wir Menschen haben es schließlich auf die Spitze getrieben, indem wir darüber reflektieren, wie wir am besten lernen; wir sind also darüber hinaus in der Lage, unsere Lernstrategien anzupassen.

Neben dem Lernen bildeten sich bei einigen Wirbeltieren weitere kognitiven Fähigkeiten heraus, die es ihnen erlauben, Handlungsmöglichkeiten im Geiste durchzuspielen und zu bewerten, auf dieser Basis zwischen Optionen zu entscheiden und ihr Verhalten zu planen. Eng damit verknüpft sind auch Aspekte wie Neugierde, Kreativität und Spielen. Es scheint im Wesen intelligenter Organismen zu liegen, dass sie sich immer wieder in Situationen erproben und Verhaltensweisen trainieren müssen. Eine Vermutung ist, dass der Wettbewerb zwischen Raubtieren und ihrer Beute ein entscheidender Faktor war, warum sich immer größere Gehirne und intelligentere Verhaltensweisen herausgebildet haben. Es war sozusagen ein evolutionäres Wettrüsten. Je klüger die Räuber wurden, desto geschickter mussten sich die Gejagten zu helfen wissen, und je schlauer die Beutetiere wurden, desto mehr mussten sich die Räuber einfallen lassen. Lange Zeit meinte man auch, dass Tiere nicht denken können, vielleicht mit Ausnahme der Schimpansen, die uns Menschen sehr nahe stehen. Doch mittlerweile ist unbestritten, dass gewisse Tierarten denken können, und man fragt sich eher: Was denken sie?

Eindrucksvoll sind z. B. die Fähigkeiten der Orcas. Sie haben verschiedene Strategien entwickelt, um ihre Beute zu fangen, und zwar immer in einer sozialen Gruppe (siehe Abb. 8.11). Sie erzeugen Wellen und bringen so Eisschollen ins Wanken; Robben, die sich auf den Schollen befinden, werden hinuntergeschleudert und so zur leichten Beute. Die Orcas kesseln mit Luftsäulen Heringsschwärme ein, während jedes Gruppentier sich einmal bedienen darf. Darüber hinaus beherrschen sie auch sehr waghalsige Manöver, bei denen sie ihren Körper an den Strand wuchten, um in der Brandung spielende Robben zu erwischen. Es ist auch schon beobachtet worden, dass sie Wale oder Delfine angreifen, indem sie ihre Schnauzen mit Wucht in die Körper der Opfer rammen. So brutal diese Beispiele für uns Menschen anmuten mögen, so faszinierend sind sie aus dem Blickwinkel der Verhaltensforschung. Denn diese Verhaltensweisen sind nicht angeboren, sondern werden von Generation zu Generation weitergereicht und müssen individuell erlernt werden.

Das Beispiel der Orcas weist auf einen weiteren Faktor hin, der bei der Evolution der Gehirne vermutlich großen Einfluss gehabt hat: das Zusammenleben in sozialen Gruppen. Wenn Orcas im Rudel jagen und

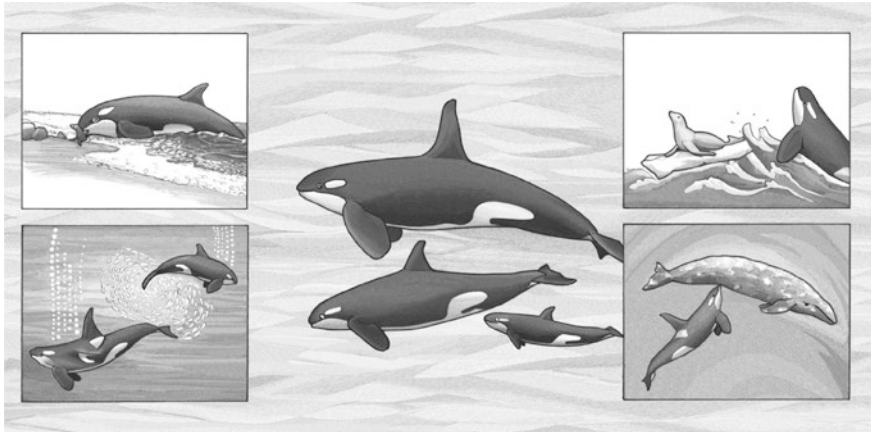


Abb. 8.11 Orcas und ihre Jagdstrategien. (© Susanne Staubli, Eckart Zitzler)

Jungtiere die ausgeklügelten Taktiken von ihren Müttern erlernen, so erfordert das eine differenzierte Verständigung. Dazu gibt es eine aus verschiedenen Lauten bestehende Sprache, die ebenfalls erlernt werden muss. Man weiß zudem, dass in sozialen Gruppen lebende Tiere, u. a. Elefanten, Delfine und Affen, vielschichtigste Kommunikationsformen entwickelt haben – auch der Mensch ist so ein Tier. Er hat die Kunst der Kommunikation am weitesten vorangetrieben, mit seinen einzigartigen Sprachsystemen kann er beliebige Informationen, nicht nur vereinbarte Nachrichten, codieren und übermitteln. Es ist naheliegend, dass das Sozialverhalten und die komplexen Interaktionen, die das Zusammenleben in Gruppen mit sich brachte, unsere Hirnentwicklung stark beeinflusst haben. Unsere Sprache und das Sprechenlernen erforderte und förderte unsere Intelligenz. Ja, mittlerweile sind wir sogar in der Lage, so komplizierte Maschinen wie Computer zu konstruieren und zu fertigen, selbst wenn oder gerade weil unzählige Menschen daran beteiligt sind.

Intelligenz hat natürlich viele Facetten, aber das Lernen und die Anwendung von Erlerntem, das Durchspielen und Bewerten von Handlungsoptionen, wie es bei Strategiespielen zur Anwendung kommt, und die Kommunikation über eine Sprache sind sicherlich zentrale Aspekte, wenn wir an die kognitive Leistungsfähigkeit des Menschen denken. Deswegen sind sie auch von so großem Interesse für Computer- und andere Wissenschaftler, die die Möglichkeiten der automatisierten Informationsverarbeitung erforschen. Beginnen wir mit dem ersten Aspekten, dem Lernen.

Lernen und Anpassungsfähigkeit

Mit Lernen meinen wir, dass vergangene Erfahrungen das zukünftige Verhalten beeinflussen. Damit geht zweierlei einher: Wir übertragen Verhaltensregeln erfolgreich auf neue, unbekannte Situationen und wir sind in der Lage, selbst in identischen bzw. ähnlichen Situationen unser Verhalten aufgrund von Erfahrungen zu verändern. Übertragen auf das einfache Verarbeitungsschema eines Computers heißt dies, dass sich die Zuordnung der Eingabedaten auf Ausgabedaten ändern kann und nicht fixiert ist. Wir haben es hierbei mit einer weiteren Stufe der Flexibilisierung eines Computersystems zu tun. Software ist die Flexibilisierung der Hardware, Lernen ist Flexibilisierung der Software.

Wir betrachten hier ein sehr einfaches Lern-Setting und fokussieren uns auf den Aspekt der Antizipation, wie es in Abb. 8.12 skizziert ist. Der Computer bekommt in einem ersten Schritt Beispiele von Eingabe-Ausgabe-Zuordnungen präsentiert. Die Aufgabe ist nun, auf dieser Basis für die Eingabedaten, für die keine Zuordnung angegeben ist, die Ausgabedaten zu antizipieren. Es ist klar, dass es sich dabei nur um eine Vermutung handeln kann, schließlich ist ja nur ein Ausschnitt der zu berechnenden Funktion bekannt.

Maschinelles Lernen, wie wir es hier skizziert haben, umfasst drei Aspekte. Erstens braucht es ein Wissensmodell, das das gelernte Wissen über die zu berechnende Funktion bzw. die gesuchte Eingabe-Ausgabe-Zuordnung beschreibt; das klingt etwas hochgestochen, doch es kann sich dabei auch einfach nur um eine Tabelle handeln. Zweitens braucht es ein Lernverfahren, das aus einer Menge an Beispielen von Eingabe-Ausgabe-Zuordnungen das Modell entwickelt; dieser Algorithmus definiert den Lernprozess.

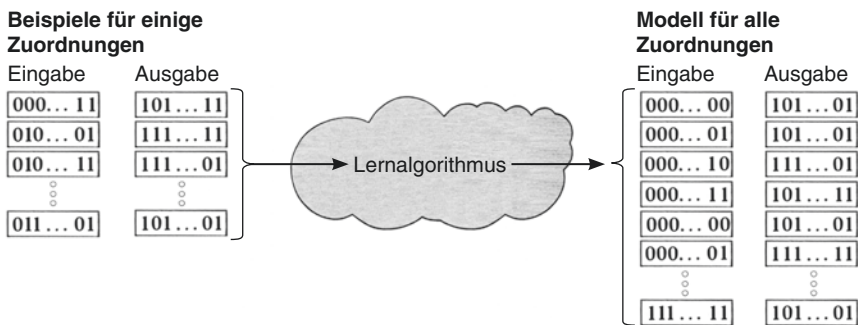


Abb. 8.12 Die Idee des maschinellen Lernens. (© Susanne Staubli, Eckart Zitzler)

Drittens braucht es ein Anwendungsverfahren, das aufgrund des Modells zu beliebigen Eingaben die korrespondierenden Ausgaben berechnet; dieser Algorithmus nutzt das Modell und wendet das Gelernte an.

Natürlich gibt es ganz verschiedene Ansätze, um so ein maschinelles Lernen zu ermöglichen. Eine einfache und durchaus effektive Methode heißt *nearest neighbor*. Dabei handelt es sich um ein Anwendungsverfahren, das in der Beispielmenge nach den ähnlichsten Eingabedaten sucht und die angegebenen Ausgabedaten als antizipierte Zuordnung für eine unbekannte Eingabe verwendet. Bei Bayesschen Netzen – sie stehen für eine Klasse von Wissensmodellen – werden Abhängigkeiten zwischen Eingabe- und Ausgabekomponenten als bedingte Wahrscheinlichkeitsverteilungen in Graphen abgebildet. Wir werden uns im Folgenden die künstlichen neuronalen Netze genauer anschauen, die – aus biologischer Sicht – äußerst primitive Nachbildungen natürlicher Nervensysteme und – aus informatischer Sicht – einen bewährten und vielfach genutzten Typ von Wissensmodellen darstellen. Bsp. 8.9 erklärt Ihnen, was genau ein künstliches neuronales Netz ist.

Beispiel 8.9: Ein Mini-Gehirn im Computer

Ein künstliches neuronales Netz ist eine Verknüpfung von künstlichen Neuronen. Sie können es sich vorstellen als eine digitale Schaltung, in der nur eine Art von Gattern vorkommt. Abb. 8.13 zeigt Ihnen ein solches Gatter, ein künstliches Neuron mit drei Eingängen, die sich hier auf der linken Seite befinden, und einem Ausgang, hier auf der rechten Seite. Jedes Schaltungselement ahmt schematisch eine Nervenzelle nach, wobei das konkrete Verhalten unterschiedlich definiert werden kann. In dem skizzierten Beispiel ist es so, dass die Eingabewerte ebenso wie der Ausgabewerte entweder 0 oder 1 sein können. In der einfachsten Variante ist der Ausgang auf 1, wenn der Mittelwert der Eingabewerte größer als 0,5 ist, ansonsten auf 0. Die Funktionalität entspricht einem Gatter, das genau dann 1 liefert, wenn die Mehrheit der Eingänge auf 1 gesetzt ist. Jetzt kommt noch hinzu – und dies ist das Besondere gegenüber einem herkömmlichen Gatter –, dass die Eingänge verschieden gewichtet werden können, genauso wie die Synapsen verschiedene Stärken aufweisen können. Wir summieren also alle Eingangswerte auf, jeweils multipliziert mit dem Gewichtungsfaktor, der in Abb. 8.13 an den Kanten angeschrieben ist. Bei diesen konkreten Gewichten feuert das künstliche Neuron nur, wenn der untere Eingang und ein weiterer Eingang auf 1 gesetzt sind.

Ein künstliches neuronales Netz ist zunächst nichts anderes als eine Beschreibung einer Verschaltung. Es kann als Wissensmodell eingesetzt werden, wenn einige ausgewählte Leitungen als Eingänge und Ausgänge der Schaltung definiert werden, über die Daten in das Netz eingespeist bzw. von ihm abgegriffen werden können. Das so festgehaltene Wissen kann schließlich

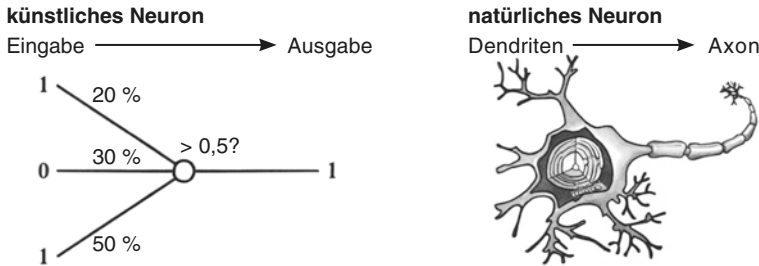


Abb. 8.13 Ein künstliches Neuron mit drei Eingängen und einem Ausgang (links) und sein natürliches Vorbild (rechts). (© Susanne Staubli, Eckart Zitzler)

abgerufen werden, indem ein Programm – das Anwendungsverfahren – das Verhalten dieses künstlichen Neuronenverbunds simuliert: Wird eine Eingabe angelegt, reagieren die so gereizten Nervenzellen, ihre Ausgaben werden im Netz weiterverarbeitet und führen dann zu einer bestimmten Belegung der Ausgangsleitungen. Die Frage ist natürlich, wie ein künstliches neuronales Netz erlernt werden kann und wie ein entsprechendes Lernverfahren aussieht. Dies möchte ich Ihnen anhand einer exemplarischen Anwendung aufzeigen, für die diese Art des maschinellen Lernens schon früh erfolgreich eingesetzt wurde: die Handschrifterkennung. Es gilt zu erkennen, ob auf einem Pixelbild die Ziffer 2 abgebildet ist oder nicht. Die Eingabe besteht aus einem einfachen Schwarz-Weiß-Bild mit 7×5 Pixeln bzw. einer Folge von 35 Bits, und die Ausgabe ist dann 1 oder 0, je nachdem, ob eine 2 erkannt wurde oder nicht. Diese Situation ist in Abb. 8.14 unten dargestellt: Sie sehen dort auch, wie ein mehrere Neuronen umfassendes künstliches Nervensystem aussehen kann.

Beim Lernen gilt es, ein künstliches neuronales Netz zu bestimmen, das – wenn das Anwendungsverfahren sein Verhalten simuliert – für alle oder möglichst viele der vorgegebenen Beispiele die korrekte Antwort liefert. Das heißt im Einzelnen: Die Struktur des Netzes, also die Anzahl der Neuronen und ihre Verbindungen, sowie die Gewichte an den Eingängen der Neuronen müssen fixiert werden. Im einfachsten Fall gibt man die Struktur vor und lässt über ein Lernverfahren nur noch die Gewichte in Abhängigkeit von den Beispielen anpassen. Eine häufig gewählte Struktur besteht aus mehreren Schichten von künstlichen Neuronen, jedes Neuron bekommt als Input alle Ausgänge der Neuronen der vorgelagerten Schicht. Bsp. 8.10 zeigt auf, wie das im Einzelnen funktioniert. Dabei gehen wir davon aus, dass unser Netz aus vier Schichten besteht; die am weitesten links stehende Schicht hat 35 Neuronen, die die Werte der Bildpixel aufnehmen, und die am weitesten rechts stehende Schicht umfasst ein Neuron, das den Ausgangswert liefert.

Meta-Meta-Strategie**Meta-Strategie**

2 ja
 2 ja
 1 nein
 5 nein

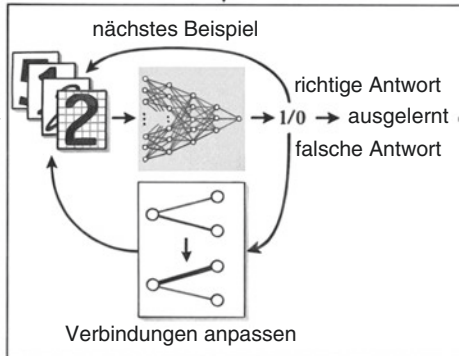
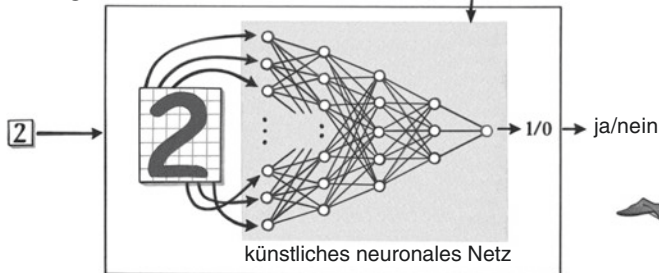
**Strategie**

Abb. 8.14 Wie ein künstliches neuronales Netz eingesetzt (unten) und gelernt (Mitte) wird. (© Susanne Staubli, Eckart Zitzler)

Beispiel 8.10: Wie ein künstliches neuronales Netz gelernt wird

Der Vorgang, wie die Gewichte eines vorgegebenen künstlichen neuronalen Netzes anhand von Beispielen eingestellt werden können, ist in Abb. 8.14 in der Mitte visualisiert. Am Anfang sind die Gewichte gleich verteilt, d. h., alle Eingänge besitzen den gleichen Einfluss; hat ein Neuron z. B. vier Eingänge, besitzen alle Eingänge ein Gewicht von 25 %. Dann werden die Beispiele einzeln durchprobiert, dabei handelt es sich um verschiedene Eingaben zusammen mit den korrekten Ausgaben. Das Netz wird mit der konkreten Eingabe simuliert und die Ausgabe berechnet. Jetzt kommt es darauf an, ob die berechnete

Ausgabe der gewünschten Ausgabe entspricht. Wenn ja, hat das Netz schon die korrekte Antwort geliefert. Wenn nicht, wird das künstliche neuronale Netz angepasst, und zwar schrittweise vom Ausgang aus. Im Englischen nennt man das *back propagation*, es handelt sich um eine gierige Strategie. Dabei werden die Gewichte der Eingänge jedes Neurons so angepasst, dass der Ausgangswert näher beim gewünschten Wert liegt. Die vorgegebenen Beispiele werden hintereinander, ggf. auch mehrfach durchexerziert und die Gewichte jeweils angepasst. Ist das Netz nun anhand verschiedener Beispiele trainiert, d. h. angepasst worden, wird es effektiv genutzt: Pixelbilder von handschriftlichen Zeichen werden eingegeben und die Antwort des Netzes ist die Antwort des Mustererkennungsprogramms.

Das Ziel beim maschinellen Lernen ist also, eine konkrete Lösungsstrategie automatisiert zu berechnen. Und für diese Berechnung brauchen wir wiederum eine Strategie, nämlich ein Lernverfahren; wir könnten hier von einer Meta-Strategie sprechen. Prinzipiell ist es sogar denkbar, dass wir auch die Lernstrategie wiederum lernen lassen; da würden dann ggf. andere Algorithmen als *back propagation* herauskommen. Wir hätten es mit einer Meta-Meta-Strategie zu tun: Wie lernt man am besten? Bei den Orcas könnte eine zusätzliche Stufe ebenso eine Veränderung der Lernstrategie sein: dass nicht nur von der Mutter gelernt, sondern Neues ausprobiert und so eine neue Jagdstrategie entwickelt wird. Damit kommen wir zum nächsten Thema: Spielen.

Spiele und Suche

Spiele sind Paradebeispiele für Situationen, in denen wir unsere kognitiven Fähigkeiten wie Denken, Planen, Bewerten und Entscheiden unter Beweis stellen können. Deswegen war man auch von Anfang an darauf aus, Computern das Spielen beizubringen – wobei es weniger um spielerisches Ausprobieren, sondern vor allem um Brett- und Kartenspiele geht. Ich möchte Ihnen am Beispiel von Schach, das zur Königsklasse der Strategiespiele zählt, demonstrieren, wie ein Computerprogramm Züge berechnen und somit den Part eines Spielers übernehmen kann. Die Situation ist bekannt: Zwei Spieler treten gegeneinander an, sie ziehen abwechselnd und versuchen sich jeweils in eine Spielsituation zu manövrieren, in der sie gewinnen. Aufgrund der aktuellen Spielsituation müssen sie entscheiden, was ihr nächster Zug ist – und dieser Zug sollte so gewählt sein, dass er die Wahrscheinlichkeit, am Ende als Sieger dazustehen, maximiert. Der Reiz beim Schach besteht in der enormen Komplexität,

die Anzahl möglicher Spielverläufe übersteigt die Anzahl der Atome im Universum bei Weitem.

Wenn wir Menschen Schach spielen, so denken wir Züge voraus, überlegen uns, was daraufhin der Gegner machen würde usw. Natürlich sind uns beim Vorausschauen Grenzen gesetzt, irgendwann müssen wir aufhören und die jeweils antizipierte Situation bewerten. Das geschieht anhand verschiedenster Erinnerungen an bislang gespielte Partien, anhand der Einschätzung des Gegners und anhand von Faktenwissen, was gute Züge sind, was gute Eröffnungen sind usw. Vieles entscheiden wir auch aus dem Gefühl heraus, insbesondere wenn wir das Verhalten des anderen errahnen wollen. Dieses Vorgehen ist in Abb. 8.15 links dargestellt. Aus der Perspektive der Informatik geht es darum, einen riesigen Raum an Möglichkeiten zu durchsuchen. Eine Lösung ist eine Folge von Spielzügen, die für uns, unabhängig davon, was der Spielgegner macht, zum Sieg führt; der erste Schritt in einer solchen Lösung verkörpert dann den nächsten Zug. Das riecht nach einem Optimierungsproblem und erinnert ein bisschen an das Hamilton-Pfad-Problem, nur dass ein Pfad hier eine Spielsequenz ist und der Zielpunkt

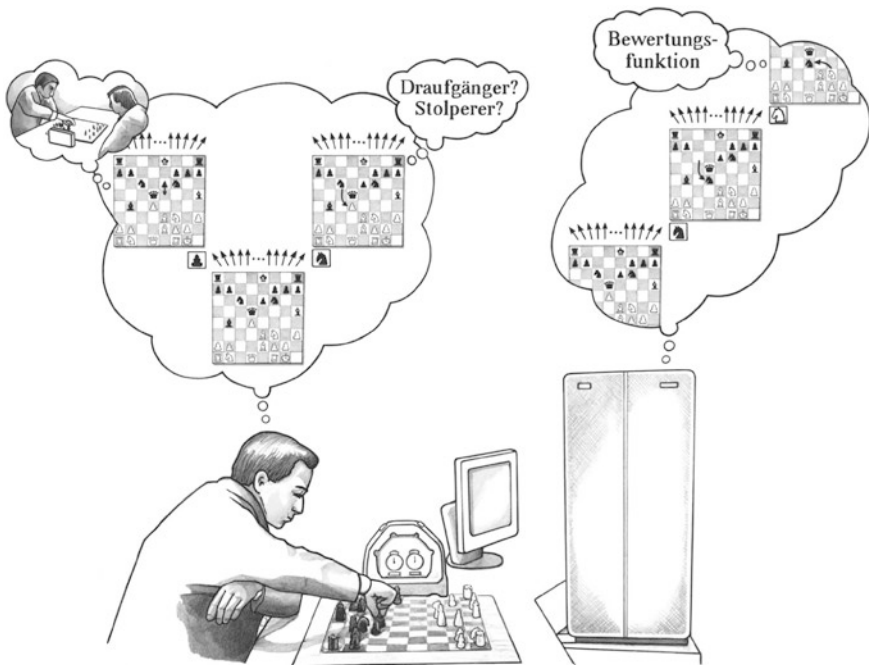


Abb. 8.15 Wie ein Mensch und ein Computer Schach spielen. (© Susanne Staubli, Eckart Zitzler)

einem Schachmatt oder zumindest einem Remis entspricht. Allerdings kann selbst der leistungsfähigste Computer nicht alle Spielsituationen und Zugfolgen durchspielen, bereits nach zwei Zügen gibt es mehrere Zehntausend Möglichkeiten. Stattdessen schaut er nur ein paar Züge voraus und schätzt dann über eine Bewertungsfunktion Spielsituationen daraufhin ein, wie vielversprechend sie sind (siehe Abb. 8.15 rechts). Diese Methode führte tatsächlich zum Erfolg, wie Sie in Anm. 8.2 nachlesen können.

Anmerkung 8.2: Als Deep Blue den Schachweltmeister schlug

Lange Zeit galt Schach als Domäne des Menschen, für viele schien es unvorstellbar, dass ein Computer einen menschlichen Schachweltmeister schlagen könne. Doch bereits 1996 war es so weit: Der auf Schach spezialisierte Computer Deep Blue schlug den amtierenden Schachweltmeister Garri Kasparow in einer regulären Partie; ein Jahr später gewann Deep Blue sogar einen ganzen Wettkampf aus sechs Partien unter Turnierbedingungen. Deep Blue entstand bei IBM und war im Wesentlichen ein Parallelcomputer mit 30 Prozessoren und weiteren spezialisierten Chips. Er rangierte unter den 300 schnellsten Computer weltweit und war in der Lage, 200 Mio. Positionen pro Sekunde zu berechnen. Die Bewertungsfunktion wurde von Hand programmiert, besaß allerdings flexible Anteile, die per maschinellern Lernen eingestellt wurden, indem Tausende von Schachpartien von Schachmeistern automatisiert analysiert wurden. Darüber hinaus hatte Deep Blue eine Datenbank mit über 4000 Eröffnungsvarianten und 700.000 Großmeisterpartien. Womit Deep Blue also gewonnen hat, ist schiere Rechenleistung, gepaart mit geschickten Bewertungsfunktionen, die der Intelligenz der Programmierer entspringen. Deep Blue hat also nicht Schachspielen erlernt, sondern ist spezifisch für dieses Spiel konstruiert worden.

Das skizzierte Vorgehen repräsentiert ein allgemeines algorithmisches Verfahren zur Berechnung von Spielzügen bei Zwei-Parteien-Spielen. Das Prinzip ist, möglichst viele Züge vorzuschauen und diese zu bewerten, um so den aussichtsreichsten nächsten Zug auszuwählen. Der Computer simuliert einen Spielzug für sich selbst, dann einen für den Gegner, immer im Wechsel. Der Suchverlauf entspricht einem riesigen Baum – Spielbaum genannt –, bei dem die aktuelle Spielsituation die Wurzel darstellt, und die Knoten die fiktiven, ausprobierten Spielsituationen repräsentieren. Die Frage ist nur, wie ein konkreter Spielzug ausgewählt wird. Schließlich weiß der Computer ja nicht, wie der Gegner reagieren wird. Nun, die Annahme ist schlicht, dass jeder bestmöglich zieht. Die Spielsituationen werden nun von den Blättern des Baumes ausgehend von unten nach oben bis zur Wurzel bewertet, der vielversprechendste Nachfolger der Wurzel verkörpert den Zug, den der Computer als Nächstes macht. Das Ganze nennt sich

Minmax-Strategie. Bsp. 8.11 erläutert diesen Ansatz anhand des einfachen Spiels Tic-Tac-Toe, bei dem zwei Spieler auf einem 3×3 -Spielfeld abwechselnd Steine legen und das Ziel haben, drei gleiche Steine in einer Spalte, Zeile oder Diagonale zu platzieren.

Beispiel 8.11: Mal optimistisch, mal pessimistisch

In Abb. 8.16 sehen Sie einen Spielbaum für Tic-Tac-Toe, ausgehend von einer Spielsituation, bei der zwei Drittel der Spielfelder bereits belegt sind und „Haken“ den nächsten Zug ausführt. Der Spielbaum umfasst alle möglichen Zugfolgen bis zum Spielende. Schauen wir uns nun an, wie die einzelnen Spielsituationen bewertet werden.

Wir starten bei den Blättern, den Endsituationen, und bewerten sie mit -10 , 0 oder 10 , wobei diese Zahlen für verloren, unentschieden und gewonnen stehen. Das lässt sich ja einfach feststellen. Anschließend bewerten wir die Spielsituationen unmittelbar vor den Blättern. Wenn dort der Gegner am Zug ist, nehmen wir den minimalen Wert der möglichen Nachfolgesituationen, und wenn wir selbst am Zug sind, wählen wir entsprechend den maximalen Wert – deswegen auch der Name Minmax-Strategie. Auf diese Weise arbeiten wir uns schrittweise von den Blättern hoch, bis schließlich alle Knoten bewertet sind. Am Ende wählen wir denjenigen Nachfolgerknoten der Wurzel als Zug aus, der die höchste Bewertung hat.

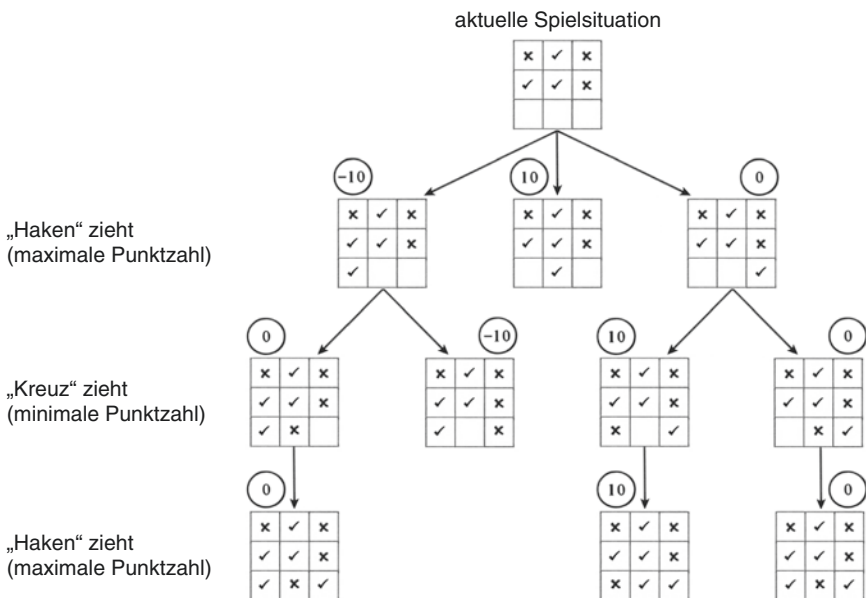


Abb. 8.16 Ein exemplarischer Spielbaum für Tic-Tac-Toe. (© Susanne Staubli, Eckart Zitzler)

Bei den meisten Strategiespielen ist es nicht möglich, den gesamten Spielbaum zu betrachten. Man versucht möglichst viele Züge vorauszuschauen. Des Weiteren lässt sich das Verfahren noch nach dem Prinzip von „Verzweige und begrenze“ beschleunigen. Indem wir uns merken, was der jeweils minimale Wert für beide Spielgegner ist, können wir bestimmte Teilbäume komplett bei der Bewertung ignorieren. Das nennt sich *alpha-beta pruning*. Wenn man die Sache so ansieht, ist es nicht verwunderlich, dass Computer Menschen bei solchen Spielen schlagen können. 20 Jahre nach dem Erfolg von Deep Blue wurden auch Profispieler im Brettspiel Go von Computern besiegt; dabei kamen ebenfalls künstliche neuronale Netze zum Einsatz. Wir könnten gespannt sein, was die Zukunft hier noch bringen wird – auch in Bezug auf das nächste Thema: die Computerlinguistik.

Sprache und Kommunikation

Die Sprache ist ein weites Feld. Ihre Verarbeitung mittels Computern ebenso, das reicht von der Umwandlung gesprochener in geschriebene Sprache über die Entschlüsselung und Interpretation von Texten bis zur visuellen und akustischen Sprachausgabe. Nicht ohne Grund hat sich mit der Computerlinguistik ein eigenes Feld etabliert, das die Sprachwissenschaften mit der Informatik verbindet. Und eigentlich hat das Thema der Sprache seit jeher einen festen Platz in der Informatik. Einerseits ist es der stetige Wunsch, die Bedienung des Computers für den Anwender und die Programmiererin zu vereinfachen – wir verständigen uns nun mal nicht über 0-1-Muster, sondern über Laut- und Schriftzeichenfolgen. Andererseits sind Sprachen aus theoretischer Warte interessant, weil über sie verschiedene Computermodelle hinsichtlich ihrer Mächtigkeit klassifiziert werden können. Aufgrund der Materialfülle greife ich hier nur einen Aspekt heraus: die Spracherkennung. Wir schauen uns eine Herangehensweise an, wie Computer Texte verstehen können, und konzentrieren uns dabei auf künstliche, stark formalisierte Sprachen, die z. B. bei der Programmierung zum Einsatz kommen. So eingeschränkt der Einblick auch sein mag, er vermittelt Ihnen eine Idee davon, was bei der Analyse natürlichsprachlicher Informationen abläuft.

Gut, worum geht es bei der Spracherkennung? Eine Textnachricht zu entschlüsseln, heißt eine Zeichenfolge in zusammengehörige Einheiten zu zerlegen, deren Bedeutungen zu erfassen und aus der Art und Weise, wie die Einheiten zueinander angeordnet sind, einen übergeordneten Sinnzusammenhang zu erschließen. In unserer Schriftsprache sind die Wörter die elementaren Bestandteile, die wir leicht abwandeln und dann

zu Sätzen zusammenfassen. Bei konstruierten Sprachen ist das ganz ähnlich, nur versucht man hier die Strukturen möglichst einfach zu halten und die erfrischenden Eigenheiten natürlicher Sprachen – Doppeldeutigkeiten, Untertöne, Wortabwandlungen etc. – zu umgehen. Programmiersprachen beispielsweise sind darauf ausgerichtet, Abläufe präzise zu beschreiben; sie sollen für Menschen leicht verständlich und gleichzeitig unmissverständlich sein. Daher ist der Sprachbegriff in der Informatik auch stark abstrahiert: Unter einer (formalen) Sprache wird schlicht eine Menge von Zeichenketten verstanden, die dieser Sprache angehören; die Zeichen entstammen einem vorgegebenen Alphabet. Bsp. 8.12 zeigt Ihnen, wie so eine formale Sprache konkret aussehen kann.

Beispiel 8.12: Die Zahlensprache

Die in Abb. 8.17 dargestellte formale Sprache dient der Darstellung von Zahlen und kann als Fragment einer Programmiersprache betrachtet werden. Ein in dieser Sprache verfasster Text besteht einzig aus einer positiven Dezimalzahl, die möglicherweise noch Nachkommastellen aufweist. Das Alphabet besteht entsprechend aus den Ziffern von 0 bis 9 und dem Komma, woraus sich natürlich auch Zeichenketten bilden lassen, die nicht Teil der Sprache sind. Übrigens wird eine Zeichenkette auch als Wort bezeichnet – das unterscheidet sich vom alltäglichen Gebrauch – und die Gesamtheit aller Zeichenketten, die sich aus einem Alphabet bilden lassen, als Kleenesche Hülle.

Beispielsweise würde die deutsche Sprache aus allen syntaktisch korrekten Satzfolgen bestehen, legten wir den Begriff der formalen Sprachen zugrunde. Und wie natürliche Sprachen sind formale Sprachen über einen Satz an Regeln definiert, ihre Grammatik. Eine Grammatik legt die Baupläne fest, nach denen Sprachkonstrukte wie Sätze aus anderen Sprachkonstrukten

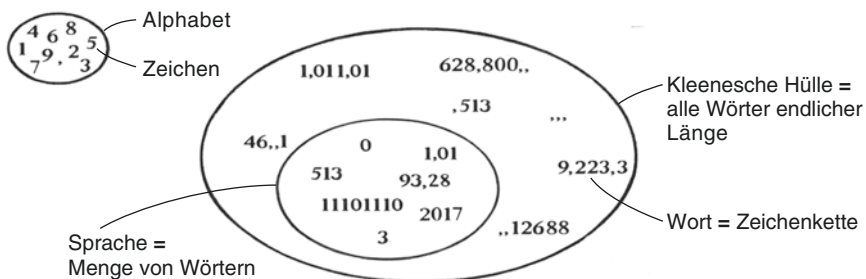


Abb. 8.17 Die Sprache der Zahlen mit Komma. (© Susanne Staubli, Eckart Zitzler)

wie Nebensätzen und Hauptsätzen gebildet werden können. Eine echte Sprache hat unglaublich viele Regeln, deswegen sind die Sprachen, die Computer erkennen, viel einfacher gestrickt. Bsp. 8.13 zeigt auf, wie eine Grammatik für die Zahlensprache aussehen kann, die die Konstruktion von Zeichenketten definiert. Entscheidend dabei: Die Grammatik besteht aus endlich vielen Zeichen und Regeln, aber die Sprache selbst kann unendlich viele Zeichenketten enthalten. Die Grammatik macht aus abstrakten Sprachkonstrukten konkrete Zeichenketten.

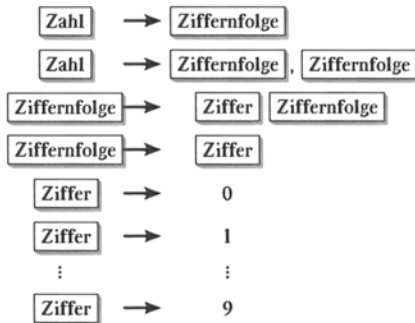
Beispiel 8.13: Wie eine Grammatik eingesetzt wird

Die Grammatik einer formalen Sprache besteht aus Regeln, die Folgen von Sprachkonstrukten und Zeichen in andere Folgen von Sprachkonstrukten und Zeichen überführen. Die in Abb. 8.18 abgebildeten Regeln sind beispielsweise folgendermaßen zu lesen: Links stehen jeweils Platzhalter, sogenannte Nichtterminalsymbole, die für einen Sprachbestandteil oder ein Sprachkonstrukt stehen, und rechts stehen die Elemente, aus denen dieser Bestandteil selbst besteht. Diese Elemente können Zeichen – im vorliegenden Kontext Terminalsymbole genannt – oder selbst wieder Platzhalter sein.

Die Regeln lassen sich nun einsetzen, um daraus korrekte Zeichenkette der Sprache zu generieren, das ist auf der rechten Seite von Abb. 8.18 visualisiert. Wir beginnen mit dem Platzhalter „Zahl“, er steht für die oberste Ebene und ist das Startsymbol, mit dem jede Wortableitung beginnt. Jetzt wenden wir schrittweise die Regeln auf die Sprachbestandteile an. Auf diese Weise können wir beliebige Dezimalzahlen mit oder ohne Nachkommateil erzeugen. Die Zahl 1,01 erzeugen wir z. B., indem wir erst die zweite Regel anwenden, anschließend die dritte, dann die sechste, dann die vierte, dann die fünfte, die dritte und zum Schluss wieder die sechste. Die Zerlegung vom Startsymbol zum konkreten Text lässt sich grafisch als Baum darstellen, der auch als Syntaxbaum bezeichnet wird. Ein Syntaxbaum für einen natürlichsprachlichen Text ist uns übrigens schon in Abb. 5.2 begegnet.

Die Grammatik beschreibt die Syntax, aber noch nicht die Bedeutung, die mit ihr erzeugte Zeichenfolgen besitzen. Um eine vorgegebene Zeichenkette zu interpretieren, müssen wir zunächst einmal den zugrunde liegenden Syntaxbaum rekonstruieren. Bei einer natürlichen Sprache könnte dies z. B. heißen, Subjekt und Prädikat zu identifizieren, auch wenn die Zerlegung eines Textes in seine Bestandteile bei uns unbewusst erfolgt. Die entsprechenden Satzbestandteile haben eine Bedeutung, die wir aus Erfahrungen kennen, und aus der Information, welche Wörter welche Rolle im Satz spielen, können wir auf den Sinnzusammenhang schließen, also wer was gemacht hat. Das ist bei der Zahlensprache ähnlich: Jede Ziffer hat eine Wertigkeit, die sich aus ihrer Position in der Zeichenkette ergibt – all diese Informationen liefert der Syntaxbaum. Auf dieser Basis kann ein Computer

Grammatik



Syntaxbaum

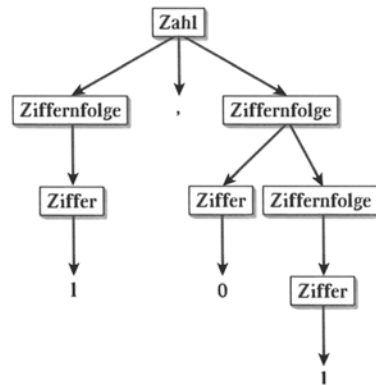


Abb. 8.18 Eine Grammatik zur Erzeugung der Sprache der Zahlen mit Komma.
(© Susanne Staubli, Eckart Zitzler)

z. B. die Zeichenfolge „16“ interpretieren und in das interne binäre Muster 1000 umwandeln. Spracherkennung heißt also, den Syntaxbaum für eine Zeichenkette aufzubauen und dann die strukturellen Informationen auszuwerten. Wir müssen quasi die Regeln in der Grammatik identifizieren und wissen, in welcher Reihenfolge welche Regeln angewendet wurde. Das kann unterschiedlich schwierig sein, es hängt davon ab, wie die Grammatik aufgebaut ist (siehe Anm. 8.3).

Anmerkung 8.3: Die Chomsky-Hierarchie

Die Regeln einer Grammatik können einem Format folgen oder beliebig sein. Man unterscheidet verschiedene Arten von Grammatiken, die sich als Hierarchie präsentieren, weil die Regeln immer weiter eingeschränkt werden. Die vier Grammatiktypen sind in Abb. 8.19 dargestellt, wobei die Kästchen für Platzhalter stehen und die Ziffern stellvertretend für konkrete Zeichen aus dem Alphabet.

Eine Grammatik vom Typ 0 hat keine Einschränkungen bzgl. der Regeln. Bei einer Grammatik vom Typ 1, auch kontextsensitiv genannt, ist garantiert, dass auf der linken Seite einer Regel immer mehr Platzhalter stehen als rechts. Eine kontextfreie Grammatik (Typ 2) besteht nur aus Regeln, auf deren linker Seite genau ein Platzhalter steht. Und bei einer regulären Grammatik (Typ 3) darf auf der linken Seite einer Regel immer nur ein Platzhalter stehen und auf der rechten Seite entweder eine leere Zeichenkette oder ein Terminalsymbol, gefolgt von einem Nichtterminalsymbol. Die erzeugten Sprachen werden ebenso wie die Grammatiken bezeichnet, d. h., eine reguläre Sprache wird durch eine reguläre Grammatik erzeugt. Die vier Grammatiktypen werden gesamthaft auch als Chomsky-Hierarchie bezeichnet, denn diese Unterteilung formaler Grammatiken geht auf den Linguisten Noam Chomsky (Abb. 8.20) zurück.

Typ 0 unbeschränkte Grammatik	0	☐	0	☐	1	→	0	☐	☐	1
Typ 1 kontextsensitive Grammatik	0	☐	1	→	0	☐	☐	1		
Typ 2 kontextfreie Grammatik			☐	→	0	☐	1	☐	1	
Typ 3 reguläre Grammatik			☐	→	☐	0				

☐ Nichtterminalsymbole = Platzhalter
 0 1 Terminalsymbole = Zeichen aus Alphabet

Abb. 8.19 Arten von formalen Grammatiken. (© Susanne Staubli, Eckart Zitzler)



Abb. 8.20 Noam Chomsky, der formale Grammatiken gemäß ihrer Ausdruckskraft klassifiziert hat. (© Camille Büsser)

Die Zahlensprache gehört zum einfachsten Sprachtyp, den regulären Sprachen, auch wenn die Grammatik in Abb. 8.18 eher nach Typ 2 aussieht. Die Grammatik lässt sich jedoch so umformulieren, dass sie tatsächlich regulär wird. Eine Möglichkeit ist in Abb. 8.21 auf der linken Seite dargestellt, wobei diese Grammatik natürlich wesentlich unübersichtlicher und umständlicher wirkt als das Original; der Platzhalter „T1“ ist das Startsymbol und entspricht dem Sprachkonstrukt „Zahl“ in der vorherigen Grammatik. Aus so einer regulären Grammatik lässt sich leicht ein Ablauf definieren, der Zeichenketten von links nach rechts abarbeitet und dabei erstens entscheidet, ob die Zeichenkette ein gültiges Wort der Sprache darstellt, und zweitens den entsprechenden Syntaxbaum erzeugen kann. Dieser Ablauf ist in Abb. 8.21 auf der rechten Seite visualisiert und erinnert nicht von ungefähr an die Abläufe aus Kap. 3, als wir die Steuerung

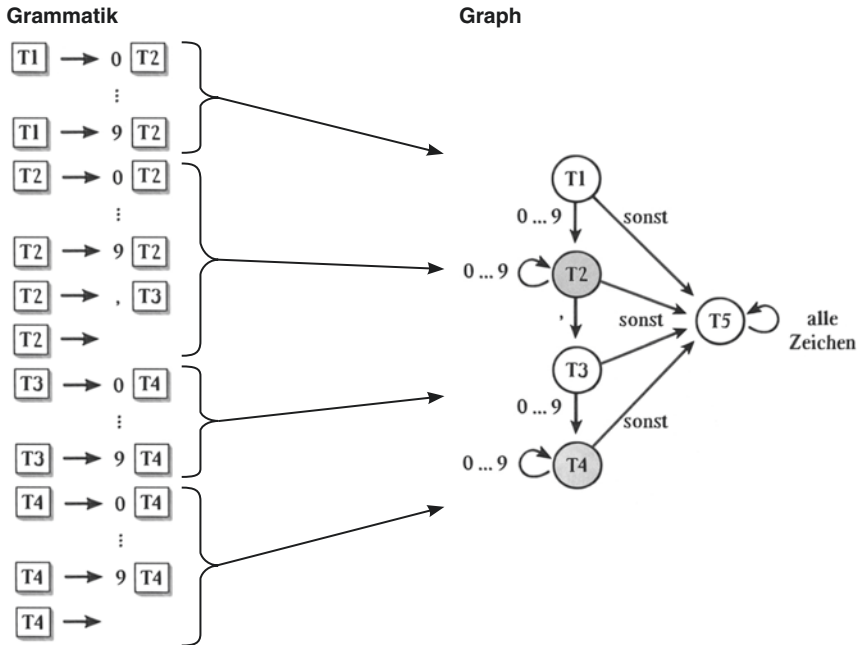


Abb. 8.21 Eine reguläre Grammatik für die Zahlensprache (links) und ein entsprechender, als Graph dargestellter Ablauf, um einen Syntaxbaum zu rekonstruieren (rechts). (© Susanne Staubli, Eckart Zitzler)

für den Prozessor definiert haben – es liegt das gleiche Prinzip zugrunde. An den Kanten ist jeweils angeschrieben, wohin verzweigt werden soll, wenn das nächste Zeichen gelesen wird. Bsp. 8.14 erläutert das Ablaufschema im Einzelnen.

Beispiel 8.14: Wie man die Zahlensprache erkennt

Um aus einer regulären Grammatik einen Ablauf zur Rekonstruktion eines Syntaxbaums zu erstellen, geht man wie folgt vor: Für jeden Platzhalter in der Grammatik definieren wir einen Schritt bzw. Zustand in unserem Ablauf, dargestellt mit einem Kreis. Zudem kreieren wir einen zusätzlichen Schritt, der den Zustand kennzeichnet, wenn eine Zeichenkette nicht ein Wort der entsprechenden Sprache ist (in Abb. 8.21 ist das der Zustand „T5“). Dann verbinden wir die Schritte mit Pfeilen, und an den Pfeilen steht jeweils, welche Zeichen zum Zustandsübergang von dem einen zum anderen Schritt führen. Das funktioniert, weil man eine reguläre Grammatik immer so ausdrücken kann, dass jeweils klar ist, welche Regel als Nächstes angewendet werden muss; in unserem Beispiel in Abb. 8.21 ist das so. Zum Schluss markieren wir die Zustände, im Beispiel „T2“ und „T4“, bei denen das Ende der Zeichenkette erreicht werden darf und die Zeichenkette als korrekt erkannt ist.

Wenn wir den Ablauf anwenden wollen, starten wir bei „T1“, nehmen das erste Zeichen der Zeichenfolge und folgen dem entsprechenden Pfeil. Anschließend nehmen wir das zweite Zeichen, folgen wieder dem Pfeil vom aktuellen Zustand, bis wir am Ende der Zeichenkette angelangt sind. Befinden wir uns zum Schluss in einem der markierten Zustände, handelt es sich um eine syntaktisch korrekte Zeichenkette, ansonsten nicht. Der Syntaxbaum ergibt sich übrigens einfach aus der Abfolge der Zustände, die wir besuchen. Bei der Zeichenfolge „1,01“ z. B. wäre die Abfolge der Schritte „T1“, „T2“, „T3“, „T4“, „T4“.

Ein so konstruierter Ablauf lässt sich nun einfach in ein Programm umsetzen. Es gibt sogar ein Maschinenmodell dafür, eine abgespeckte Turing-Maschine, die solche Abläufe ausführen bzw. reguläre Sprachen erkennen kann. Diese nennt sich endlicher Automat (Anm. 8.4). Ein endlicher Automat kann jedoch keine Sprachen erkennen, die mittels Grammatiken der anderen Typen erzeugt werden. Dafür gibt es wiederum andere Maschinenmodelle, z. B. Kellerautomaten für die Erkennung kontextfreier Sprachen, zu denen Programmiersprachen in der Regel gehören. Für kontextsensitive Sprachen braucht es dann bereits die ganze Turing-Maschine ohne Einschränkungen. Es ergibt sich also eine Hierarchie von verschiedenen mächtigen Computermodellen, die mit der Chomsky-Hierarchie verknüpft sind, und das ist wiederum interessant für die Theoretische Informatik. Und man hat jeweils Standardverfahren entwickelt, wie solche Modelle automatisiert in Programme umgesetzt werden können, die die entsprechenden Sprachen erkennen.

Anmerkung 8.4: Der kleine Bruder der Turing-Maschine

Die eingeschränkte Turing-Maschine wird endlicher Automat genannt und unterscheidet sich von der Turing-Maschine in vier Punkten. Erstens wird das Band nach jedem Rechenschritt automatisch um eins nach links verschoben, es gibt keine individuellen Bandbewegungen pro Rechenschritt. Zweitens kann das Band nur ausgelesen und nicht beschrieben werden. Drittens ist das Band nur so lang wie die zu prüfende Zeichenkette; der endliche Automat stoppt, sobald alle Zeichen gelesen wurden. Und viertens ist zu jedem Rechenschritt anzugeben, ob es sich um einen akzeptierenden oder nichtakzeptierenden Zustand handelt; stoppt die Maschine in einem akzeptierenden Zustand, leuchtet die „OK“-Lampe und das Wort ist korrekt, ansonsten ist die Zeichenkette auf dem Band kein Wort der Sprache.

Das Programm kann ebenso über Tabellen beschrieben werden, wobei eben die Bandbewegung und das zu schreibende Zeichen wegfallen (siehe Abb. 8.22). Bei endlichen Automaten wird jedoch häufig eine grafische Darstellung gewählt, und zwar mittels eines Graphen; für unser Beispiel entspricht sie dem in Abb. 8.21 abgebildeten Graphen. Jeder Knoten repräsentiert

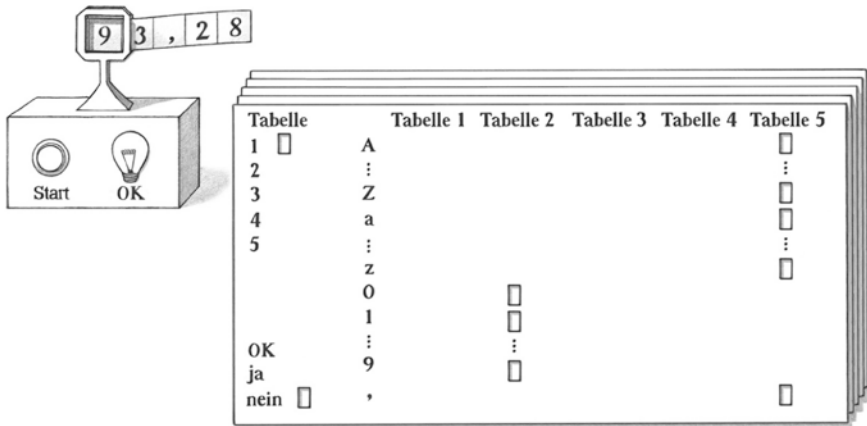


Abb. 8.22 Ein endlicher Automat. (© Susanne Staubli, Eckart Zitzler)

eine Tabelle, also einen Rechenschritt oder Zustand. Die Kanten repräsentieren die Zustandsübergänge: welche Tabelle auf welche Tabelle folgt in Abhängigkeit des eingelesenen Zeichens unter dem Lesekopf. Und schließlich gibt es noch die Knotenmarkierungen, die für akzeptierende Zustände stehen.

Damit wollen wir es hier belassen. Die Fähigkeit eines Computers, mit Sprache umzugehen, wurde stets als guter Gradmesser für den Stand der Künstlichen Intelligenz betrachtet. Das erklärt, warum Spracherkennung, aber auch Spracherzeugung von so großem Interesse sind, vor allem in Hinblick auf die Frage, ob Computer intelligent sein können.

Grenzen der Berechenbarkeit

Bis hierher stand im Vordergrund, Probleme zu lösen und entsprechende Algorithmen zu entwerfen – von einfachen klassischen Datenverarbeitungsaufgaben wie Sortieren von Namenslisten bis zu komplexen Fragestellungen, die mit Intelligenz in Verbindung gebracht werden. Und wie wir gesehen haben, können Computer immer mehr, weit mehr, als man sich jemals vorstellen konnte. Doch jetzt beschäftigen wir uns mit den Grenzen. Kann es Situationen geben, in denen wir keinen Algorithmus finden oder vielleicht nur einen, der praktisch nicht umsetzbar ist? Und selbst wenn wir ein geeignetes Lösungsverfahren im Kopf haben, woher wissen wir dann, dass ein anderes nicht besser ist? Solchen Themen geht die

Theoretische Informatik nach, sie hat ihre Wurzeln in der Mathematik. Und die Frage, ob und was sich eigentlich alles berechnen lässt, stand schon im Raum, bevor die ersten Computer das Licht der Welt erblickten. Ja, es war sogar diese Frage, die schließlich zur Konzeption der Turing-Maschine führte. Doch eins nach dem anderen.

Ich weiß, dass ich nichts weiß

Kommen wir kurz noch einmal auf das Verhalten zu sprechen. Je ausgeklügelter die Gehirne wurden, umso weniger starr waren ihre Besitzer an die angeborenen Verhaltensweisen gebunden und desto differenzierter konnten sie auf innere und äußere Reize reagieren. Der Mensch mit seiner Großhirnrinde kann enorme Anpassungsleistungen erbringen. Nicht nur, dass er sein Verhalten reflektieren und durch Überlegungen ändern kann: Situationen vorausdenken, sie durchspielen und im Geiste bewerten, auf dieser Basis Konsequenzen ziehen. Er ist mit seinem Denkorgan in der Lage, Erkenntnisse über die Welt zu gewinnen und zu Einsichten zu kommen, die zu äußerst komplexen Verhaltensweisen führen können, z. B. dem Bau von Computern. Mit diesen Werkzeugen hat er den Radius seines Denkens erweitert, und er kann nun Probleme automatisiert bearbeiten, wofür Menschengehirne nicht oder nur beschränkt geschaffen sind. Da kann man sich durchaus fragen, ob das so weitergeht. Natürlich, unserem Körper sind Grenzen gesetzt, aber auch dem logischen Denken?

Dass es mit der Logik so eine Sache ist, musste die Mathematik Anfang des 20. Jahrhunderts schmerzlich erfahren. Man arbeitete damals intensiv daran, das logische, mathematische Denken zu formalisieren – mittels sogenannter Kalküle (Anm. 8.5). Dafür schuf man künstliche Sprachen, mit denen man mathematische Aussagen präzise, prägnant und unmissverständlich ausdrücken konnte. Damit verbunden war auch eine exakte Definition des Schlussfolgerns, und zwar in Form von Regeln, die beschreiben, wie aus Aussagen weitere Aussagen abgeleitet werden können; die Basis bildet ein Satz an elementaren Aussagen, die als gegeben betrachtet werden.

Anmerkung 8.5: Kalküle, die Vorreiter der Programmiersprachen

Am Anfang war die Mathematik untrennbar mit der Wirklichkeit verbunden, mathematische Objekte wie Zahlen spiegelten reale Beobachtungen wie Anzahlen, Längen usw. wider, und das Schlussfolgern entsprach unserem kausalen Denken, bei dem wir Gegebenheiten in eine logische Reihenfolge bringen. Im 19. Jahrhundert verselbstständigte sich die Mathematik immer mehr,

d. h., sie löste sich von der Wirklichkeit und wurde abstrakter. Zahlen wurden auf einmal als eigenständige gedankliche Konstrukte aufgefasst, die nicht notwendigerweise eine Entsprechung in der Realität haben mussten. Aus den Grundannahmen, die wir aus unserer Erfahrungswelt als gegeben betrachteten, z. B. dass es zwischen zwei Punkten immer eine gerade Linie gibt, wurden abstrakte Axiome. Das logische Schlussfolgern schließlich wurde in formale Regeln übersetzt, wie sich mathematische Aussagen kombinieren lassen und zu neuen Aussagen führen. Und man entwickelte formale Sprachen, wie wir sie z. B. von der Aussagenlogik oder der Prädikatenlogik kennen, um mathematische Aussagen losgelöst von der natürlichen Sprache zu formulieren.

Ein Kalkül bringt all diese Aspekte unter den Hut eines formalen Systems und beschreibt, wie sich Aussagen aus einem Satz an Grundannahmen herleiten lassen. Es besteht aus einer formalen Sprache, Axiomen, Schlussfolgerungsregeln und einer Semantik, die dem Ganzen Bedeutungen zuweist. Ein für die Informatik wichtiges Kalkül ist das von Alonzo Church und Stephen Kleene entwickelte Lambda-Kalkül, mit dem sich die berechenbaren Funktionen beschreiben lassen; es fungierte später als Vorbild und Grundlage für diverse Programmiersprachen.

An und für sich ging es um die Mechanisierung des Beweisführens und somit um die Automatisierung des mathematischen Denkens. Das übergeordnete Ziel war, salopp gesagt, Ordnung in das logische Denken zu bringen und die Sprache der wahren mathematischen Aussagen zu definieren. Der berühmte Mathematiker David Hilbert war ein Verfechter der Theorie, dass eine Mechanisierung möglich ist. Aus diesem Grund rief er 1900 seine Kollegen dazu auf, ein formales Grundgerüst für die Mathematik zu schaffen, mit dem sich aus gewissen Axiomen und Regeln alle wahren Aussagen herleiten lassen (Vollständigkeit), mit dem sich keine sich widersprechenden Aussagen erzeugen lassen (Widerspruchsfreiheit) und mit dem für alle mathematisch formulierbaren Aussagen entscheidbar ist, ob sie sich aus den Axiomen über die Regeln ableiten lassen oder nicht (Entscheidbarkeit).

Bereits zwei Jahre später zeigte sich, vor welcher Herausforderung man stand. Als der deutsche Mathematiker Gottlob Frege ein solches Grundgerüst für ein Teilgebiet der Mathematik, die Mengenlehre, zu erschaffen versuchte, traten fundamentale Probleme zu Tage. Der damals verwendete Mengenbegriff führte zu Paradoxien, d. h. widersprüchlichen Aussagen. Bertrand Russell war derjenige, der dies mit der Russellschen Antinomie offenlegte: Die Menge, die alle Mengen beinhaltet, welche sich nicht selbst als Element beinhalten, stellt so ein Paradoxon dar (siehe Abb. 8.23). Denn enthält die Menge sich selbst, dann darf sie laut Definition nicht in sich enthalten sein; enthält sie sich nicht, dann muss sie sich laut Definition selbst enthalten.

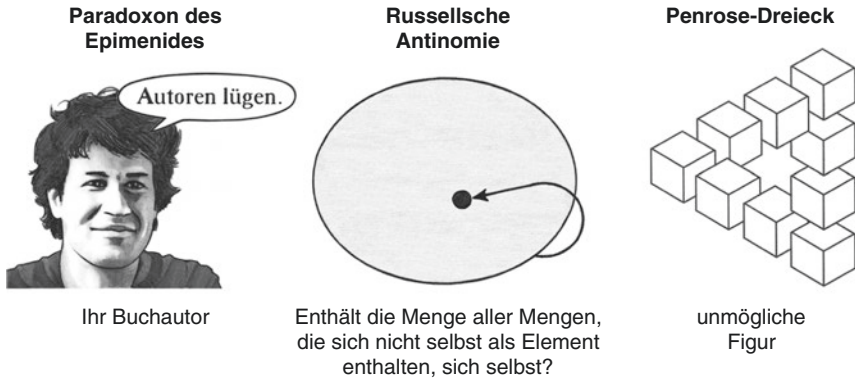


Abb. 8.23 Beispiele für Paradoxien. (© Susanne Staubli, Eckart Zitzler)

Ein anderer Mathematiker, der junge Kurt Gödel, nahm Hilbert schließlich seinen Traum. Gödel konnte zeigen, dass Vollständigkeit und Widerspruchsfreiheit gleichzeitig so nicht erfüllbar sind (Anm. 8.6). Aber es kam noch schlimmer. Alan Turing konnte anhand seiner maschinell denkenden Maschine, der Turing-Maschine, beweisen, dass auch der Aspekt der Entscheidbarkeit so nicht umsetzbar ist. Man kann also nicht für jede Aussage entscheiden, ob sie nun mit den Regeln aus den Axiomen ableitbar ist oder nicht, ganz abgesehen davon, ob diese Überprüfung ein Mensch oder eine Maschine vornimmt.

Anmerkung 8.6: Wenn eine Aussage auf sich selbst verweist

Kurt Gödels Beweis ist in der arithmetischen Mathematik angesiedelt. Ihm gelang es, arithmetische Formeln selbst als Zahlen darzustellen. Dazu erfand er einen geeigneten Code, die sogenannte Gödelnummerierung, die es erlaubt, Zahlen und Aussagen über Zahlen eindeutig aufeinander abzubilden. Jetzt war es auf einmal möglich, die Ebene der arithmetischen Formeln zu mischen mit der Ebene, auf der Aussagen über solche Formeln gemacht werden. Er konstruierte eine Formel, die sinngemäß die Aussage codierte: „Diese Formel ist nicht beweisbar.“ Jetzt ergibt sich folgende Situation: Entweder ist die Formel wahr, aber dann lässt sie sich im formalen System der Zahlentheorie nicht herleiten bzw. beweisen; in diesem Fall ist das Kalkül unvollständig. Oder die Formel ist aus den Axiomen herleitbar, dann führt das zu einem Widerspruch und das Kalkül ist nicht widerspruchsfrei.

Dass es keinen hinreichend komplexen Kalkül geben kann, der vollständig, widerspruchsfrei und entscheidbar ist, hat weitreichende Konsequenzen. Es handelt sich um fundamentale Grenzen nicht nur für die Mathematik,

sondern auch für die Informatik. Alan Turing hat nämlich nicht nur Hilberts Frage nach der Entscheidbarkeit beantwortet, sondern gleichzeitig bewiesen, dass nicht alles berechenbar ist. Es gibt Probleme, für die es keine Algorithmen gibt. Diesen werden wir uns als Nächstes zuwenden und dabei Turings Beweis genauer unter die Lupe nehmen.

Unentscheidbarkeit

Hilbert hatte unter dem Aspekt der Entscheidbarkeit gefragt, ob sich eine Beweiskette von den Axiomen bis zu einer vorgegebenen Aussage berechnen lässt. Anstatt „berechnen“ verwendete er den Ausdruck „systematisches Verfahren“, nur war zum damaligen Zeitpunkt gar nicht formal geklärt, was damit gemeint ist. Heute würden wir sagen: „Logisch, er meint einen Algorithmus!“, doch es war Turings Verdienst, dass er mit der Konzeption seiner Turing-Maschine den Vorgang des menschlichen Rechnens formalisierte, ihn in einem Maschinenmodell exakt beschrieb und damit auch den Grundstein für eine Definition des Begriffs Algorithmus legte.

Eine weitere zentrale Konzeption in Turings Arbeit war die Beschreibung eines Problems, das nicht berechenbar ist: das Halteproblem. Das Halteproblem fragt, ob eine Turing-Maschine irgendwann stoppt, wenn sie mit einem spezifischen Programm und einer spezifischen Eingabe gefüttert wird. Verallgemeinert lautet die Aufgabe, ein Programm zu schreiben, das für ein beliebiges Testprogramm mit einer beliebigen Testeingabe herausfindet, ob es jemals abgearbeitet sein wird oder den Prozessor zu einer endlosen Ausführung von Instruktionen veranlasst. Dabei gehen wir von einem Szenario aus, in dem das Testprogramm stoppt, sobald es die gewünschte Ausgabe produziert hat. Wenn Sie wissen möchten, warum für das Halteproblem kein entsprechendes Lösungsprogramm existieren kann, dann schauen Sie in Bsp. 8.15 hinein.

Beispiel 8.15: Warum das Halteproblem nicht berechenbar ist

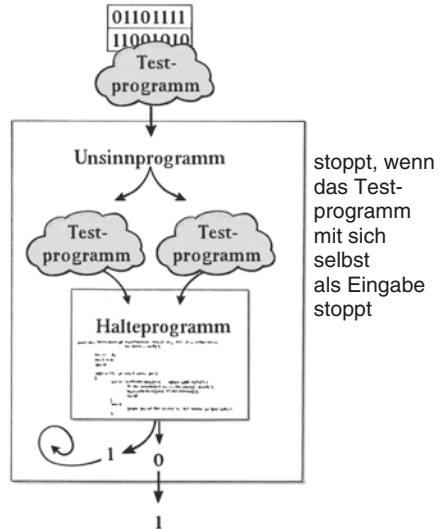
Nehmen wir an, das Halteproblem sei berechenbar und es gäbe ein dazugehöriges Programm namens Halteprogramm (siehe Abb. 8.24 oben links). Das Halteprogramm erhält als Eingabe das zu überprüfende Testprogramm in einer geeigneten binären Codierung zusammen mit einer 0-1-Zeichenkette, die die Testeingabe für das Testprogramm darstellt.

Wir konstruieren nun ein anderes Programm namens Unsinnprogramm (in Abb. 8.24 oben rechts), das wie folgt arbeitet: Es erhält ein Programm als Eingabe und startet anschließend das Halteprogramm, wobei das Programm ebenso als Testprogramm wie auch als Testeingabe für das Halteprogramm

Funktionsweise des Halteprogramms



Konstruktion des Unsinnprogramms



zwei Widersprüche bei der Verwendung des Unsinnprogramms

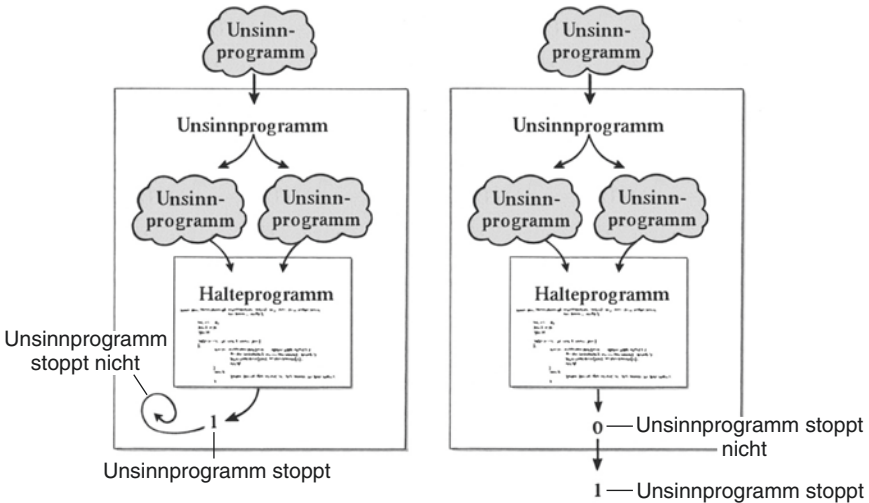


Abb. 8.24 Das Halteproblem und der Beweis, warum es nicht berechenbar ist.
(© Susanne Staubli, Eckart Zitzler)

dient. Das erscheint ziemlich unsinnig: Warum sollte ich ein Programm starten und den Binärcode selbst als Eingabe für dieses Programm verwenden? Es stimmt, das macht keinen Sinn, aber es ist zumindest möglich. Wir werden gleich sehen, warum es unserem Zweck dient. Also, das Unsinnprogramm ruft das Halteprogramm auf, und je nach dem Ergebnis, das das Halteprogramm

zurückliefert, verhält sich das Unsinnprogramm anders. Ist die Antwort vom Halteprogramm „ja“ (also das Testprogramm stoppt, wenn es als Eingabe den Binärcode von sich selbst erhält), dann geht das Unsinnprogramm in eine Endlosschleife und stoppt nicht; ist die Antwort „nein“, so gibt das Unsinnprogramm „ja“ aus.

Ok, das klingt ziemlich mysteriös. Doch der Clou kommt jetzt (vgl. Abb. 8.24 unten). Denn nun rufen wir das Unsinnprogramm auf und nehmen als Eingabe den Binärcode vom Unsinnprogramm selbst. Das Unsinnprogramm lässt daraufhin per Halteprogramm prüfen, ob das Unsinnprogramm mit der Eingabe Unsinnprogramm stoppt oder nicht. Wenn das Halteprogramm „ja, es stoppt“ antwortet, dann geht anschließend das Unsinnprogramm in eine Endlosschleife, stoppt also nicht. Wenn das Halteprogramm „nein, es stoppt nicht“ antwortet, dann spuckt das Unsinnprogramm „ja“ aus, stoppt also. Beide Situationen führen letztlich zu einem Widerspruch. Der einzige Ausweg aus dieser Zwickmühle ist das Infragestellen unserer Voraussetzung: So ein Halteprogramm kann es nicht geben.

Nun gut, aber was hat das Halteproblem mit Hilberts Frage zu tun? Der Verbindung lautet: Der Berechnungsverlauf einer Turing-Maschine lässt sich als mathematische (prädikatenlogische) Formel beschreiben, und wenn diese Formel beweisbar ist, dann gibt es auch diese Berechnung. Auf dieser Überlegung aufbauend hat Alan Turing gezeigt: Wenn es ein systematisches Verfahren zur Entscheidbarkeit mathematischer Formeln gibt, dann gibt es auch ein Turing-Maschinenprogramm zur Lösung des Halteproblems. Die Erkenntnisse von Turing – gleichzeitig und unabhängig wurden sie auch von Alonzo Church, dem Namensgeber der Churchschen These, gewonnen – waren bahnbrechend und haben die Theoretische Informatik begründet. Dass es ein Entscheidungsproblem gibt, das nicht berechenbar – man sagt auch: unentscheidbar – ist, zeigt Nutzungsgrenzen der automatisierten Informationsverarbeitung auf. Und dass es diese gibt, ist eigentlich nicht verwunderlich, denn die Menge aller Probleme ist mächtiger, umgangssprachlich „größer“ als die Menge der möglichen Turing-Maschinenprogramme. Die Busy-Beaver-Funktion ist eine weitere, etwas andere nicht berechenbare Funktion (siehe Anm. 8.7).

Anmerkung 8.7: Der eifrige Biber

Die Busy-Beaver-Funktion (deutsch auch Fleißiger-Biber-Funktion) wurde vom ungarischen Mathematiker Tibor Radó 1962 erfunden. Diese Funktion nimmt eine Zahl n als Eingabe und gibt als Funktionswert zurück, wie viele Einsen eine Turing-Maschine mit n Tabellen bzw. Zuständen maximal auf das Band schreiben kann – es ist das Maximum über alle möglichen Turing-Maschinen-Programme mit n Tabellen bzw. Rechenoperationen. Diese Funktion wächst

enorm schnell, schneller als jede berechenbare Funktion. Ihre Werte sind derzeit nur für die Eingabe 1 bis 4 exakt bekannt, die entsprechenden Funktionswerte sind 1, 3, 6 und 13. Für $n = 5$ weiß man nur, dass der Funktionswert größer gleich 4098 ist.

Sie können jetzt zu Recht fragen: Ist das Ganze nicht einfach nur theoretische Spielerei? Leider nicht, das Halteproblem ist in der Tat von hoher praktischer Relevanz. Eine beliebte Fehlerquelle beim Programmieren besteht nämlich darin, unbeabsichtigt sogenannte Endlosschleifen einzubauen, bei denen ein Programm einen Block an Instruktionen ständig wiederholt und quasi nicht zum Ende kommt. Wenn man mit einem Halteprogramm automatisch entscheiden könnte, ob ein Programm einen solchen Fehler enthält oder nicht, wäre viel gewonnen. Das Gleiche gilt für alle anderen Arten von Programmierfehlern, und auch hier sind der automatisierten Informationsverarbeitung Grenzen gesetzt: Henry Gordon Rice konnte bereits 1953 zeigen, dass es unmöglich ist, bestimmte nichttriviale Eigenschaften eines Programms mithilfe eines anderen Programms überprüfen zu lassen.

Doch es muss nicht gleich Hopfen und Malz verloren sein, wenn wir es mit einem nicht berechenbaren Problem zu tun haben. So ist es auch beim Halteproblem bzw. seiner Verallgemeinerung. Die Aussage von Rice ist ja nur, dass es keinen Algorithmus gibt, der für jedes mögliche Programm eine Eigenschaftsüberprüfung vornehmen kann. Für eine eingeschränkte Menge von Programmen ist dies jedoch in gewissem Maße möglich. Hundertprozentige Sicherheit gibt es allerdings nicht. Denn wenn wir ein Programm entwickelt haben, das für bestimmte Programme die Korrektheit der Ausgabe überprüfen kann, wie können wir dann sicher sein, dass dieses Programm auch korrekt funktioniert? Dann bräuchten wir wiederum ein weiteres Korrektheitsprüfprogramm usw.

Von praktischen und unpraktischen Algorithmen

Der Gesichtspunkt der Berechenbarkeit, ob ein Problem überhaupt mit einem Computer lösbar ist, stand also am Anfang der Theoretischen Informatik. Je leistungsfähiger Computer, je komplexer die Probleme und je größer die Eingaben wurden, desto mehr rückte eine andere Frage in den Fokus, nämlich: Wenn ein Problem berechenbar ist, ist es dann auch praktisch lösbar, und wenn ja, wie schnell? Es ging dabei um die Laufzeit, aber auch um den Speicherplatzbedarf von Programmen. Man stieß zunehmend

auf Probleme, für die man zwar Algorithmen finden konnte, die auf einem Computer aber nicht einsetzbar waren, weil sie entweder nicht in vernünftiger Zeit Ergebnisse lieferten oder einen astronomischen Bedarf an Speicher erforderten. Überhaupt einen Algorithmus finden, ist also noch nicht alles. So rückte die Brauchbarkeit von Algorithmen in den Mittelpunkt des Interesses, und man begann Lösungsverfahren zu bewerten und zu vergleichen, und zwar aus theoretischer Warte. Doch wie bestimmt man die Zeitressourcen, die ein Algorithmus benötigt? Schauen wir uns das an – den Aspekt des Speicherplatzbedarfs werden wir hier nicht weiter verfolgen, weil er zumeist eine untergeordnete Rolle spielt.

Die Laufzeit eines Algorithmus, um eine Lösung zu einer bestimmten Problem Instanz zu berechnen, lässt sich als die Zeit definieren, die wir warten müssen, bis ein entsprechendes Programm auf einem Computer die korrekte Ausgabe generiert hat. Wir können ihm einen anderen Algorithmus gegenüberstellen, indem wir die Laufzeiten unter gleichen Rahmenbedingungen vergleichen. Dabei werden allerdings immer konkrete Implementierungen und Computersysteme gegeneinander abgewogen, die effektive Zeit zur Berechnung ist der Maßstab. Im Zusammenhang mit Algorithmen ist man jedoch häufig am Wesen des Verfahrens interessiert und möchte von Details wie dem verwendeten Computer, der Programmiersprache und der Kunstfertigkeit der Programmiererin abstrahieren. Man möchte vor allem wissen, wie sich die Laufzeit in Abhängigkeit von der Eingabegröße ändert. Dabei nimmt man recht grobe Abschätzungen vor, wie Bsp. 8.16 demonstriert.

Beispiel 8.16: Wie lange braucht ein Algorithmus?

Lassen Sie uns zwei Sortieralgorithmen gegenüberstellen und deren Laufzeit abschätzen (siehe Abb. 8.25). Wir werden einzig die Anzahl der erforderlichen Kartenvergleiche betrachten; diese Zahl ist ein guter Indikator für die Laufzeit, alle anderen Aspekte blenden wir aus. Beim ersten Algorithmus probieren wir alle Anordnungen der Karten durch und prüfen einfach, ob die aktuelle Anordnung sortiert ist oder nicht, und stoppen gegebenenfalls. Das entspricht dem Prinzip „Versuch und Irrtum“. Beim zweiten Algorithmus lesen wir die Karten erst auf und legen sie dann nacheinander auf den Tisch; dabei fügen wir die jeweilige Karte immer an der richtigen Position in der Kartenreihe auf dem Tisch ein.

Wenn wir nun grob die Anzahl der Schritte abschätzen wollen, dann können wir das beim ersten Algorithmus wie folgt: Bei vier Karten gibt es 4 mal 3 mal 2 mal 1, also 24 verschiedene mögliche Anordnungen, und um eine Anordnung zu prüfen, müssen wir vier Karten durchgehen. Im schlimmsten Fall benötigt die Variante „Versuch und Irrtum“ also 96 Vergleiche. Beim Ansatz des „Einfügens“ fügen wir die vier Karten nacheinander in die bereits

Versuch und Irrtum



Einfügen

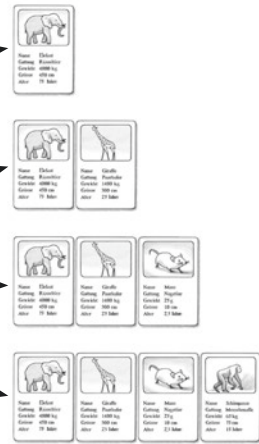


Abb. 8.25 Zwei Sortieralgorithmen im Vergleich. (© Susanne Staubli, Eckart Zitzler)

bestehende, immer größer werdende Reihe ein, und für das Einfügen müssen wir im schlimmsten Fall alle Karten in der Reihe anschauen, also maximal vier Vergleiche pro Einfügeoperation. Insgesamt kommen wir also auf höchstens 16 Vergleiche. Der Laufzeitunterschied zwischen den Algorithmen ist beachtlich und er nimmt zu, je mehr Karten betrachtet werden. Im ersten Fall ist die maximale Anzahl der Vergleiche gleich der Fakultät der Kartenanzahl multipliziert mit der Kartenanzahl, im zweiten Fall ist die Anzahl der Vergleiche höchstens gleich dem Quadrat der Kartenanzahl.

Das Beispiel zeigt auf, dass die Laufzeit sehr unterschiedlich mit der Eingabegröße ansteigen kann. Tatsächlich lassen sich verschiedene Wachstumsordnungen unterscheiden. Um diese zu illustrieren, lassen Sie uns das berühmte Szenario mit dem Reiskorn auf dem Schachbrett betrachten, das einem persischen Märchen entstammt. Auf dem linken unteren Feld befindet sich ein Reiskorn, und die Anzahl der Reiskörner auf den anderen Feldern ergibt sich jeweils aus der Anzahl der Körner auf dem vorherigen Feld – die Felder seien dabei von links nach rechts und von unten nach oben zu durchlaufen. In Abb. 8.26 sehen Sie unterschiedliche Schemata, wie

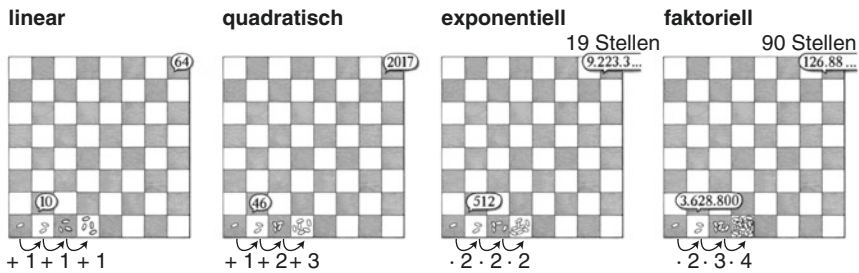


Abb. 8.26 Verschiedene Wachstumsordnungen, ausgehend von einem Reiskorn auf einem Schachbrettfeld. (© Susanne Staubli, Eckart Zitzler)

ausgehend vom ersten Feld sukzessive die weiteren Felder mit Reiskörnern belegt werden können.

Im ersten Fall links legen wir auf das nächste Feld immer ein Reiskorn mehr als auf das Vorgängerfeld. Da das Schachbrett 64 Felder besitzt, liegen auf dem letzten, dem obersten rechten Feld zum Schluss 64 Reiskörner. Wenn wir jedoch, wie im zweiten Beispiel von links gezeigt, auch die Anzahl der Reiskörner, die hinzukommen, permanent um eins erhöhen, dann steigt die Anzahl Reiskörner auf dem letzten Feld beträchtlich auf 2017. Wir können aber sogar noch weiter gehen: Wir können die Reiskörner jeweils verdoppeln, d. h., auf einem Feld liegen immer doppelt so viele Reiskörner wie auf dem vorherigen Feld. Zum Schluss haben wir einen Berg von Reiskörnern, denn die Anzahl an Reiskörnern auf dem letzten Feld ist eine Zahl mit 19 Stellen. In diesem Fall haben wir ein exponentielles Wachstum. Und auch das können wir weiterspinnen, indem wir den Faktor, um den erhöht wird, in jedem Schritt um eins erhöhen (wie vorher bei der Addition). In diesem Fall lässt sich das Wachstum über die Fakultätsfunktion beschreiben, am Ende hätten wir eine schier unvorstellbare Anzahl, die die Zahl der Atome im Universum übersteigt. Wir sehen: Es macht einen großen Unterschied, wie die Zunahme ist. Diese Varianten repräsentieren verschiedene Wachstumsordnungen und werden in der Informatik Komplexitätsklassen genannt (siehe Bsp. 8.17).

Beispiel 8.17: Grenzen des Wachstums und die O-Notation

In der Informatik spielen vor allem die drei linken, in Abb. 8.26 gezeigten Szenarien eine wichtige Rolle. Bei einem linearen Wachstum ist die Laufzeit proportional zur Größe der Eingabe. Ein polynomiellles Wachstum liegt vor, wenn die Laufzeit proportional ist zur Größe der Eingabe potenziert mit einer Konstanten; ist die Konstante gleich 2, spricht man von quadratischem

Wachstum, ist sie gleich 3, ist das Wachstum kubisch. Und von exponentiellem Wachstum ist die Rede, wenn die Laufzeit proportional ist zu einer Konstante potenziert mit der Größe der Eingabe. Wachstumsordnungen werden häufig mit der O-Notation (gesprochen „Groß-O-Notation“) angegeben, die für eine gegebene Funktion eine Komplexitätsklasse definiert; alle Laufzeiten, die nicht schneller wachsen als die angegebene Funktion für immer größer werdende Eingaben, fallen in die entsprechende Komplexitätsklasse.

Natürlich kann es ganz unterschiedliche Eingaben mit der gleichen Größe geben, manche resultieren in einer langen Laufzeit, andere vielleicht in einer kurzen. Deswegen muss man bei der Laufzeit jeweils genau spezifizieren, was man meint: den besten, den schlechtesten und den durchschnittlichen Fall. In Bsp. 8.16 haben wir die Sortieralgorithmen aus der Perspektive des Worst-Case-Szenarios betrachtet.

Informatikerinnen sind also an den Größenordnungen interessiert, in welchen sich die Laufzeiten von Algorithmen bewegen. Und klar ist: Exponentiell ist immer schlecht. Man sucht nach effizienten Algorithmen, und darunter versteht man allgemein Algorithmen, deren Laufzeit polynomiell in der Eingabegröße beschränkt ist. Das mag Ihnen zu Recht absurd erscheinen, denn es macht in der Praxis einen erheblichen Unterschied, ob die Laufzeit linear, quadratisch, kubisch usw. wächst. Doch exponentielle Laufzeit ist noch viel schlimmer, da brauchen bei entsprechender Eingabegröße selbst die schnellsten Computer Jahre. Klar, man kann einen Algorithmus geschickt und ungeschickt in ein Programm umsetzen, aber wenn die Laufzeit exponentiell ist, wird es schwierig. Und jetzt kommt's: Für viele praktisch relevante Probleme kennt man keine Algorithmen, die das Problem (für jede denkbare Eingabe!) schneller als mit exponentieller Laufzeit lösen. Das Hamilton-Pfad-Problem ist so ein Beispiel. Und das ist in der Tat irritierend.

P oder nicht P, das ist hier die Frage

Die Frage, die die Fachwelt seitdem umtreibt, lautet: Muss man für diese Probleme einfach noch weitertüfteln oder lässt sich zeigen, dass für sie keine polynomiellen Algorithmen existieren? Bislang hat man sich daran die Zähne ausgebissen. Und weil man nicht weiterkam, fing die Informatikgilde in den 1970er-Jahren an, Probleme miteinander zu vergleichen und gegenüberzustellen. Man stellte dabei fest, dass einige Probleme miteinander verwandt zu sein schienen. Warum? Weil sich ein Problem auf ähnliche Weise wie ein anderes lösen ließ: Der Algorithmus für das zweite Problem kann zur Lösung des ersten eingesetzt werden. Schauen wir uns das konkret an.

Betrachten wir dazu das Problem des Handlungsreisenden. Ziel ist eine Rundreise zu finden, die in einer Region oder einem Land alle Städte genau einmal besucht und deren Gesamtlänge minimal ist. Wenn wir für dieses Problem einen polynomiellen Algorithmus hätten, dann könnten wir auch das Hamilton-Pfad-Problem effizient lösen. Die Idee ist einfach: Wir konvertieren die Eingabe für das Hamilton-Pfad-Problem zunächst einmal in eine entsprechende Eingabe für das Handlungsreisenden-Problem, dann wenden wir den Handlungsreisenden-Algorithmus an und die Ausgabe rücktransformieren wir in eine Ausgabe für unsere Hamilton-Pfad-Problem Instanz. Bsp. 8.18 erläutert im Detail, wie und warum das funktioniert.

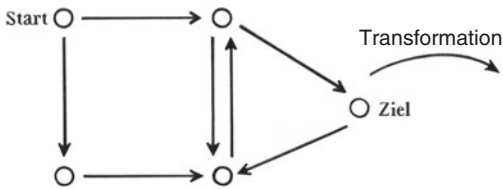
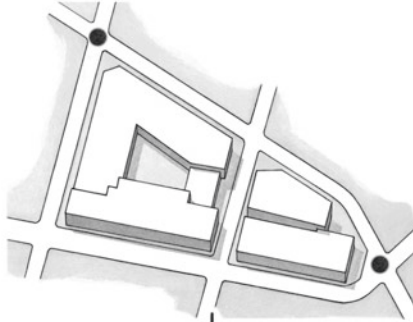
Beispiel 8.18: Hamilton wird Handlungsreisender

Betrachten Sie das in Abb. 8.27 skizzierte Szenario. Links sehen Sie Ein- und Ausgabe für das Hamilton-Pfad-Problem, rechts Ein- und Ausgabe für das Problem des Handlungsreisenden. Die Probleme sind sich ähnlich, auch wenn es zwei Unterschiede gibt: Bei Letzterem sind alle Städte bzw. Kreuzungen von jeder anderen Stadt bzw. Kreuzung erreichbar und die Route muss wieder beim Startpunkt enden. Die Ähnlichkeiten lassen sich nutzen, um den Algorithmus für das Handlungsreisenden-Problem zur Lösung des Hamilton-Pfad-Problems einzusetzen, und zwar wie folgt.

Wir transformieren den Stadtplan für das Hamilton-Pfad-Problem in eine entsprechende Straßenkarte für das Handlungsreisenden-Problem, wobei beide als Graphen dargestellt sind. Für jede Kante im Stadtplangraphen fügen wir eine Kante im Straßenkartengraphen hinzu und schreiben sie mit einer Distanz von 1 km an. Danach vervollständigen wir den Straßenkartengraphen: Wir fügen von jedem Knoten zu jedem anderen Knoten eine Kante hinzu – insofern nicht schon eine besteht – und schreiben diese Kanten mit einer Distanz von 3 km an. Zum Schluss ändern wir noch die Distanzen von allen Kanten, die direkt vom Ziel zum Start führen; sie werden neu mit 2 km angeschrieben.

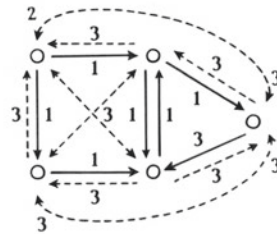
Auf den erzeugten Straßenkartengraphen wenden wir nun den Algorithmus zum Lösen des Handlungsreisenden-Problems an. Wenn die gefundene Rundreise eine Länge von 6 km aufweist, können wir sie einfach in einen Hamilton-Pfad umwandeln, indem wir die Strecke vom Ziel zum Start entfernen; ansonsten gibt es keinen Hamilton-Pfad. Warum das so ist? Gibt es einen Hamilton-Pfad für den Stadtplan, dann gibt es auch eine Rundreise von 6 km Länge, die einerseits vier Wegstrecken mit je 1 km Länge vom Start zum Ziel enthält und andererseits die Wegstrecke vom Ziel zum Start mit 2 km Länge. Wenn jedoch die Wegstrecke vom Ziel zum Start nicht Teil der Rundreise ist, dann umfasst diese mindestens 7 km (jede andere Strecke vom Ziel hat ja eine Länge von 3 km). Eine minimale Rundreise mit einer Länge von 7 km oder mehr kann folglich keinen Hamilton-Pfad enthalten.

Hamilton-Pfad-Problem



Gesucht: Weg vom Start zum Ziel, der alle Knoten einmal berührt

Problem des Handlungsreisenden



Gesucht: Rundreise minimaler Länge, die jeden Knoten einmal besucht

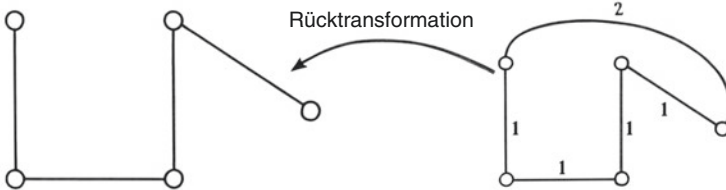


Abb. 8.27 Wie sich das Hamilton-Pfad-Problem auf das Problem des Handlungsreisenden zurückführen lässt. (© Susanne Staubli, Eckart Zitzler)

Insofern die Transformation von Eingabe zu Eingabe und die Rücktransformation von Ausgabe zu Ausgabe effizient berechenbar sind, lässt sich ein Problem auf ein anderes zurückführen und mithilfe des entsprechenden Lösungsverfahrens bewältigen. Man spricht auch von polynomieller Reduktion. Die Entdeckung der Reduktionsmethode war ein gewaltiger Fortschritt, weil man nun Probleme nach Schwierigkeitsgrad relativ zueinander anordnen konnte. Das Problem des Handlungsreisenden ist z. B. mindestens so schwierig wie das Hamilton-Pfad-Problem. Mit der Zeit konnte man immer mehr solcher Beziehungen zwischen verschiedenen Problemen aufzeigen, doch den Graben zwischen den polynomiell lösbaren oder den hartnäckigen Problemen überwand man nicht. Vielfach konnte man schwierige auf

schwierige Probleme reduzieren, aber schwierige auf leichte zurückzuführen, gelang nicht. Die Ausgangsfrage blieb also unbeantwortet.

Um der Antwort näher zu kommen, überlegte man sich, wie sich denn die schwierigen Probleme charakterisieren lassen. Und tatsächlich: Die meisten lassen sich lösen, wenn systematisch alle Möglichkeiten nach dem Prinzip „Versuch und Irrtum“ durchprobiert werden. Die Prüfung, ob ein Lösungskandidat tatsächlich eine Lösung ist, kann in der Regel effizient durchgeführt werden. Beim Hamilton-Pfad-Problem hatten wir es ja gesehen: Es ist leicht festzustellen, ob eine bestimmte Kantenfolge effektiv einen Hamilton-Pfad darstellt. Doch weil der Suchraum bei diesen Problemen exponentiell mit der Eingabegröße wächst, führt die „Versuch und Irrtum“-Strategie zu einer exponentiellen Laufzeit. Aus dieser Beobachtung heraus hat man sich eine etwas obskure Variante der Turing-Maschine ausgedacht, die über ein Orakel verfügt. Ja, Sie haben richtig gehört: ein Orakel. Wenn Sie vermuten, dass hier ein bisschen zu viel Wein im Spiel war, so kann ich das nicht widerlegen, aber selbst bei nüchternem Kopf ist die Idee dahinter äußerst nützlich. Das Orakel kapselt den Teil des systematischen Suchens bei „Versuch und Irrtum“ ab, es kann nämlich Lösungen vorschlagen, insofern welche existieren. Das Programm muss nur noch prüfen, ob die vom Orakel vorgeschlagene Zeichensequenz tatsächlich eine Lösung repräsentiert, und die Lösung dann ausgeben. Bsp. 8.19 zeigt, was es mit dieser Orakel-Maschine im Einzelnen auf sich hat.

Beispiel 8.19: Wie eine nichtdeterministische Turing-Maschine funktioniert

Das Orakel schreibt, bevor die Turing-Maschine startet, einen Orakelspruch, bestehend aus Nullen und Einsen, auf das Band. Und wie das mit Orakeln ist, so ist der Spruch etwas orakelhaft: Er kann eine Lösung repräsentieren oder auch nicht. Sicher ist nur, dass das Orakel, falls es effektiv eine Lösung gibt, auch eine Lösung hinschreibt, doch wenn nicht, dann schreibt das Orakel irgendetwas auf das Band. Das Turing-Maschinenprogramm kann das Orakel verwenden und muss nur noch prüfen, ob es sich dabei um eine gültige Lösung handelt. Man nennt so eine Turing-Maschine eine nichtdeterministische Turing-Maschine, weil die Ausführung vom Orakel abhängt. Bei gleicher Eingabe können verschiedene Rechenwege entstehen, es ist also bei gegebener Eingabe nicht determiniert, welche Rechenoperation auf welche folgt (ohne Kenntnis des Orakels). Die klassische Turing-Maschine aus Kap. 1 ist hingegen eine deterministische Turing-Maschine.

Wozu die Einführung eines Orakels gut sein soll? Erstens, und nur damit keine Missverständnisse auftreten: Bei der nichtdeterministischen Turing-Maschine handelt es sich um ein Gedankenspiel, sie lässt sich nicht bauen, sondern hilft nur, Probleme zu charakterisieren. Das Orakel steht für den Teil des Ausprobierens, den man so ausblendet. Auf diese Weise lassen sich viele schwierige Probleme in polynomieller Zeit lösen – wenn also das Orakel den

entscheidenden Tipp gibt. Beim Hamilton-Pfad-Problem geht das auch, wie es in Abb. 8.28 für die Entscheidungsvariante illustriert ist. Der Orakelspruch wird im ersten Schritt als Auswahl an Kanten interpretiert, und im zweiten Schritt muss dann nur noch überprüft werden, ob die Kantenauswahl effektiv einen Hamilton-Pfad darstellt. Dieses Prüfverfahren geht in polynomieller Zeit von-statten.

Viele relevante, schwierige Probleme lassen sich mit so einer nichtdeterministischen Turing-Maschine polynomiell berechnen – die Probleme, für die es effiziente Algorithmen gibt, ja sowieso, da kann das Orakel einfach ignoriert werden. Die Frage lautet nun: Ist eine nichtdeterministische Turing-Maschine mächtiger als eine deterministische? Die Antwort scheint auf der Hand zu liegen („Ja, logisch!“), aber man weiß es schlichtweg nicht, sondern vermutet es nur. In der Informatik hat man diese offene Frage mit der Formel „ $NP = P?$ “ auf den Punkt gebracht. Dabei steht NP für die Menge aller Probleme, deren Entscheidungsvarianten mit einer nichtdeterministischen Turing-Maschine (dafür steht das N) in polynomieller Laufzeit (dafür steht das P) lösbar sind. Die Menge P steht analog für die Menge aller Probleme, deren Entscheidungsvarianten von einer

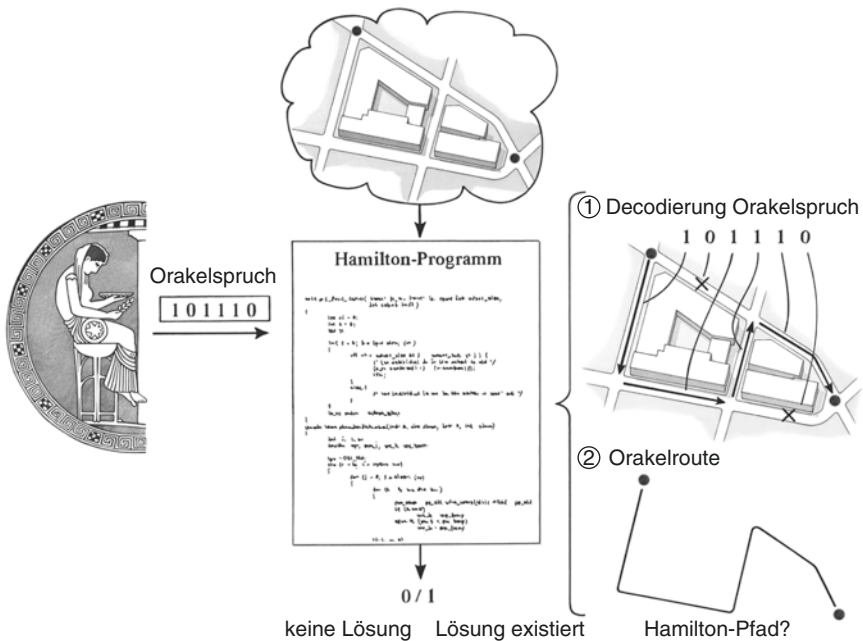


Abb. 8.28 Eine Turing-Maschine mit Orakel. (© Susanne Staubli, Eckart Zitzler)

klassischen deterministischen Turing-Maschine in polynomieller Laufzeit bearbeitet werden können. Es ist klar, dass P in NP enthalten ist, aber sind alle Probleme in NP auch in P ? Ist also $NP = P$? Wenn Sie diese Frage beantworten, können Sie sicher sein, dass Sie die Frage der Informatik beantwortet haben und den Turing-Award erhalten werden.

Jetzt sind wir immer noch kein Stück weiter, oder? Doch, man ist weiter. Im Jahr 1971 stellte der Kanadier Stephen A. Cook einen Beweis vor, der zeigte, dass es ein Problem in NP gibt, auf das sich alle Problem in NP polynomiell reduzieren lassen. Sprich: Findet jemand für Cooks Problem einen effizienten Algorithmus, so hat ist gleichzeitig einen effizienter Algorithmus für alle anderen Probleme in NP gefunden, woraus $NP = P$ folgern würde. Bei so einem Problem spricht man von einem NP -vollständigen Problem. Wenn es Sie interessiert, Anm. 8.8 zeigt die Hintergründe auf.

Anmerkung 8.8: Die schwierigsten der schwierigen Probleme

Cook hat gezeigt, dass das SAT-Problem – SAT ist eine Abkürzung für das englische Wort *satisfiability*, also *Erfüllbarkeit* – NP -vollständig ist, d. h., jedes Problem in NP lässt sich polynomiell auf das SAT-Problem reduzieren. Bei der Entscheidungsvariante des SAT-Problems (siehe Abb. 8.29) geht es darum, für eine nur aus UND- und ODER-Gattern bestehende Schaltung mit beliebig vielen Eingängen und einem Ausgang zu entscheiden, ob es eine Eingabebelegung gibt, sodass am Ausgang eine 1 steht. Die Idee für den Beweis ist ebenso einfach wie genial, die Ausführung hingegen kompliziert und langwierig. Beschränken wir uns also auf die Idee.

Nehmen wir an, wir wollten das Hamilton-Pfad-Problem lösen. Dann können wir ja ein Programm für eine nichtdeterministische Turing-Maschine schreiben, so wie oben skizziert. Und wenn nun ein Graph vorliegt, für den ein Hamilton-Pfad existiert, dann gibt es ja für unser Turing-Maschinenprogramm sowohl einen Orakelspruch als auch einen Berechnungsweg, sodass die Ausgabe 1 ist. Jetzt kommt der Trick: Cook hatte die Idee, das gesamte Turing-Maschinenprogramm in eine Schaltung umzusetzen, die genau dann 1 ausspuckt, wenn es so einen Berechnungsweg gibt; den Orakelspruch baute er einfach mittels mehrerer Zusatzeingabeleitungen für die Schaltung ein. Mit so einer Transformation brauchen wir nur noch mittels des SAT-Algorithmus zu prüfen, ob die Schaltung eine Lösung hat, und in diesem Fall gibt es auch einen Hamilton-Pfad.

Dieses Prinzip, eine nichtdeterministische Turing-Maschine zusammen mit dem Orakelspruch als Schaltung darzustellen, lässt sich verallgemeinern, und damit kann jedes Problem in NP auf das SAT-Problem reduziert werden. Es kann zudem gezeigt werden, dass die Umwandlung in polynomieller Zeit möglich ist. Und es kommt noch etwas hinzu: Jedes Problem A , auf das sich das SAT-Problem reduzieren lässt, ist selbst wiederum NP -vollständig. Denn jedes Problem kann erst auf SAT und dann auf A reduziert werden. Die NP -vollständigen Probleme bilden eine Menge für sich, denn sie lassen sich auf sich untereinander reduzieren.

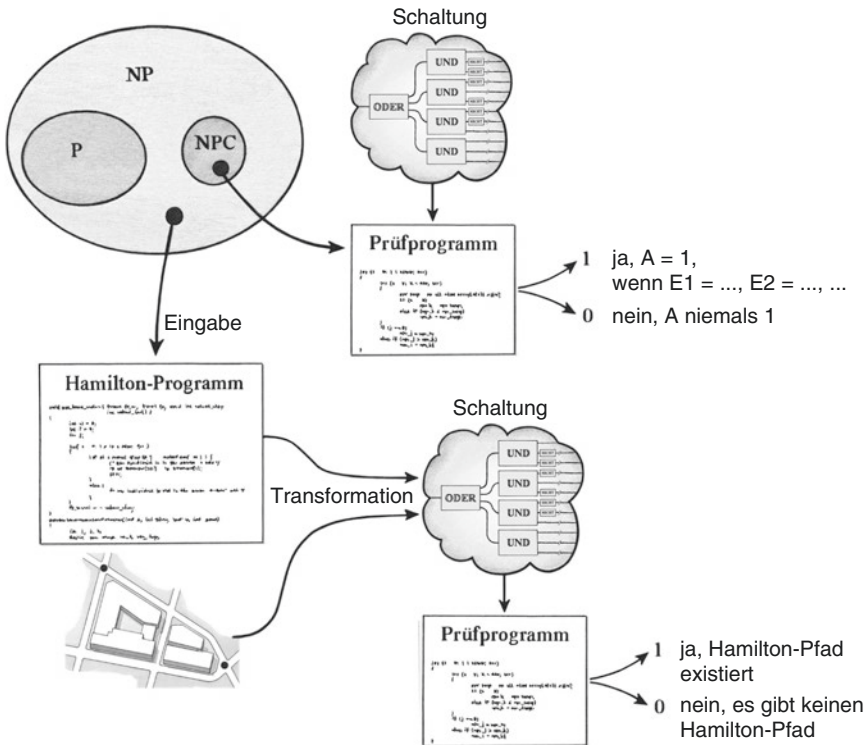


Abb. 8.29 Das Konzept der NP-Vollständigkeit. (© Susanne Staubli, Eckart Zitzler)

Viele bekannte Probleme sind (in ihrer Entscheidungsvariante) NP-vollständig, vom Hamilton-Pfad-Problem über das Rucksackproblem bis zum Problem des Handlungsreisenden. Wir müssten nur für eines dieser Probleme einen effizienten Algorithmus finden, und schon wären alle Probleme in NP effizient lösbar. Aber selbst das ist bislang nicht gelungen. Daher vermutet man, dass NP größer ist als P und alle NP-vollständigen Probleme nicht in P liegen. Mit anderen Worten: Wenn ein Problem NP-vollständig ist, gilt es als schwieriges Problem und man wird keinen effizienten Algorithmus konstruieren können – insbesondere auch nicht für die Such- und Optimierungsvarianten. Entsprechend versuchen Informatiker, wenn sie mit einem neuen hartnäckigen Problem konfrontiert sind, zunächst einmal herauszufinden, ob sich ein NP-vollständiges Problem auf dieses Problem reduzieren lässt, ob also die Eigenschaft der NP-Vollständigkeit gegeben ist. So etwas ist gut zu wissen, denn es verhindert, dass man sich umsonst das Hirn zermartert.

Heißt das nun in der Praxis, das Problem des Handlungsreisenden beispielsweise lässt sich gar nicht in vernünftiger Zeit lösen? Nein, das heißt es nicht. Zum einen sind nicht alle Probleminstanzen gleich schwierig, für viele Eingaben lassen sich durchaus optimale Lösungen generieren. Zum anderen können Heuristiken, z. B. eine gierige Strategie, gute Resultate liefern, insofern wir nicht den Anspruch haben, in jedem Fall das Optimum finden zu wollen. Vielen kniffligen Optimierungsproblemen ist auf solchen Wegen beizukommen, dennoch gibt es jene Probleme, bei denen man tatsächlich davon ausgeht, dass sie nicht in praktikabler Zeit gelöst werden können. Letzteres lässt sich auch zu einem Vorteil ummünzen; beim Public-Key-Verschlüsselungsverfahren RSA nutzt man aus, dass das Problem der Primfaktorzerlegung großer Zahlen als praktisch unlösbar gilt.

Fazit

Mit der Computer-Hardware hat der Mensch sein Denkorgan extern ausgebaut und kognitive Leistungen ausgelagert. Mit der Computer-Software ist es ihm möglich, dieses Werkzeug zur Lösung unterschiedlichster Aufgaben und Probleme einzusetzen. Dieses Kapitel hat aufgezeigt, dass die Entwicklung geeigneter Programme eine Welt für sich ist: die Welt der Algorithmen. Algorithmen sind präzise Handlungsanweisungen zur Verarbeitung von Informationen und sie entscheiden, ob und wie gut ein Computer eine Aufgabe bewältigen kann. Wir haben verschiedene Strategien für den Algorithmenentwurf kennengelernt, um so verschiedenste Arten von Problemen automatisiert zu lösen. Algorithmen können sogar so konzipiert werden, dass Computer selbstständig werden. Wir haben gesehen, wie Programme anpassungsfähig werden können und rudimentäres Lernen möglich ist, wie Programme Spielstrategien umsetzen können und wie menschenverständliche Sprachkonstrukte erkannt werden können, z. B. im Zusammenhang mit Programmiersprachen. Es ist aber auch klar geworden, dass dem Einsatz von Computern Grenzen gesetzt sind: Es gibt Probleme, die nicht berechnet werden können. Und für viele praktisch relevante Probleme geht man davon aus, dass es keine effizienten Algorithmen gibt und sie höchstens näherungsweise gelöst werden können. Ob dies tatsächlich so ist oder ob bislang nur die zündenden Ideen für bessere Algorithmen fehlten, ist eine der zentralen offenen Fragen in der Theoretischen Informatik.

Zum Nach- und Weiterlesen

- Cormen, T. H., Leiserson, C. E., Rivest, R., & Stein, C. (2013). *Algorithmen – Eine Einführung* (4. Aufl.). München: De Gruyter Oldenbourg.
- Doxiadis, A., & Papadimitriou, C. H. (2010). *Logicomix: Eine epische Suche nach der Wahrheit* (1. Aufl.). Hamburg: Atrium.
- Gopnik, A. (2010). *Kleine Philosophen: Was wir von unseren Kindern über Liebe, Wahrheit und den Sinn des Lebens lernen können*. Berlin: Ullstein.
- Harel, D., & Feldman, Y. (2006). *Algorithmik: Die Kunst des Rechnens*. Berlin: Springer.
- Hoffmann, D. W. (2013). *Grenzen der Mathematik: Eine Reise durch die Kerngebiete der mathematischen Logik* (2. Aufl.). Berlin: Springer.
- Hoffmann, D. W. (2015). *Theoretische Informatik* (3. Aufl.). München: Hanser.
- Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2011). *Einführung in Automatentheorie, Formale Sprachen und Berechenbarkeit* (3. Aufl.). München: Pearson Studium.
- Kappeler, P. M. (2017). *Verhaltensbiologie* (4. Aufl.). Berlin: Springer.
- Ottmann, T., & Widmayer, P. (2012). *Algorithmen und Datenstrukturen* (5. Aufl.). Heidelberg: Spektrum Akademischer Verlag.
- Penrose, R. (2002). *Computerdenken: Die Debatte um Künstliche Intelligenz, Bewußtsein und die Gesetze der Physik*. Heidelberg: Spektrum Akademischer Verlag.
- Robertson, J., & Webb, W. (1998). *Cake-cutting algorithms: Be fair if you can*. Natick: Peters.

9

Im Licht der Evolution

Überblick

In diesem letzten Kapitel steht die Software im Mittelpunkt, genauer gesagt: die Software-Technik. Wir werden sie aus drei Blickwinkeln beleuchten und dabei die Evolution als thematisches Grundgerüst einsetzen. Zuerst geht es um das Entstehen: wie sich die Rolle der Software verändert hat und sich schichtenartig immer leistungsfähigere und komplexere Software-Strukturen gebildet haben. Dabei werden wir unter dem Stichwort der Schad-Software auch negative Entwicklungen diskutieren. Danach ist das Entwickeln an der Reihe: Sie lernen die Prinzipien kennen, nach denen Verarbeitungsabläufe als Programme formuliert und große Programmsysteme gebaut werden. Und schließlich dreht es sich um das Entdecken, nämlich wie speziell zugeschnittene Software-Systeme helfen, die biologische Evolution zu verstehen. Sie werden eine Vorstellung davon bekommen, wie genetische Stammbäume mithilfe von Computern rekonstruiert werden können. Dass zudem die Evolution als Vorbild für Software dienen kann, sehen Sie dann am Ende des Kapitels.

Der Erfolg des Computers liegt in seiner Universalität begründet. Weil Verarbeitungsabläufe mittels eines Maschinencodes dargestellt und – wie sonstige Daten – flexibel im Hauptspeicher abgelegt werden können, lässt er sich in verschiedensten Anwendungsszenarien einsetzen. Die Software stellt also die Hardware für eine bestimmte Aufgabe ein, allerdings muss sie entsprechend konstruiert werden: Die zu verarbeitenden Informationen sind zu codieren, die resultierenden Daten intern zu strukturieren und für die Berechnungen müssen adäquate Algorithmen gefunden werden. Mit den konzeptuellen Grundlagen dafür haben wir uns in den vorangegangenen Kapiteln beschäftigt. Doch letztlich müssen all

diese Überlegungen in ein konkretes Programm gegossen und in eine Folge von Maschineninstruktionen umgesetzt werden. Und je komplexer die Anwendung ist, desto weniger kann man von einem Programm sprechen und desto mehr hat man es mit einem Software-System zu tun, das aus einer Vielzahl von Programmen besteht. Daher haben sich bestimmte Vorgehensweisen etabliert.

Das Gebiet, welches sich mit der systematischen Erstellung von Software-Systemen auseinandersetzt, wird mit dem Begriff Software-Technik bezeichnet, im Englischen *Software Engineering*. Seine Methoden erstrecken sich auf weit mehr als die reine Programmierung, es geht auch um Spezifizieren, Konzipieren, Evaluieren und Dokumentieren. Im Folgenden erhalten Sie einen Einblick in die Grundlagen. Zunächst sehen wir uns die Evolution der Software an, u. a. auch zu welchen Problemen und Herausforderungen sie geführt hat. Anschließend widmen wir uns dem Handwerk der Programmierung und dem Bau großer Software-Systeme, insbesondere behandeln wir die verschiedenen Phasen der Systementwicklung. Zum Schluss demonstriere ich Ihnen die Umsetzung dieser Ideen in zwei realen Anwendungen, die sich um die biologische Evolution drehen. Dort werden Sie sehen, wie Biologie und Informatik sich konkret gegenseitig befruchten.

Vom Entstehen

Komplexe Software-Systeme entstehen nicht, indem man eine Folge von Maschinenbefehlen von oben nach unten niederschreibt. Da könnten Sie genauso gut eine Turing-Maschine programmieren und aus unzähligen Kartonbögen Schritt für Schritt einzelne Lochkarten ausschneiden und stanzen. Es bedarf einer gestuften Komposition, denn komplex heißt so viel wie vielschichtig, viele verschiedene Dinge umfassend. Schauen wir uns also an, wie Komplexität entsteht, und beginnen bei einem der vielschichtigsten Organismen überhaupt: dem Menschen.

Schicht für Schicht zur Komplexität

Der Homo sapiens hat eine lange Entwicklungsgeschichte hinter sich, und die begann schon, lange bevor die ersten Menschenartigen vor knapp 2 Mio. Jahren Afrika verließen. In verschiedenen Stufen der Evolution mussten zahlreiche Voraussetzungen geschaffen werden bzw. sich einstellen, und in jeder Stufe unterlagen den Veränderungsprozessen andere Mechanismen

(siehe Abb. 9.1). Die Elementarteilchen schlossen sich zu Atomen und zu den Elementen zusammen, Atome wiederum verbanden sich miteinander zu Molekülen. Als sich dann die Grundmoleküle des Lebens, u. a. DNA, RNA und Proteine, bildeten und sich ihr Zusammenspiel etablierte, war der Weg zu den ersten Organismen vorbereitet. Es sollte zwar noch etwas dauern, bis die Urzellen entstanden, doch ab da wurde aus der chemischen eine biologische Evolution. Aus Zellen entwickelten sich Zellverbände und Vielzeller mit spezialisierten Zellen, und aus diesen wiederum komplexe Lebewesen wie der Mensch. Grundlage dafür waren unzählige Veränderungen unseres genetischen Erbguts durch Mutationen und Rekombinationen, die sich in diversen Auslesevorgängen durchgesetzt haben. Schließlich brachte die kulturelle Evolution eine weitere Dynamik ins Spiel. Die Entwicklung einer Kultur im allgemeinen Sinne fußt auf unserer Fähigkeit, Verhaltensweisen durch Erfahrungen, Nachahmungen und Einsichten zu erlernen und zu verändern. Menschliche Gemeinschaften erfanden Hilfsmittel, bildeten Symbolsysteme wie Sprache und Schrift für die Vermittlung von Informationen und schufen auf deren Basis Traditionen für das Zusammenleben. Die Flexibilität, die unser Gehirn ermöglicht, hat unsere Entwicklung drastisch beschleunigt, denn durch sie können Verhaltensänderungen nicht nur von Generation zu Generation, sondern bereits auf individueller Ebene weitergereicht werden.

In all diesen Veränderungsprozessen finden wir dieselben Grundprinzipien. Da ist zum Beispiel das Prinzip des Zusammenschlusses: Elemente interagieren miteinander, schließen sich zusammen und bilden ein Ganzes mit neuen Eigenschaften. So entwickelten sich aus Einzellern mehrzellige

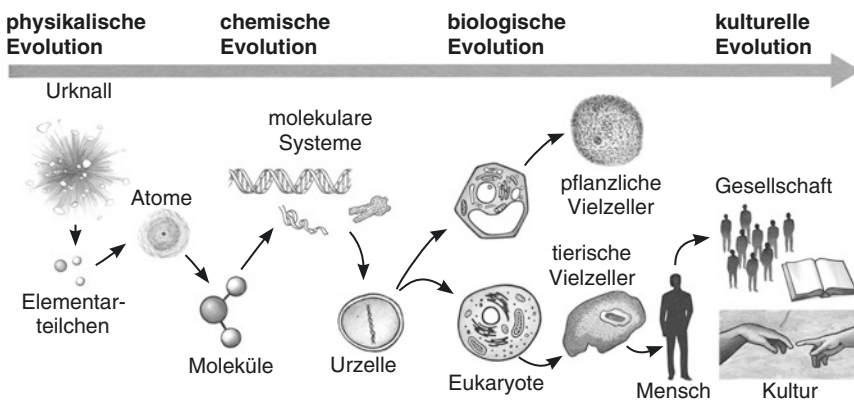


Abb. 9.1 Wie die Evolution – im Sinne verschiedener Prozesse – das Leben in all seiner Komplexität hervorgebracht hat. (© Susanne Staubli, Eckart Zitzler)

Organismen. Das Prinzip der Kapselung ist eng damit verknüpft: Häufig grenzen sich solche Verbunde gegenüber der Umwelt ab – um sich vor störenden Einflüssen zu schützen und den Selbsterhalt zu ermöglichen. Bei der Zelle ist es die Zellwand, die das Innen vom Außen trennt. Und schließlich ist da noch das Prinzip der Differenzierung, d. h., ganz unterschiedliche Systeme entstehen, obwohl ihre Elemente möglicherweise ähnlich sind. Vielzeller haben mit der Zeit immer differenziertere Substrukturen – Gewebearten oder Organe – entwickelt, die auf bestimmte Funktionen spezialisiert sind. Auf diese Weise gehen aus Elementen neue Systeme hervor, die eine neue Stufe der Komplexität schaffen. Das langfristige Resultat ist eine Komplexitätshierarchie. Die einzelnen Stufen sind nicht unbedingt klar trennbar, doch sie helfen uns, die Entstehungsgeschichte zu strukturieren und besser zu verstehen. Oder wie die Ameisenforscher Hölldobler und Wilson es formulieren (Holldöber und Wilson 2010, S. 8): „Alles Leben umfasst eine sich selbst reproduzierende Hierarchie von Ebenen. Biologie ist die Wissenschaft von den verschiedenen Ebenen.“

So wie komplexe natürliche Systeme entstehen, so werden auch komplexe künstliche Systeme geschaffen. Bei der Computer-Hardware haben wir das bereits mehrfach gesehen: Aus den Grundelementen elektrischer Stromkreise ergaben sich logische Gatterbausteine, aus diesen wiederum Bausteine zum Rechnen und Speichern, und letztere bildeten schließlich die Hauptbestandteile für einen Prozessor; aus einzelnen Computern wurde durch das Internet schließlich ein arbeitsteiliger Rechnerverbund. Bei der Entwicklung von Computer-Software gibt es ebenso allgemeine Entwurfsprinzipien, wie Bsp. 9.1 aufzeigt. Denn komplexe Programmsysteme können so umfangreich sein, dass sie Abermillionen von Verarbeitungsschritten umfassen und ein einzelner Mensch mehrere Hundert Jahre zu ihrer Erstellung benötigt. Sie lassen sich nur realisieren, wenn sie stufenweise aus überschaubaren und handhabbaren Programmteilen zusammengesetzt werden. Informatiker orientieren sich also an der natürlichen Vorgehensweise, auch wenn beim Entwurf von Software-Systemen weitere Aspekte wie z. B. Wartbarkeit oder Erweiterbarkeit hineinspielen.

Beispiel 9.1: Prinzipien der Software-Entwicklung

Dem Entwurfsprozess von Software-Systemen liegen verschiedene Grundsätze zugrunde, die sich im Laufe der Zeit aus Erfahrungen und Erkenntnissen herausgebildet haben. Drei zentrale Prinzipien, die an die vorher erwähnten Evolutionsprinzipien erinnern, sind folgende:

- **Modularisierung** Ein System wird in eine möglichst übersichtliche Anzahl von Komponenten untergliedert, die auf eine vordefinierte Art und Weise miteinander kommunizieren. Dabei versucht man die Module so zu wählen, dass jedes für andere Aufgaben verantwortlich ist und die Zuständigkeiten klar getrennt sind.
- **Kapselung** Eine Komponente präsentiert sich gegenüber den restlichen Systembestandteilen als eine BlackBox, in der Details der konkreten Umsetzung verborgen sind. Einzig das Verhalten und die Kommunikationskanäle zum Außen sind bekannt; was genau im Inneren der BlackBox geschieht, ist für die anderen Komponenten nicht ersichtlich.
- **Hierarchisierung** Die Komponenten selbst werden – sofern es ihr Umfang erfordert – sukzessive verfeinert und in weitere, miteinander verflochtene Untereinheiten aufgeteilt. Auf diese Weise ergibt sich eine Rangordnung von ineinander verschachtelten Systemkomponenten.

Viele weitere Prinzipien ließen sich hier aufzählen, doch ich möchte hier nur noch das Prinzip der Standardisierung explizit erwähnen. Dabei geht es um die Austauschbarkeit und Wiederverwendbarkeit von Komponenten. Sie wird durch normierte Erscheinungsbilder und Interaktionsmuster erreicht. Auch auf der Hardware-Ebene ist so eine Vereinheitlichung wesentlich: Wenn beispielsweise Gatterbausteine zu einer digitalen Schaltung kombiniert werden sollen, dann muss gewährleistet sein, dass sie alle mit denselben Spannungen und Anschlüssen arbeiten.

Die Software, die ein Computersystem konfiguriert, ist natürlicherweise geschichtet. Selbst wenn wir nur ein simples Programm zur Berechnung des größten gemeinsamen Teilers schreiben, bedarf es mehrerer Software-Ebenen, um es zur Ausführung zu bringen. Ein Betriebssystem ist notwendig, damit die Grundfunktionalitäten der Computer-Hardware genutzt werden können; ihm liegt in der Regel eine komplexe Software-Architektur zugrunde, bei der viele Programmteile ineinandergreifen und parallel ihre Dienste verrichten (vgl. Kap. 4). Und weil Programme so gut wie nie in Maschinencode formuliert, sondern in formalen Sprachen ausgedrückt werden, die eine Mischung aus natürlichsprachlichen Ausdrücken und mathematischen Formeln darstellen, braucht es noch etwas Weiteres. Es benötigt eine oder mehrere Software-Schichten, die die Befehle einer höheren Programmiersprache in eine Folge von Maschineninstruktionen transformieren (siehe Bsp. 9.2). Ein Computersystem ist folglich von der Hardware bis zur Software ein geschichtetes Konstrukt, die Speicherhierarchie und das OSI-Netzwerkmodell repräsentieren nur exemplarische Ausschnitte. Es gibt sogar die Möglichkeit, künstliche Hardware-schichten mittels Software zu realisieren. Beispielsweise gibt es virtuelle Prozessoren, die von einer konkreten Hardware-Architektur

und einem konkreten Mikroprozessor abstrahieren. Ein Programm simuliert das Verhalten einer Zentraleinheit und führt eine von der Hardware entkoppelte Maschinensprache aus. Der Vorteil ist Portabilität, d. h., Programme sind auf verschiedensten Hardware-Plattformen ausführbar. Die Programmiersprache Java realisiert so ein Prinzip, nämlich die Java Virtual Machine. So gesehen ist die Informatik wie die Biologie eine Wissenschaft der Ebenen.

Beispiel 9.2: Der Computer als Zwiebel

Wenn wir einen Computer als eine programmierbare Rechenmaschine betrachten, dann eröffnet sich uns ein zwiebelartiger Aufbau (siehe Abb. 9.2): Eine Schicht baut auf der anderen auf und stellt weitere, komplexere Funktionen zur Verfügung. Unten haben wir die Schaltungen, oberhalb den Prozessor, schließlich die Maschinensprache, darüber die Assemblersprache und oben die höhere Programmiersprache, die von der konkreten Hardware abstrahiert. Höhere Programmiersprachen sind natürlich nichts anderes als ein künstliches Konstrukt, sie sind so nicht auf einem Computer ausführbar, genauso wenig

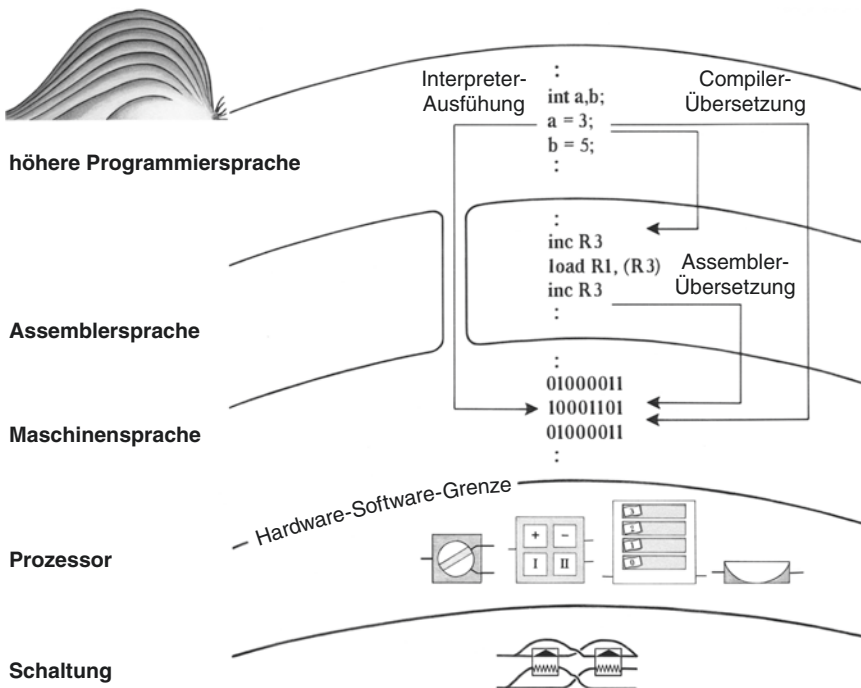


Abb. 9.2 Das Schichtenmodell eines Computers. (© Susanne Staubli, Eckart Zitzler)

wie ein Assemblerprogramm. Es braucht entsprechende Programme, die einen Text, der eine Befehlsfolge in einer höheren Programmiersprache darstellt, liest und auswertet. Analog zu einem Assembler braucht es ein Programm – Compiler genannt –, das den Sprachtext in eine konforme Folge von Maschinensprache-Befehlen übersetzt, die auf dem Zielprozessor ausführbar sind. Alternativ kann dieses Programm auch den Sprachtext interpretieren, d. h. direkt ausführen, was im Sprachtext geschrieben steht; sollen beispielsweise zwei Zahlen miteinander addiert werden, so wird das unmittelbar gemacht und das Ergebnis angezeigt. Solche Programme nennt man auch Interpreter. Interpreter und Compiler stehen so zueinander wie Simultanübersetzer zu Übersetzer: Bei Letzterem entsteht ein Zwischenprodukt, bei Ersterem wird das Gesagte unmittelbar in die Zielsprache umgesetzt. Beide vermitteln zwischen Schichten und ermöglichen das Schaffen neuer Ebenen.

Indes dürfen wir uns das Ganze nicht so vorstellen, dass der Fortschritt immer nur auf der obersten Schicht geschieht. Schichten sind nicht fix, sondern entwickeln sich parallel weiter: Je leistungsfähiger die Elemente auf einer Schicht, desto leistungsfähiger werden auch die darüber liegenden Schichten. Die Schritte der Informationsverarbeitung wurden demnach immer komplexer, vor allem durch die Software konnten immer höhere Abstraktionsniveaus erreicht werden. Allerdings gibt es ein vielfältiges Wechselspiel zwischen Hardware- und Software-Entwicklung, sie stehen ungefähr so zueinander wie die biologische zur kulturellen Evolution. Dass mal die eine und mal die andere den Ton angeben und sich so gesehen der Einfluss der Ebenen wandeln kann, zeigt die Geschichte von IBM, Microsoft und Apple – drei Unternehmen, die die Computerevolution nachhaltig geprägt haben – eindrucksvoll auf (Bsp. 9.3). Und selbst wenn Software prinzipiell viel einfacher zu ändern ist als die Hardware: Komplexe Software-Systeme können schwerfällig sein und Entwicklungen bremsen. Bei großen Programmgeflechten stößt man immer wieder an die Grenzen des Machbaren und Handhabbaren.

Beispiel 9.3: Das Tauziehen zwischen Hardware und Software

Zu Beginn der Computerära war vor allem die Hardware ausschlaggebend, und die Firma, die den Markt damals beherrschte, war IBM, die International Business Machines Corporation. IBM fertigte vor allem Großrechner an, stellte aber 1981 mit dem IBM PC einen Personal Computer vor, der sich an eine breitere Masse richtete. Für diesen Rechner, der aus Standardkomponenten bestand und daher preisgünstig gefertigt werden konnte, benötigte IBM ein Betriebssystem. Der junge Bill Gates (Abb. 9.3 links), der mit Paul Allen 1975 Microsoft gegründet hatte, erwarb kurzerhand von einem versierten Programmierer ein von diesem entwickeltes Betriebssystem, modifizierte es und verkaufte es als Disc

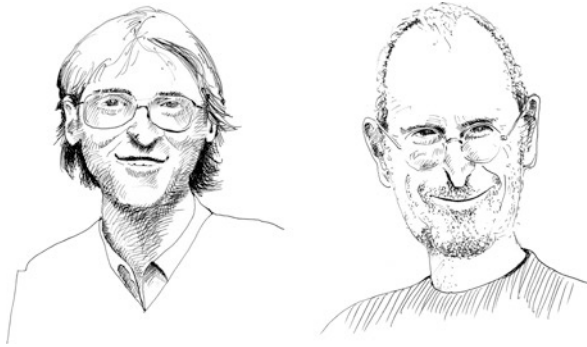


Abb. 9.3 Bill Gates und Steve Jobs, Mitbegründer von Microsoft bzw. Apple. (© Camille Büsser)

Operating System (DOS) an IBM. Der Clou daran: Gates war so pfiffig, das Betriebssystem nicht komplett aus der Hand zu geben, sondern auch anderen Rechnerherstellern zur Verfügung zu stellen und für jede verkaufte Version eine Lizenzgebühr zu verlangen. Damit war der Erfolg von Microsoft geebnet. Der IBM PC verkaufte sich prächtig, diverse Hersteller folgten, bauten das Gerät nach und verwendeten ebenfalls Microsofts Betriebssystem. Somit konnten viele Entwickler Programme schreiben, die auf allen DOS-betriebenen Computern liefen. IBM verlor das Monopol auf die Hardware, während Microsoft im Software-Sektor lange Zeit die Zügel in der Hand hielt: Microsoft Windows, der Nachfolger zu Microsofts DOS, war zu Beginn des 20. Jahrhunderts das mit Abstand am weitesten verbreitete Betriebssystem.

Parallel dazu verläuft die Geschichte von Apple, einem Unternehmen, das 1976 von Steve Jobs (Abb. 9.3 rechts), Steve Wozniak und Ron Wayne gegründet wurde. Mit dem Apple I und dem Apple II war es eines der ersten Unternehmen, die Computer für den Heimbereich anboten, und mit diesen Geräten provozierte Apple letztlich auch die Produktion des IBM PC. Nach anfänglichen Erfolgen mit dem Apple II und dem nachfolgenden Apple Macintosh übernahm Apple eine Vorreiterrolle im Design-Bereich, auch was die grafischen Benutzeroberflächen anbetraf. Doch 1985 verließ Steve Jobs das Unternehmen, und 1996 stand es kurz vor der Übernahme bzw. Zahlungsunfähigkeit. 1997 übernahm Steve Jobs wieder das Ruder, führte das Unternehmen aus der Krise und machte es zehn Jahre später zu einem der führenden in der IT-Branche. Eine entscheidende Hardware-Innovation gelang Apple 2007 mit der Einführung des iPhone, einer Kreuzung aus tragbarem Telefon und tragbarem Computer. Apple gab die Kontrolle über die Software jedoch nicht aus der Hand: Mit dem App Store schuf Steve Jobs einen proprietären Vertriebskanal für Programme, der entscheidend war für den durchschlagenden Erfolg des iPhone.

Der historische Exkurs um die drei IT-Riesen offenbart auch, warum die Gegenüberstellung von natürlicher Evolution und technischer Entwicklung durchaus Parallelen aufweist. Es gibt gesellschaftliche und ökonomische Einflussfaktoren, die sich der Kontrolle der Entwickler entziehen und zu einer eigenen Dynamik führen. Beim Internet hatten wir das bereits diskutiert: Hier ist ein System entstanden, das dezentral wächst, dessen Topologie niemand im Detail kennt und dessen Evolution zu neuen technischen Herausforderungen führt. Ein weiteres Beispiel, das weitreichende technische Konsequenzen hat, aber gesellschaftlich begründet ist, ist das Phänomen der Schad-Software, das wir uns als Nächstes anschauen werden – wieder ausgehend von der biologischen Perspektive.

Gefahren lauern überall

Das Leben ist an sich permanent gefährdet. Ein Organismus ist verschiedensten Umwelteinflüssen ausgesetzt, gegenüber denen er sich behaupten muss – und er ist gefährdet durch andere Lebensformen. Letzteres war schon von Anfang an so, als sich Zellen etablierten, die selbstständig Energie zur Lebenserhaltung produzieren konnten – aus ihnen gingen Pflanzen hervor –, und solchen, die ihre Energieversorgung durch andere Lebewesen sicherstellten – die Urahnen der Tiere. Und unter den Tieren bildeten sich komplexe Abhängigkeiten und Hierarchien von Jägern und Gejagten. Es gibt jedoch auch Organismen und Erreger, bei denen andere Lebewesen nicht nur Energiequellen, sondern Lebensräume sind; Viren beispielsweise betreiben selbst keinen Stoffwechsel und benötigen einen Wirt, um sich reproduzieren zu können. Organismen mussten also Schutzmechanismen entwickeln, um sich gegen äußere Bedrohungen zu wehren. Für jede weitere Komplexitätsstufe kamen weitere Sicherheitsvorrichtungen hinzu – das beginnt bei der Zellwand und geht bis hin zum Immunsystem.

Dass auch Computersysteme gefährdet sein können, trat erst mit der Zeit ins Bewusstsein der Öffentlichkeit, auch wenn es eigentlich auf der Hand liegt. Weil Software eine so große Flexibilität bietet, lässt sich damit auch nahezu alles machen – im positiven wie im negativen Sinne. Sobald ein Programm auf einem Computer ausgeführt wird, kann dieses Dinge tun, die der Benutzer so nicht beabsichtigt. Es können Daten unberechtigt ausgelesen, gesammelt und weitergeleitet werden; oder schlimmer, Daten werden manipuliert, durch Verschlüsselung gesperrt oder gelöscht; oder der Betrieb des Computers kann beeinträchtigt oder blockiert werden. Besonders heikel ist das bei eingebetteten Systemen, wie sie beispielsweise in Industrieanlagen

für Steuerungsaufgaben eingesetzt werden: Wenn ein Programm Amok läuft, können durchaus Schäden an Gerätschaften verursacht werden. Ein Wunder ist es also nicht, dass sich Schadprogramme schreiben lassen, die ein unerwünschtes oder sogar schädliches Verhalten an den Tag legen. Und dass diese auch eingesetzt werden, erstaunt ebenfalls nicht, schließlich ging es bei der automatisierten Informationsverarbeitung von Anfang an immer auch um militärische und kommerzielle Anwendungen. Nein, die Frage ist eher: Warum sollte man Schad-Software überhaupt zur Ausführung bringen, wenn sie das System doch zum Erliegen bringen kann?

Der eine Grund ist offensichtlich: weil jemand tatsächlich Sabotage betreiben will. In diesem Fall muss solchen Personen der Zugriff zu einem Computersystem verwehrt werden. Früher war die Lösung einfach, man konnte einen Rechner einfach in einem Raum verschließen und so von der Umwelt abschotten, doch mit dem Internet bedarf es anderer Ansätze – wir kommen noch darauf zu sprechen. Der andere Grund kann sein: Man weiß gar nicht, dass man ein Schadprogramm ausführt. Mit dem Aufkommen der Personal Computer traten auf einmal Computerviren in Erscheinung, d. h. kleine Programme, die sich unbemerkt auf einem Computer einnisten und dort ihr Unwesen treiben können. Die ersten Viren wurden 1976 entwickelt, als Massenphänomen traten sie jedoch erst ab den 1990er-Jahren in Erscheinung. Die Bezeichnung „Virus“ ist in diesem Fall sehr zutreffend, denn ein Virus-Schadprogramm muss sich erstens einschleusen, zweitens zur Ausführung kommen, drittens sich einnisten und viertens sich verbreiten – wie sein biologisches Pendant. Was im Detail bei einer Virusinfektion abläuft, erläutert Bsp. 9.4.

Beispiel 9.4: Wie Computer krank werden

In der Natur ist ein Virus ein Partikel, der aus einem DNA-Strang und weiteren Molekülen wie Proteinen, die zumeist eine Hülle bilden, besteht. Ein Virus kann sich nicht selbst vervielfältigen – deswegen wird es auch nicht als Lebewesen betrachtet –, sondern benötigt dafür die Protein-Produktionsmaschinerie einer Wirtszelle, die den auf der viralen DNA gespeicherten Reproduktionsablauf in die Realität umsetzt. Der schematische Ablauf einer Virusinfektion ist in Abb. 9.4 rechts dargestellt. Das Virus dockt an die Zellwand an und wird – weil es sich quasi als nützliches Molekül verkleidet – von der Zelle ins Innere aufgenommen. Anschließend wird die Virus-DNA in die DNA der Wirtszelle eingefügt mit der Konsequenz, dass der virale DNA-Teil genauso abgelesen wird wie die zelleigene DNA. Dadurch werden neue Viren erzeugt, bis die Zelle schließlich zerstört wird und die neuen Viren in die Außenwelt treten und weitere Zellen zu infizieren versuchen.

Das gleiche Prinzip finden wir bei den Computerviren wieder (siehe Abb. 9.4 links). Ein Computervirus ist ein Stück Programmcode, das einem

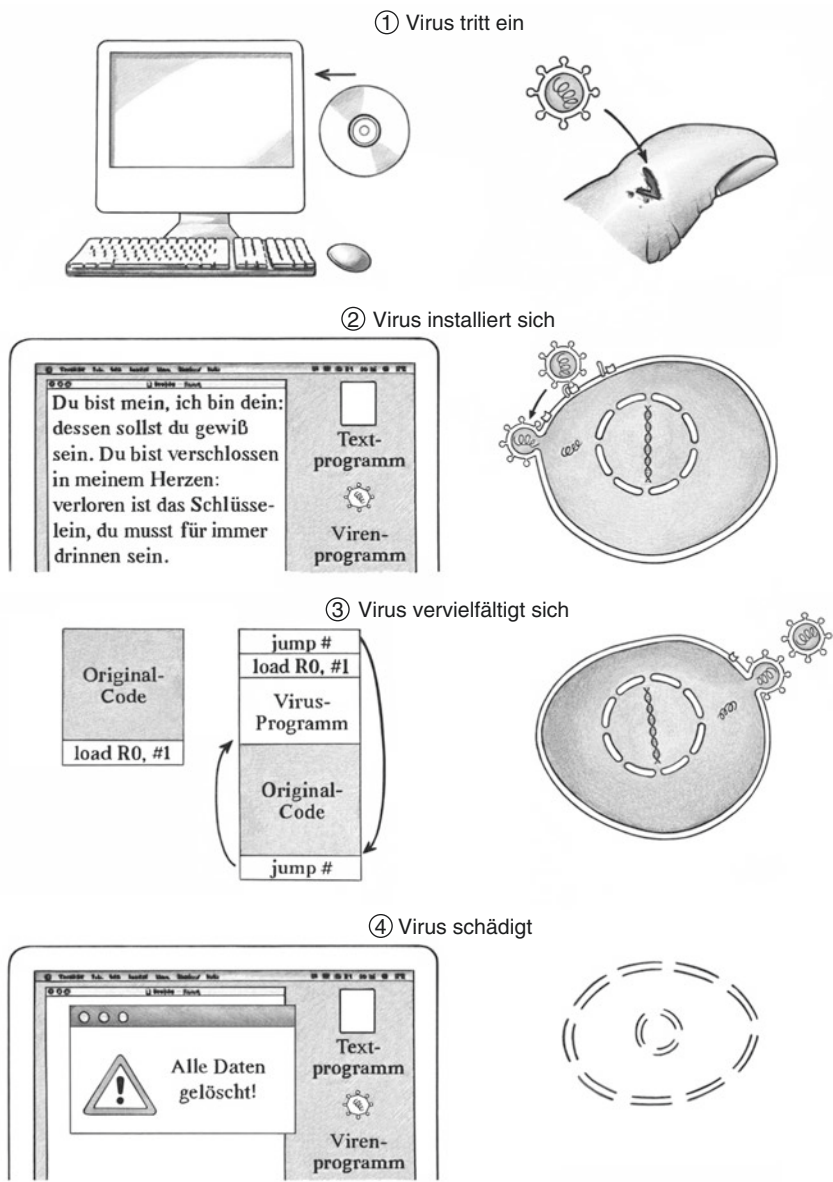


Abb. 9.4 Infektion und Reproduktion bei Computerviren und biologischen Viren. (© Susanne Staubli, Eckart Zitzler)

gewöhnlichen Programm angehängt ist, zusammen mit diesem ausgeführt wird und sich daraufhin selbst reproduziert (interessanterweise stammt diese Idee der Selbstreproduktion von John von Neumann, der auch die Computerarchitektur so nachhaltig geprägt hat). Wie geht das genau? Nun, kommt die Virusroutine zur Ausführung, dann durchsucht sie zunächst einmal den Massenspeicher nach dort abgelegten Programmen. Wird sie fündig, versucht sie den Viruscode an das entsprechende Programm anzuhängen; gleichzeitig manipuliert sie den ursprünglichen Programmcode, sodass als Erstes zur Virusroutine verzweigt wird und erst im Anschluss zum eigentlichen Programm. Des Weiteren kann die Virusroutine versuchen, irgendwelche schadhaften Aktionen durchzuführen, z. B. Dateien zu manipulieren. Wann immer nun ein Programm weitergegeben wird, das einen Virus enthält, breitet sich das Virusprogramm auf dem neuen Rechner aus und kann auch dort Schaden anrichten.

Viren können sich auf einem Computer aktiv vervielfältigen, aber die Verbreitung auf andere Rechner geschieht passiv und muss durch Benutzer erfolgen, die Daten von einem System zum anderen weiterreichen. Mit dem Aufkommen von Computernetzwerken trat eine weitere Form von Schad-Software in Erscheinung, die selbstständig versucht, andere Computer zu infizieren: Die Rede ist von den Würmern. Die ersten Würmer verbreiteten sich per E-Mail. Sie versendeten vom befallenen Computer aus E-Mails an im Adressbuch aufgeführte Empfänger und hängten verseuchten Programmcode an, in der Hoffnung, die Empfänger würden den Anhang öffnen und dadurch die Ausführung des Schadcodes auf ihrem Computer auslösen. Später wurden die Methoden, mit denen die Möglichkeiten des Internets auf destruktive Art und Weise genutzt werden, immer ausgeklügelter. Denn Schad-Software kann sich über ein Netzwerk nicht nur schneller verbreiten, sondern zudem einen Peer-to-Peer-Verbund bilden und somit eine massive Zerstörungskraft entwickeln. Man spricht in diesem Zusammenhang von einem Botnet, wobei dieser Begriff ganz allgemein für eine koordinierte Gemeinschaft von verteilten, autonom agierenden Computerprogrammen (den Bots, abgeleitet von dem englischen Wort *robots*) steht. Zu destruktiven Zwecken kann so ein Botnet beispielsweise eingesetzt werden, um Server und Internetdienste außer Gefecht zu setzen (Bsp. 9.5) oder in großem Stil sensible Daten wie Passwörter und Kreditkartendetails zu stehlen. In der Regel wissen und merken die Besitzer der eingebundenen Computer gar nicht, dass ihr Rechner einen Bot beherbergt und so Teil einer kriminellen Vereinigung ist.

Beispiel 9.5: Erzwungene Dienstverweigerung

Eine Möglichkeit, einen Server und den durch ihn erbrachten Dienst lahmzulegen, besteht darin, ihn mit Anfragen zu überfluten. Einerseits wird dadurch der Server überlastet, andererseits der Durchsatz im Netzwerk reduziert; in letzter Konsequenz könnten dann Anfragen regulärer Nutzer nur noch eingeschränkt oder gar nicht mehr beantwortet werden. Diese Art von Angriff wird *denial of service* genannt: Ein Angreifer versucht, einen Server in die Knie zu zwingen, sodass dieser seinen Dienst verweigern muss. Solange so eine Attacke nur von einem Computer ausgeht, kann sie einfach pariert werden: Zum Beispiel kann die Anzahl von Anfragen pro Nutzer begrenzt oder eine bestimmte Internetadresse gezielt vom Dienst ausgeschlossen werden. Wenn der Angreifer allerdings seine IP-Absenderadresse fälscht und ständig ändert, wird es schwieriger.

Ganz anders sieht die Situation hingegen aus, wenn die Anfragen von sehr vielen, auf der Welt verteilten Computern kommen, und zwar gleichzeitig. Man spricht dann von *distributed denial of service*. Dabei sind die beteiligten Computer meist von Hackern mittels Viren oder Würmer gekapert und in ein riesiges Botnet eingebunden worden. Das Botnet – es kann Hunderttausende bis viele Millionen Bots umfassen – wird über eine Kommandozentrale, einen Command-and-Control-Server, gesteuert (siehe Abb. 9.5). Auf diesem Weg können Teilnetzwerke so massiv mit Daten überflutet werden, dass keine andere Lösung mehr bleibt, als sie temporär vom Internet zu trennen.

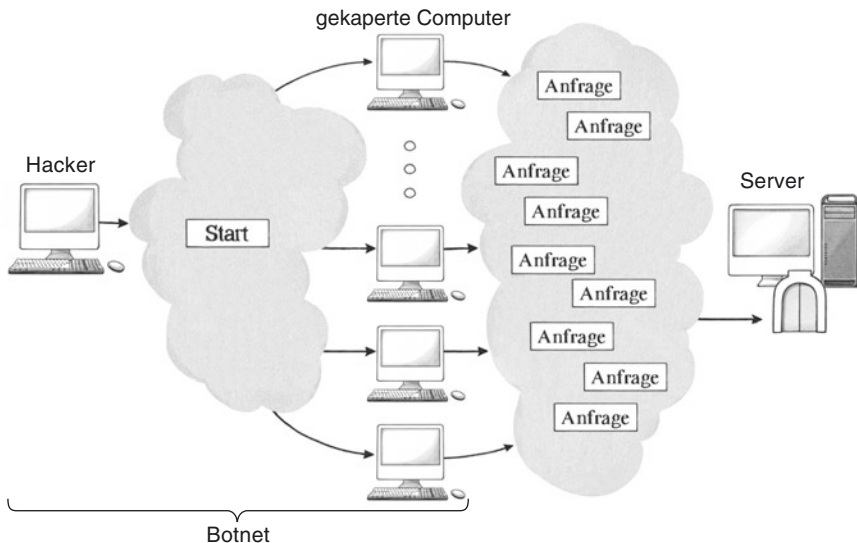


Abb. 9.5 Wie eine Gemeinschaft infizierter Computer einen Server mit Anfragen überhüllt. (© Susanne Staubli, Eckart Zitzler)

Sie sehen, die Möglichkeiten, ein Computersystem zu kompromittieren, sind vielfältig. Neben Viren und Würmern gibt es weitere Arten von Schad-Software, z. B. Trojanische Pferde, bei denen ein nützliches Programm einen bewusst eingebauten schädlichen Teil enthält. Häufig werden Schadprogramme auch nach der Art der Schädigung unterschieden: Spyware zeichnet das Nutzerverhalten auf, um diese Daten unberechtigterweise, z. B. für Werbung, zu verwenden; Ransomware verschlüsselt Dateien auf einem Computer und verlangt für die Entsperrung ein Lösegeld. Und diese Bedrohungen nehmen zu, weil Computer in nahezu allen Umgebungen eingesetzt werden und vom Internet abhängig sind. Aus diesem Grund hat man verschiedenste Sicherheitsmaßnahmen eingeführt, die an die Schutzmechanismen erinnern, die in der Natur vorkommen (siehe Bsp. 9.6). Man versucht so, Unberechtigten den Zugang zu einem Computer und Netzwerk zu verwehren und die Einflussmöglichkeiten von Schad-Software zu begrenzen. Doch im Endeffekt läuft es auf einen permanenten Wettlauf zwischen Angriffs- und Abwehrmechanismen hinaus. Es ist zu erwarten, dass mit der Zeit das Immunsystem eines Computers immer komplexer wird – das menschliche Immunsystem macht es vor. Tatsächlich hat man schon mit Imitationen des menschlichen Immunsystems gearbeitet, um fremde Datenpakete in einem Netzwerk zu erkennen und auszusortieren.

Beispiel 9.6: Wie Computersysteme geschützt werden

Ebenso wie der Mensch verfügen Computersysteme über Schutzhüllen, die sie von der Außenwelt abgrenzen; Interaktionen mit der Umwelt erfolgen nur über kontrollierte Zugänge. Abb. 9.6 gibt Ihnen einen Überblick über die wichtigsten Maßnahmen.

Einer der ältesten Schutzmechanismen ist die Benutzerkontrolle. Personen müssen sich über ein Passwort oder andere individuelle Zugangsdaten ausweisen und können den Computer erst bei korrekter Identifikation nutzen. Wie bei der Haut oder einer Zellwand wird nur ausdrücklich Bekanntes hindurchgelassen. Zudem können die Rechte von Benutzern, was sie auf dem jeweiligen Computer anstellen dürfen, beschränkt werden. Nur einem eingegrenzten Personenkreis – den Administratoren, die für den Unterhalt des Computersystems verantwortlich sind – stehen alle Rechte zu.

Auch auf der Netzwerkebene gibt es solche Zugangskontrollen. Sie heißen Firewalls und bilden quasi einen Feuerschutzwall um ein lokales Netzwerk oder einen Computer. Firewalls – sie können beispielsweise auf Routern oder Endgeräten installiert sein – kontrollieren den Datenverkehr und lassen nur zulässige Pakete hindurch. Auf diese Weise können lokale Netzwerke geschützt und sensible Ressourcen wie z. B. ein Dateiserver von der Umwelt abgeschottet werden. Damit ein Zugriff von außen trotzdem möglich ist, wurde das Konzept des virtuellen Tunnels eingeführt. Dabei wird ein externer Computer über eine verschlüsselte Verbindung zu einem Router in ein lokales Netzwerk eingebunden und kann so auf geschützte Ressourcen wie einen Dateiserver zugreifen.

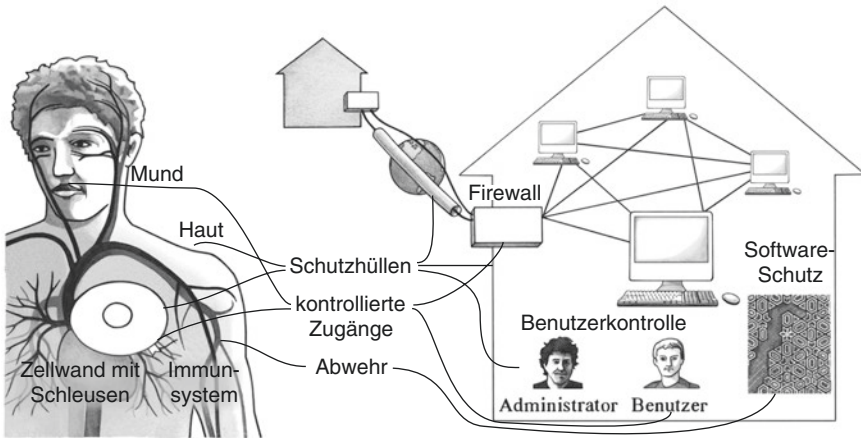


Abb. 9.6 Sicherheitsmaßnahmen beim menschlichen Organismus und bei Computersystemen. (© Susanne Staubli, Eckart Zitzler)

Und schließlich gibt es noch diverse Software-Systeme, die darauf ausgerichtet sind, den Zustand eines Computersystems zu überwachen und Anomalitäten zu erkennen. Sie komplementieren die Schutzmaßnahmen und kommen dann zum Tragen, wenn Schad-Software das System erreicht hat. Sie stellen wie das Immunsystem eine Art Früherkennungs- und Reparaturmechanismus zur Verfügung.

Doch jeder Zugangsschutz hat natürlicherweise Lücken und kann untergraben werden. Das größte Sicherheitsrisiko stellt dabei der Mensch dar. Wenn mit Zugangsdaten nachlässig umgegangen wird, ungeprüft Programme gestartet oder auf gefälschten Webseiten sicherheitsrelevante Informationen preisgegeben werden, kann ein Computersystem leicht gekapert werden. Ein anderes menschlich verursachtes Problem sind Programmierfehler, die Löcher in der Schutzhülle hervorrufen. Besonders gravierend sind Lücken, die bislang nur Hacker entdeckt haben und die so gesehen geheime Hintertüren darstellen. Ein Beispiel, dessen Schwere ein bekannter Sicherheitsforscher damals als katastrophal bezeichnete, ist der Heartbleed-Bug (Anm. 9.1); hier konnte u. U. ein kryptografisch gesicherter Datenverkehr entschlüsselt werden. Und selbst wenn die Lücken bekannt sind und bereinigte Software-Versionen vorliegen, können sich Schadensfälle enormen Ausmaßes ereignen. Im Jahr 2017 setzte eine breit angelegte Ransomware-Attacke die IT-Infrastruktur in zahlreichen englischen Krankenhäusern außer Betrieb, mit weitreichenden Konsequenzen für die

Patienten. Die Schad-Software nutzte eine bereits bekannte Sicherheitslücke im Betriebssystem Windows aus; weil sehr viele Computersysteme noch nicht aktualisiert waren, konnte sich die Ransomware massiv ausbreiten und viele Tausend Rechner infizieren.

Anmerkung 9.1: Wenn das virtuelle Herz blutet

Als Heartbleed-Bug wurde ein 2014 entdeckter und über zwei Jahre unbemerkter Fehler in einer Open-Source-Programmbibliothek bezeichnet, die Routinen für die verschlüsselte Kommunikation über das Internet zur Verfügung stellte. Dadurch entstand eine Sicherheitslücke, die Angreifern unter Umständen ermöglichte, private Schlüssel auszulesen und so die kryptografischen Sicherheitsmaßnahmen auszuhebeln. Diese Bibliothek wurde vielfach eingesetzt, um das HTTPS-Protokoll (vgl. Kap. 6) zu implementieren, und betraf somit u. a. den Zahlungsverkehr über das Internet.

Der Fehler bestand in einer Komponente, die eine sogenannte Heartbeat-Funktionalität realisierte: Sie erlaubt es zwei Kommunikationspartnern, regelmäßig zu überprüfen, ob die Verbindung intakt ist und beide Seiten betriebsbereit sind. Dazu schickt ein Partner einen Datenblock an die Gegenseite mit der Bitte, ihn so zurückzuschicken. Allerdings kann er angeben, mehr Daten zu schicken, als er effektiv überträgt. Aufgrund des Programmierfehlers wurde so ein Fall auf der Gegenseite nicht bemerkt. Sie sendete die gewünschte Datenmenge zurück, aber – weil sie ja weniger Daten erhalten hatte – schickte sie dabei unbemerkt Daten aus dem eigenen Hauptspeicher mit. Darin konnten sich dann sensible Daten wie private Schlüssel oder Passwörter befinden.

Letztlich geht es aber auch um Vertrauen. Woher wissen Sie, dass Ihr neu erworbenes Software-Produkt tatsächlich tut und nur das tut, was es soll? Sie setzen auf vertrauensvolle Entwickler, Anbieter und Vertriebskanäle. Transparenz kann helfen, das Vertrauen zu stärken, doch auch Open-Source-Software ist keine Garantie dafür, dass Programmierfehler entdeckt werden. Die absolute Sicherheit gibt es nicht, genauso wenig wie im wirklichen Leben. Dennoch sollte man versuchen, Fehler beim Programmieren so gut wie möglich zu vermeiden. Das leitet direkt über zum nächsten Thema: Wie baut man Software-Systeme, sodass diese wirklich tun, was sie sollen?

Vom Entwickeln

Es liegt wohl in der Natur der Sache, dass man immer mehr möchte. Entsprechend nehmen die Anforderungen an Software-Systeme ständig zu, die Anwendungsszenarien werden immer reichhaltiger und komplexer. Damit umfangreiche Programmpakete überhaupt entwickelt werden kön-

nen und handhabbar bleiben, bedarf es eines systematischen Vorgehens und geeigneter Entwurfsmethoden. Denn Software-Fehler sind nicht nur gefährlich im Hinblick auf potenzielle Saboteure. Im Fall von eingebetteten Computersystemen können ganze Anlagen zerstört werden, wie das Beispiel der Ariane 5 zeigt, einer europäischen Trägerrakete, mit der u. a. Satelliten in den Weltraum befördert werden. Bei ihrem Erstflug 1996 stellte sich die Rakete eine halbe Minute nach dem Start quer und sprengte sich anschließend selbst; der Grund waren diverse Konstruktionsfehler bei der Steuerungs-Software. Man spricht in diesem Zusammenhang von Bugs (siehe Anm. 9.2). Dieses Beispiel verdeutlicht, wie wichtig es ist, Software-Systeme so zu bauen, dass sie korrekt funktionieren und das gewünschte Verhalten zeigen.

Anmerkung 9.2: Der Mythos vom Bug

Ein Bug bezeichnet einen Fehler in einem Software-System, kann aber auch generell einen Konstruktionsfehler in einem Computersystem titulieren. Wie es zu dem Begriff kam, ist nicht genau belegt, aber es gibt einen Mythos. Dieser besagt, dass am 9. September 1947 der Harvard-Mark-II-Computer in den USA nicht richtig funktionierte, weil sich eine Motte in einem der Relais verklemmt hatte. Das lästige Insekt – dafür steht der englische Ausdruck *bug* im weitesten Sinne – wurde ins Logbuch eingeklebt, und der entsprechende Vermerk lautete: „First actual case of bug being found.“ Es handelt sich hierbei um eine Geschichte von Grace Murray Hopper (Abb. 9.7), einer Programmiererin der ersten Stunde, die nicht nur den ersten Compiler entwickelte, sondern auch maßgeblich an der Entwicklung der Programmiersprache COBOL (*Common Business Oriented Language*) beteiligt war. Auch wenn diese Geschichte ausreichend belegt ist und als wahr angesehen werden kann, so wurde das Wort *bug* nicht erst seit diesem Ereignis verwendet; schon Edison, der Erfinder der Glühbirne, benutzte es, um einen Defekt zu beschreiben.



Abb. 9.7 Grace Murray Hopper, von der die Geschichte zum ersten realen Computerbug stammt. (© Camille Büsser)

Im Folgenden schnuppern wir hinein in die Methoden der Software-Entwicklung. Einmal geht es um das Programmieren: wie Abläufe mittels abstrakter Kunstsprachen formuliert werden können und welche Leitbilder dabei zugrunde liegen. Und es geht um den Entwicklungsprozess, von der Formulierung der Aufgabenstellung über den Entwurf einer Programmarchitektur bis hin zum getesteten Software-Produkt.

Handwerk der Programmierung

Die Maschinensprache stellt zwar das Fundament für die Programmierung eines Computers dar, ist aber für die Erstellung großer Software-Produkte ungeeignet. Viel zu kleinschrittig ist sie, selbst wenn moderne Prozessoren umfangreiche Instruktionssätze zur Verfügung stellen. Die Entwicklungszeit ist zu lang, der Programmumfang zu groß und damit auch die Fehleranfälligkeit zu hoch. Deswegen fing man schon bald an, mächtigere Befehlssätze für Computer zu schaffen, und zwar nicht auf der Hardware-, sondern auf der Software-Ebene. Man dachte sich die erwähnten höheren Programmiersprachen aus, mit denen Abläufe einfacher, verständlicher und übersichtlicher beschrieben und komplexere Probleme gelöst werden konnten. Weil man dabei von konkreten Prozessoren abstrahierte, ließen sich die codierten Abläufe auf verschiedenen Rechnersystemen nutzen. Und man entwickelte dazugehörige Programme – Interpreter und Compiler –, die die Sprachen erkennen und in Maschinencode-Sequenzen umsetzen konnten.

Höhere Programmiersprachen sind letztlich den drei Prinzipien Modularisierung, Kapselung und Hierarchisierung verpflichtet. Zunächst einmal geht es um eine weitere Stufe der Abstraktion: Sie bieten umfassende Möglichkeiten, den Kontrollfluss und den Datenfluss zu definieren. Zum einen besitzen sie leistungsfähigere Elementaroperationen, z. B. können wiederholte Ausführungen von Programmsegmenten mit verschiedenen Schleifenkonstrukten ausgedrückt werden, ohne umständliche Sprungbefehle. Zum anderen besitzen sie Datentypen. Ein Datentyp spezifiziert die Art und Weise, wie Daten im Speicher abgelegt werden, welche Codierung dafür gewählt wird und welche Grundoperationen darauf erlaubt sind. Beispielsweise gibt es Datentypen für Zeichenketten – sie werden auch Strings genannt und bestehen aus Folgen von Unicode-Symbolen –, natürliche Zahlen, ganze Zahlen und Fließkommazahlen. Darüber hinaus bieten höhere Programmiersprachen die Möglichkeit, Unterprogramme und Module von Unterprogrammen zu definieren. Somit wird Programmcode sowohl strukturiert als auch wiederverwendbar, gleichzeitig lassen sich so

Details kapseln. Bsp. 9.7 veranschaulicht diese Ideen anhand einer konkreten Programmiersprache.

Beispiel 9.7: Die Programmiersprache C

C ist eine Programmiersprache, die in den 1970er-Jahren entstand, um die Weiterentwicklung des Betriebssystems UNIX zu vereinfachen; sie wurde von Dennis Ritchie entwickelt. C ist eine schlanke Sprache, die verhältnismäßig nah zur Assemblersprache steht. Anweisungen bestehen entweder aus Rechenoperationen, aus Kontrollflussbefehlen oder aus Unterprogrammaufrufen. Rechenoperationen sind Grundrechenarten und weitere Assembler-nahe Befehle, mit denen beispielsweise eine Zahl um eins erhöht oder die Ziffern einer Dualzahl um eine Position verschoben werden können. Befehle zur Steuerung des Kontrollflusses umfassen Schleifenkonstrukte und Konstrukte für die alternative Ausführung im Stil von Wenn-dann-Abfragen. Darüber hinaus ermöglicht C, Anweisungen mittels geschweifeter Klammern zu gruppieren und Unterprogramme zu definieren. Umfangreiche Programmbibliotheken, mit den sich Operationen auf Zeichenketten und viele weitere Funktionen einfach durchführen lassen, ergänzen das basale Sprachgerüst.

Für die Manipulation von Daten gibt es in C Variablen. Eine Variable ist nichts anderes als ein Platzhalter für eine Zelle oder einen Bereich im Hauptspeicher, in dem der Wert dieser Variable festgehalten wird. Der Datentyp legt fest, wie der Wert konkret abgespeichert wird und benutzt werden kann; dafür muss eine Variable vor ihrer erstmaligen Benutzung im Programm deklariert werden, d. h., ihr Datentyp muss spezifiziert werden. C kennt verschiedene Datentypen: Zum Beispiel werden ganze Zahlen als `int` (für Englisch *integer*) bezeichnet, Fließkommazahlen als `float`, und Schriftzeichen sind vom Datentyp `char` (für Englisch *character*). C erlaubt es sogar, direkt mit Zeigern zu arbeiten, womit sich beliebige Datenstrukturen konstruieren lassen.

In Abb. 9.8 sehen Sie, wie die Berechnung des größten gemeinsamen Teilers als C-Programm formuliert werden kann. Die erste Zeile des Programmcodes definiert das Unterprogramm `ggT` und sagt aus, dass es zwei ganzzahlige Werte in den Variablen `a` und `b` entgegennimmt und einen ganzzahligen Wert zurückliefert. Die `while`-Anweisung definiert eine Schleife und führt die eingerückte `if-else`-Kombination aus, solange `a` und `b` verschiedene Werte beinhalten. Der `if-else`-Anweisungsblock zieht entweder `a` von `b` ab oder umgekehrt, je nachdem, welcher Wert größer ist. Die `return`-Anweisung liefert den berechneten Wert für den größten gemeinsamen Teiler zurück.

Nun gibt es aber nicht einfach die höhere Programmiersprache, sondern eine Menge an unterschiedlichen Sprachen. Ein Grund dafür ist die Verschiedenartigkeit der Anwendungsszenarien. Weil C verhältnismäßig nah an der Assemblersprache ist, hat es sich für die Entwicklung von Betriebssystemen bewährt. Andere Sprachen sind auf mathematische Anwendungen ausgerichtet und stellen mächtige Rechenoperationen zur

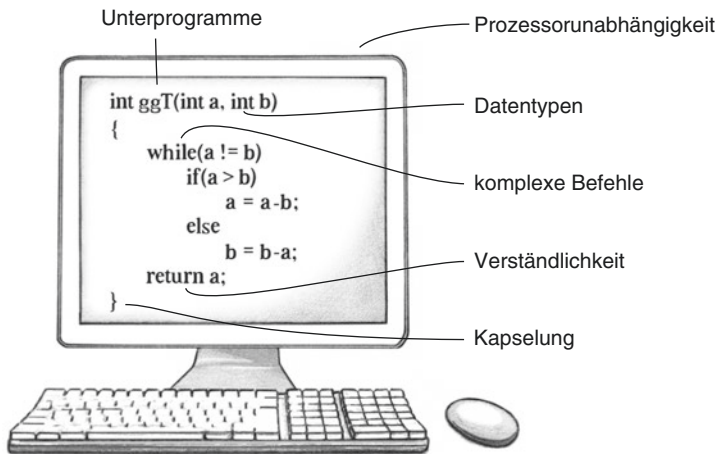


Abb. 9.8 Neuerungen von höheren Programmiersprachen gegenüber Maschinensprachen am Beispiel von C. (© Susanne Staubli, Eckart Zitzler)

Verfügung. Viele Anwendungsprogramme wie Textverarbeitungs- und Tabellenkalkulationspakete stellen Makrosprachen zur Verfügung; über Makros lassen sich automatische Anpassungen vornehmen, wenn beispielsweise Formulare auszufüllen sind und die Inhalte der Felder voneinander abhängen. Dann gibt es Sprachen, die in Webbrowsern eingebunden sind und auf deren Basis Webseiten mit den Benutzern interagieren können; auch serverseitig gibt es Sprachen, um Webseiten dynamisch aufbauen zu können, z. B. auf der Basis einer Datenbank. Skriptsprachen sind eher auf kleine Programme ausgerichtet und dienen vor allem dazu, aus bestehenden Einzelprogrammen spezifische Anwendungsprogramme zu konstruieren. Und es gibt spezielle Sprachen, um Datenbankabfragen und -manipulationen zu formulieren. Ein weiterer Grund für die Sprachenvielfalt ist der stetige Fortschritt. Mit der Zeit haben sich Denkschemata und ganze Philosophien entwickelt, wie man programmieren sollte. Sie werden als Programmierparadigmen bezeichnet (siehe Bsp. 9.8), und mit jedem neuen Paradigma entstehen neue Sprachen.

Beispiel 9.8: Leitbilder für das Programmieren

Am Anfang stand das Paradigma der imperativen Programmierung, das dem Grundsatz folgt: Ein Programm ist eine Abfolge von Befehlen, die der Computer ausführt. Dieses Paradigma ergibt sich unmittelbar aus der Art und Weise, wie Prozessoren arbeiten und wie Maschinensprachen aufgebaut sind. Viele höhere Programmiersprachen bauen auf diesem Paradigma auf, unterscheiden

sich jedoch von Assemblersprachen dadurch, dass sie komplexere Befehle und Datentypen anbieten.

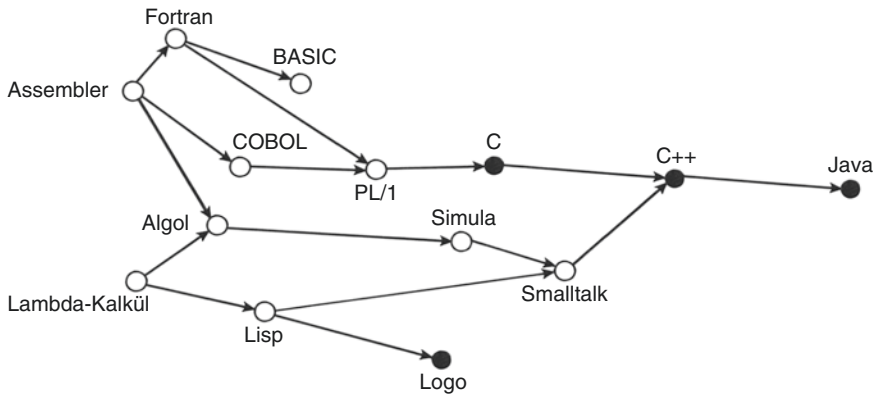
Später entstand daraus die strukturierte Programmierung, bei der es darum geht, Programme in verständliche, handhabbare und funktional geschlossene Einheiten zu zerlegen und somit übersichtlich und wartbar zu halten. Insbesondere das Konzept des abgekapselten Unterprogramms ist hier wichtig, genauso wie die Beschränkung auf drei elementare Arten des Programmablaufs, über die der Kontrollfluss beeinflusst werden kann: erstens die Hintereinanderausführung, zweitens die Verzweigung und drittens die Wiederholung. Man versuchte den berüchtigten Spaghetticode zu vermeiden, indem unübersichtliche Sprungbefehle – vergleichbar zum jump-Befehl auf Maschinensprache-Ebene – gar nicht erst erlaubt wurden.

Die strukturierte Programmierung wurde weiterentwickelt zum objektorientierten Programmierparadigma. Dieses propagiert die Zusammenführung und Kapselung von Daten sowie den darauf angewendeten Algorithmen in sogenannten Objekten. Objekte bieten die Möglichkeit, komplexe Datentypen zu definieren, und können über das Konzept der Vererbung leicht erweitert und wiederverwendet werden.

Schließlich gibt es noch ein gänzlich anderes Programmierparadigma, und zwar das der deklarativen Programmierung. Hier wird ein Ansatz verfolgt, der auf den ersten Blick irritiert, weil er diametral zum imperativen Paradigma steht: Die Programmiererin sagt nicht mehr, **wie** genau etwas berechnet werden soll, sondern sie deklariert, d. h. beschreibt die gesuchte Lösung. Diese Beschreibung wird dann interpretiert und ein allgemeines Suchverfahren, das zumeist auf Rekursion und Rückverfolgung basiert, versucht dann eine passende Lösung zu finden.

Mit der Zeit haben sich verschiedenste Programmiersprachen herausgebildet, darunter gibt es sogar visuelle, bei denen Programme grafisch dargestellt werden. Ihre Namen klingen abenteuerlich: von Fortran und COBOL über C, C++ (sprich: C plus plus), Pascal, Modula, Java zu JavaScript, Perl, Python, Scratch, SQL und Swift. Und wie bei den natürlichen Sprachen haben sich neue Sprachen aus alten gebildet, Ideen wurden – auf der Basis neuer Programmierparadigmen – weiterentwickelt und kombiniert. C beispielsweise ist sowohl imperativ als auch strukturiert, jedoch nicht objektorientiert; aus C entwickelte sich dann C++, das die Objektorientierung, die mit Small talk eingeführt wurde, aufnimmt. Abb. 9.9 zeigt die Anfänge der Programmiersprachen auf, ein vollständiger Stammbaum wäre nicht einmal ansatzweise auf einer Seite darstellbar. All diese Sprachen zielen darauf ab, die Formulierung von Informationsverarbeitungsvorgängen zu vereinfachen, sind aber immer auch ein Kompromiss zwischen der Präzision, die die Ausführung auf einem Computer verlangt, und dem Abstraktionsgrad, den die Beschreibung komplexer Vorgänge voraussetzt.

Programmiersprachen



natürliche Sprachen

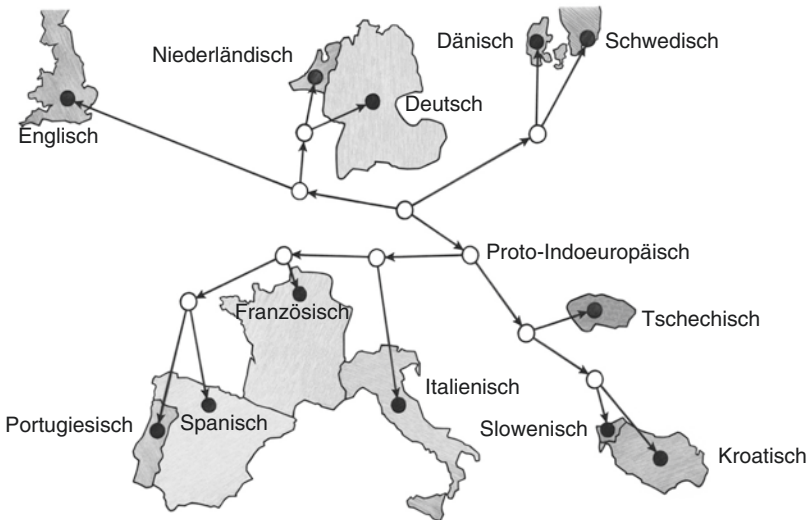


Abb. 9.9 Ein Stammbaum ausgewählter Programmiersprachen (oben) und europäischer Sprachen (unten). (© Susanne Staubli, Eckart Zitzler)

Spannend ist, dass nicht immer technische Aspekte den Ausschlag geben, welche Sprachen sich durchsetzen. Es gibt einige wohldurchdachte und sauber konzipierte Programmiersprachen aus dem akademischen Umfeld, die für die Ausbildung eingesetzt wurden, sich aber in der Praxis nicht etablieren konnten. Die Evolution der Sprachen besitzt offenbar ihre eigene Dynamik. Lohnenswert ist in diesem Zusammenhang ein Blick auf die natürlichen Sprachen.

Jede Sprache besitzt ihren Wortschatz und ihr Regelwerk, die wir – insofern wir sie beherrschen wollen – verinnerlichen müssen. Doch die Wörter und Regeln sind nicht starr, sie stellen vielmehr ein lebendiges Gefüge dar. Welche Mechanismen dem ständigen Veränderungsprozess unterliegen, untersucht die Linguistik, und natürlich gibt es verschiedene Theorien dazu. Falls es Sie interessiert: Anm. 9.3 erläutert Ihnen einen Erklärungsversuch, den ich persönlich interessant finde und deshalb hier aufgenommen habe, den Sie aber auch getrost überspringen können. Wie auch immer: Weil sich Sprachen verändern und weil sich zudem menschliche Gemeinschaften aufspalteten und isolierten, entstand eine Evolution der Sprachen. Wie in Abb. 9.9 exemplarisch zu sehen ist, war es in Europa das Proto-Indoeuropäische, aus dem sich viele heutige Sprachen entwickelt haben. Die Quintessenz: Sobald Menschen mit Sprachen arbeiten, erarbeiten sie sich ihre Freiheiten.

Anmerkung 9.3: Die Entwicklung natürlicher Sprachen

In seinem Buch über die Entwicklung der Sprache unternimmt der Linguist Guy Deutscher eine faszinierende Reise durch die Entstehungsgeschichte der Sprache und beschreibt eindrucksvoll, wie sich so komplizierte Gebilde wie die deutsche Sprache mit ihren vielseitigen Ausdrucksformen und Regelsystemen gebildet haben – ohne dass sich je ein Mensch hingesetzt und dieses Regelwerk bewusst konzipiert hätte. Deutscher argumentiert, dass es im Wesentlichen zwei Arten von Vorgängen gibt, die eine Sprache formen: die Erosion und die Innovation. Erosion steht für den Umstand, dass sich Wörter, Regeln und Redewendungen abnutzen, sie quasi im täglichen Sprachgebrauch abgeschliffen werden. Sie umfasst aber auch die Tendenz, sich ausdrucksvoll mitzuteilen: Ein „verboten“ reicht häufig nicht, wir müssen unterstreichen, dass es wirklich verboten ist, und entsprechend liest man auf Verbotsschildern dann „streng verboten“ oder „strengstens verboten“. Innovation bezeichnet die Tendenz, die Sprache ökonomisch zu verwenden mit dem Resultat, dass Lästiges und Anstrengendes kurzerhand verschwindet. Innovation umfasst auch den Drang, Regelmäßigkeiten in den Dingen zu sehen und Analogien zu verwenden; so werden neue Wörter und Wortzusammensetzungen in neuer Bedeutung kreiert, neue Regeln geschaffen – ohne dass eine Regierung dies per Dekret angewiesen hätte. Erosion und Innovation zusammen sind laut Deutscher die Triebfedern der sprachlichen Entwicklung. So entstehen zum Teil obskure Sprachschöpfungen, wie das nachfolgende, aus Deutschers Buch entnommene Beispiel zeigt.

Sowohl im Deutschen als auch im Französischen gehen die Begriffe für „heute“ auf einen lateinischen Ausdruck zurück, nämlich *hoc die* (an diesem Tag), der später zu *hodie* erodierte (Abb. 9.10). Im Altfranzösischen wurde daraus das Wörtchen *hui*, welches ebenfalls ein Resultat der Erosion ist. Weil dieser Ausdruck aber vermutlich als zu schwach empfunden wurde, wenn man ausdrücken wollte, dass etwas wirklich heute passiert, stellte man kurzerhand die Floskel *au jour* voran, was zur Redewendung *au jour d'hui* führte (an dem

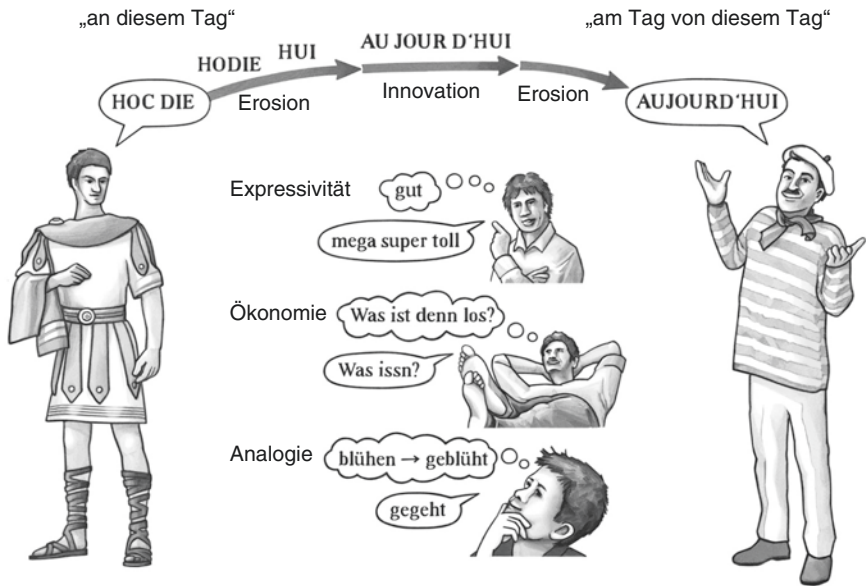


Abb. 9.10 Illustration der der Sprachentwicklung unterliegenden Mechanismen anhand des Beispiels „heute“ im Französischen – das Beispiel entstammt Deutscher (2008). (© Susanne Staubli, Eckart Zitzler)

Tag von diesem Tag). Daraus wurde durch Erosion dann der bekannte Begriff *aujourd'hui*, dem man im Alltagsgebrauch mittlerweile schon wieder ein *au jour* voranstellt: *au jour aujourd'hui* (an dem Tag an diesem Tag von diesem Tag). In nicht allzu ferner Zukunft wird man vielleicht von *ajourajui* oder etwas Ähnlichem sprechen.

Bezogen auf die Programmierung bedeutet das: Eine höhere Programmiersprache ist kein Garant für fehlerfreie Software. Genauso wie wir in unserer Alltagssprache einen Sachverhalt unterschiedlich ausdrücken können, einfach und kompliziert, umständlich und elegant, direkt und verblümt, so lassen sich Abläufe in einer Programmiersprache auf ganz unterschiedliche Arten ausdrücken. Es erfordert viel Disziplin, wartbare, robuste, übersichtliche und verständliche Programme zu schreiben – gerade unter Zeitdruck, der immer da ist, und bei großen Teams. Zwar haben höhere Programmiersprachen die Entwicklung von Software enorm beschleunigt, doch will man komplexe Systeme bauen, braucht es darüber hinaus eine Methodik für den Entwicklungsprozess. Denn niemand setzt sich an den Computer und beginnt einfach so zu programmieren. Es braucht einen Plan.

Wie komplexe Software-Systeme gebaut werden

Die Software-Technik ist die Disziplin, die sich mit der systematischen Entwicklung von Software auseinandersetzt. Sie entstand in einer Zeit, die als Software-Krise bekannt ist. Ende der 1960er stieß man mit dem klassischen Ansatz der Programmentwicklung an die Grenze. Bislang hatte man es nur mit relativ überschaubaren Programmen und Anwendungen zu tun gehabt, zumeist ging es um die Automatisierung wissenschaftlicher Berechnungen. Doch weil die Hardware immer leistungsfähiger und erschwinglicher wurde, gerieten auch die Software-Pakete größer und größer. Die Komplexität wuchs den Programmierern über den Kopf, und so brauchte es Wege aus der Krise. Als Resultat etablierten sich verschiedene Vorgehensweisen und Aufgabenfelder rund um die Software-Entwicklung. Dabei kommen jene Prinzipien zur Anwendung, die wir schon kennengelernt haben: die Zerlegung eines Systems in mehrere gekapselte Module, die wiederum aus Subsystemen bestehen. Das Ganze lässt sich durchaus mit dem Bau eines Gebäudes vergleichen. Kein Architekt legt einfach so los, es werden zuerst Erkundigungen eingeholt, Pläne angelegt, Modelle erstellt usw., bevor dann mit dem Bagger der Platz für das Fundament ausgehoben wird. Bei der Software ist das ähnlich.

Grundsätzlich lassen sich bei der Software-Entwicklung verschiedene Phasen unterscheiden, wie es in Abb. 9.11 skizziert ist. Diese Phasen stehen für verschiedene Arbeitsschritte, müssen jedoch nicht zwingend in der angegebenen Reihenfolge ausgeführt werden. Zu Beginn sollte man wissen, was das System leisten soll: Das Systemverhalten und die Anforderungen sind zu spezifizieren. Gerade wenn Kunden ein Software-Produkt bestellen, ist es enorm wichtig, mit ihnen zusammen festzuhalten und zu definieren, was das Programm können und tun soll. Anschließend versucht die Informatikerin eine Architektur des zu bauenden Systems zu entwerfen. Dabei geht es

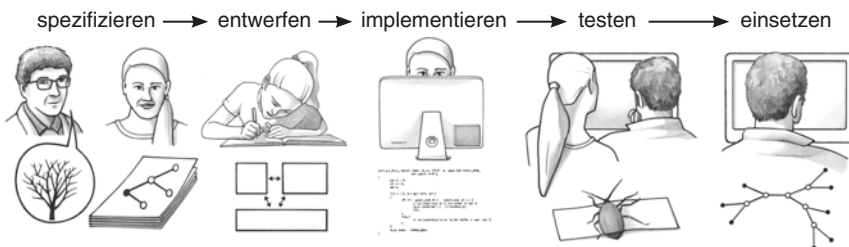


Abb. 9.11 Allgemeines Vorgehen bei der Realisierung eines Software-Systems.
(© Susanne Staubli, Eckart Zitzler)

darum, dem System eine innere Struktur zu geben und das Zusammenspiel der Komponenten festzulegen. Auf der Basis der Systemarchitektur werden die resultierenden Module programmiert, man sagt auch implementiert. Häufig sind die Komponenten selbst zu groß, als dass sie einfach so realisiert werden könnten – nötigenfalls sind weitere Strukturierungsschritte erforderlich, in denen die Architektur immer feiner untergliedert wird. Steht dann schließlich das Programm für die geplante Software-Anwendung, muss es noch ausführlich getestet werden, und auch hier braucht es ein systematisches Vorgehen. Der Aufwand, der hierbei betrieben wird, kann je nach Anwendungsfeld stark variieren: Bei einer Textverarbeitung sind Fehler ärgerlich, bei einer Flugzeugsteuerung fatal. Erst wenn die Testphase abgeschlossen ist, kommt das gesamte System zum Einsatz, und dabei kann es ebenfalls passieren bzw. ist die Regel, dass es noch Nachbesserungen bedarf und Updates notwendig sind. Ja, ein Software-System muss gepflegt werden.

Lassen Sie uns noch einen genaueren Blick auf den Entwurf der Software-Architektur werfen, ein zentraler Aspekt bei der Software-Entwicklung. Die Überlegungen, die dabei angestellt werden müssen, möchte ich anhand eines Beispiels verdeutlichen, nämlich der Architektur eines Take-away-Restaurants (siehe Abb. 9.12). Zunächst einmal würden die zukünftigen Betreiber sagen, wie sie sich ihr Restaurant vorstellen, und auf dieser Basis entwirft der Architekt einen Plan des Gebäudes: vorne eine Theke, die innen von einem großen Raum bedient wird; dieser Raum grenzt an eine Küche und von beiden Räumen aus ist ein Vorratsraum zugänglich. Die Aufteilung in funktional zweckgebundene Räume soll den Abläufen dienen, die in so

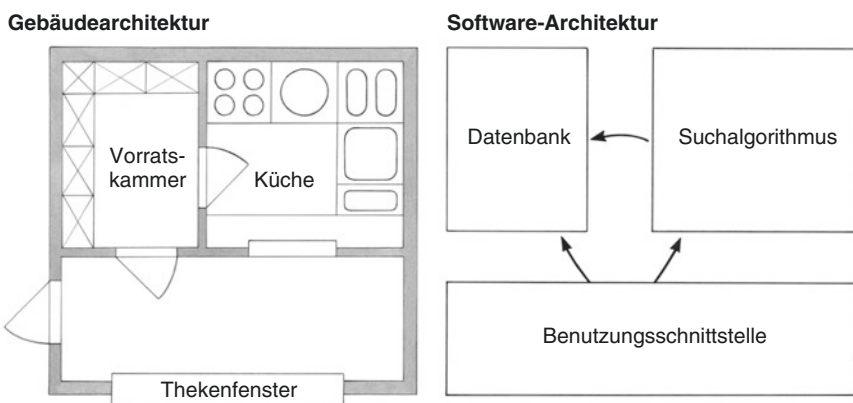


Abb. 9.12 Eine Gebäudearchitektur als Analogie zu einer Software-Architektur. (© Susanne Staubli, Eckart Zitzler)

einem Restaurant für den Betrieb notwendig sind. Die Vorratskammer muss kühl und dunkel sein, die Küche über die erforderlichen Gerätschaften verfügen, während der Raum zum Bedienen Platz bieten muss, damit verschiedene Mitarbeitende rasch die Kundenanfragen entgegennehmen und bearbeiten können. Ähnliche Betrachtungen spielen bei dem Entwurf einer Software-Architektur eine Rolle. Man versucht, funktionale Einheiten zu identifizieren und so die Aufgabenbereiche aufzuteilen. Eine Strategie kann sein, ein Modul für die Datenspeicherung – es umfasst Datenstrukturen und Zugriffsoperationen auf diesen –, ein Modul für die Datenverarbeitung – hier finden die eigentlichen Berechnungen statt – und ein Modul für die Benutzungsschnittstelle – also die Kommunikation mit dem Anwender – vorzusehen: Diese Variante ist in Abb. 9.12 zu sehen. Reale Software-Systeme haben natürlich wesentlich mehr Module, die fein verästelt in viele weitere Module aufgegliedert sind.

Ganz zentral für den nachfolgenden Schritt der Implementierung ist, dass für jedes Modul genau spezifiziert ist, was es tut und wie es mit den anderen Modulen kommuniziert. Dafür werden Schnittstellen beschrieben, häufig in Form von Unterprogramm-Deklarationen, die dieses Modul zur Verfügung stellt und die von den anderen Modulen verwendet werden können; Bsp. 9.9 gibt Ihnen einen konkreten Einblick. Es gibt keine Vorschrift, wie man die Aufteilung vorzunehmen hat, das Vorgehen speist sich aus Erfahrung und aus verschiedenen Zielen: Verdaubarkeit der Portionen, Bearbeitbarkeit durch Teams, Vermeidung von Fehlern, Wiederverwendbarkeit usw. Natürlich gibt es Kriterien, die eine gute Architektur ausmachen, doch diese sind nicht fix. Der Entwurf einer guten Architektur ist eine Kunst, genauso wie Architektur selbst eine Kunst ist. Teil davon ist Handwerk, doch andere Teile entziehen sich einem klar festgelegten Schema und werden zum Teil auch durch Moden bestimmt. Während in der Architektur mal die Ästhetik, mal die Kosten, mal die Ökologie im Vordergrund steht, sind es bei der Software die Kosten, die Erweiterbarkeit, die Entwicklungszeit usw.

Beispiel 9.9: Eine Architektur und ihre Schnittstellen

Nehmen wir an, wir wollten ein Schachspielprogramm schreiben, mit dem der Benutzer gegen den Computer spielen kann. Unsere Schachanwendung könnte eben aus jenen in Abb. 9.12 gezeigten drei Modulen bestehen: einem Suchalgorithmus, der den jeweils nächsten Zug für den Computer ermittelt, einer Datenbank, in der Spielzüge aus anderen Spielen wie in einer Bibliothek zusammen mit ihren Erfolgsaussichten gespeichert sind, und einer

Benutzungsschnittstelle, die ein Schachbrett auf den Bildschirm projiziert und die Interaktionen mit dem Benutzer, z. B. das Ziehen einer Spielfigur, abwickelt.

Die Schnittstelle des Suchalgorithmus könnte so aussehen, dass es genau ein Unterprogramm gibt, das zu einem Spielstand einen Zug ermittelt; sein Name könnte `berechneNaechstenZug` sein. Und die Datenbank könnte eine Routine `ermittleChancenBeiSpielstand` zur Verfügung stellen, die einen ganzzahligen Wert zurückgibt, der die Erfolgsaussichten für einen Spielstand und eine bestimmte Farbe widerspiegelt; diese Routine kann sowohl vom Suchalgorithmus als auch vom Benutzer genutzt werden. Das Modul Benutzungsschnittstelle wäre schließlich für die Steuerung zuständig und würde das Hauptprogramm beinhalten, welches auf die anderen beiden Module zugreift und sie aufruft. Das Hauptprogramm wird – in C – durch eine Funktion `main` angegeben; so weiß der Compiler, an welcher Stelle das Programm zu starten ist. Insgesamt sähen die Schnittstellen der drei Module dann wie folgt aus (das Wort `void` heißt, dass kein Wert übermittelt wird):

```
void    berechneNaechstenZug(spielfeld    aktuellerSpielstand,
                             spielfeld neuerSpielstand, int farbe);
int    ermittleChancenBeiSpielstand(spielfeld    aktuellerSpiel-
                                     stand, int farbe);
void main(void);
```

Die Reihenfolge, in der ich die Schritte der Software-Entwicklung präsentiert habe, spiegelt ein idealtypisches Szenario wider, das in der Realität selten anzutreffen ist. So weiß ein Kunde am Anfang nicht unbedingt, was er genau will; die Anforderungen ändern sich im Laufe der Diskussionen und vor allem dann, wenn das System im Einsatz ist. Nur bei sehr klar umrissenen Aufgaben kommt das sogenannte Wasserfallmodell zum Tragen, bei dem die Abfolge Spezifikation, Entwurf, Implementierung und Testen im Wesentlichen so eingehalten wird. Deswegen haben sich mit der Zeit andere Prozessmuster herausgebildet, z. B. das Spiralmodell – alle Schritte werden immer wieder verfeinert durchlaufen – und die agile Software-Entwicklung. Bei diesen Herangehensweisen versucht man möglichst früh einen Software-Prototyp zu erstellen, anhand dessen dann die Anforderungen verfeinert werden können. Auch hier haben wir es wieder mit Paradigmen zu tun, die alle ihre Vor- und Nachteile haben und je nach Situation besser oder schlechter geeignet sind.

Letztlich sind es auch philosophische oder geschmacklich-subjektive Aspekte, die hier den Ausschlag geben. Und es ist immer ein Wechselspiel zwischen technologischen Entwicklungen und Anschauungen, die zu bestimmten Strömungen führen. Sehr gut lässt sich das in der Baukunst beobachten, wo Architekturstile stark durch den Zeitgeist geprägt sind (siehe Anm. 9.4). Manifeste markieren dabei Umbrüche im Denken wie

z. B. das Bauhaus-Manifest von 1919, das bescheiden erklärt: „Das Endziel aller bildnerischen Tätigkeit ist der Bau!“ Interessanterweise verfassen auch Informatiker Manifeste – ebenso wie die Künstler. Beispielsweise unterschrieben 2001 verschiedene Software-Entwickler ein Agiles Manifest, das die Werte beschreibt, die der agilen Software-Entwicklung zugrunde liegen. Darin wird beispielsweise gesagt, dass Menschen und Interaktionen wichtiger sind als Prozesse und Werkzeuge oder dass die Zusammenarbeit mit dem Kunden wichtiger ist als die ursprünglich, evtl. vertraglich ausgearbeitete Leistungsvereinbarung. Wenn das nicht philosophisch ist! Doch auch wenn das Vorgehen selbst variieren kann und sich unterschiedliche Stile herausgebildet haben – die Grundmethodik der Software-Technik ist stets dieselbe.

Anmerkung 9.4: Zwischen Technologie und Zeitgeist

Bei der Architektur von Tempeln und Kirchen ging es lange Zeit darum, religiöse Vorstellungen zu vermitteln. Als man im 12. Jahrhundert entdeckte, wie hohe Gewölbe mittels Stützen gebaut werden können, entstand eine neue Architektur, die Gotik. Sie beruhte auf verschiedenen technischen Errungenschaften, doch die zentrale Erfindung im Gegensatz zum vorangegangenen romanischen Stil war: Kirchen konnten nach dem gleichen Prinzip wie Glashäuser gebaut werden, mit einem entsprechenden Steingerippe, das im Wesentlichen die Pfeiler braucht, die Kreuzrippen und Stützen, aber ohne massive Wände dazwischen auskommt. Es setzte eine Befreiung von der reinen Funktion ein, gleichzeitig gab es eine gesellschaftliche Entwicklung, dass sich die Städte emanzipierten und ihre eigenen Kirchen bauten. All das prägte schließlich den Stil der Gotik (siehe Abb. 9.13). Heutzutage sind wiederum ganz andere Bauformen möglich und gefragt.



Abb. 9.13 Die gotischen Türme des Basler Münsters (links) und das Vitra Design Museum in Weil am Rhein (rechts). (© Judith Büsser)

Es bleibt anzumerken, dass die diskutierten Bauprinzipien zwar denen in der Natur ähneln, aber es einen entscheidenden Unterschied gibt: In der Biologie sind Systeme nicht entworfen, um sie nachher gut warten zu können; sie haben sich auf dem Aufbauenden entwickelt, sie sind das Resultat der Umweltbedingungen und der historischen Möglichkeiten. In der Informatik gibt es andere Kriterien: Die Systeme müssen beherrschbar, verständlich und erweiterbar sein. Das hat Konsequenzen für den Bau eines Systems, d. h., man versucht, Blinddärme zu vermeiden. Dass das nicht immer so möglich ist, hat mit der Systemkomplexität zu tun. Software-Systeme im Dienstleistungs- oder Industriesektor können riesige, über Jahre gewachsene Gebilde sein, die nicht einfach von Grund auf neu gebaut werden können; hier ist Evolution eigentlich schon Realität.

Vom Entdecken

Kommen wir zum Abschluss auf zwei konkrete Anwendungsszenarien zu sprechen, die im Zusammenhang mit der biologischen Evolution stehen. Das eine Mal geht es um die Rekonstruktion der Evolution auf der Basis genomischer Daten, das andere Mal um die Nachahmung der Evolution, um komplexe Konstruktionsprobleme zu lösen. Die beiden Szenarien dienen dazu, die ganze Spanne von der Aufgabenstellung über die Problemlösung bis zum Einsatz eines Software-Systems zu illustrieren. Dabei haben wir es nicht mit einzelnen isolierten Problemen zu tun, wie wir sie in Kap. 8 betrachtet haben, sondern mit mehreren miteinander verschränkten Problemen, die sich aus der Fragestellung ergeben.

Evolution verstehen

Woher kommen wir? Diese Frage treibt uns seit jeher um, und verallgemeinert steht sie auch im Zentrum der Evolutionsbiologie, die sich damit beschäftigt, wie Arten entstehen und wie sich das Leben entwickelt hat. Charles Darwin war hat diesbezüglich einen entscheidenden Schritt getan, um Licht in das Dunkel zu bringen. Er hat die biologische Evolution als selbst organisierten Prozess entdeckt, der von zwei Phänomenen angetrieben wird: Selektion und Variation. Auch wenn wir dadurch eine Idee bekommen, wie sich Organismen angepasst haben – es ist sehr schwer vorstellbar, wie sich das Leben in dieser großen Zeitspanne entwickelt hat. Um den vergangenen Entwicklungsprozessen auf die Spur zu kommen, kön-

nen wir Lebensformen anhand verschiedenster Merkmale vergleichen, von der Anatomie über die individuelle Entwicklung bis zum Sozialverhalten (Abb. 9.14). In der Biologie wird seit jeher versucht, eine Systematik in all die Beobachtungen und Vermutungen zu bringen. So entstanden Stammbäume, die die Zusammenhänge, wer von wem abstammt und wie sich das Leben entwickelt hat, grafisch darstellen. Die in der DNA gespeicherten Erbinformationen repräsentieren eine weitere Möglichkeit, auf deren Basis so eine Systematisierung vorgenommen werden kann. Ihre Verwendung brachte einen enormen Fortschritt mit sich, denn sie erlaubt präzisere Rückschlüsse auf die zeitliche Reihenfolge, in der sich Spezies abgespalten haben und eigenständig geworden sind.

Das Genom – die Gesamtheit der in einer Zelle vorhandenen DNA-Moleküle – verändert sich von Individuum zu Individuum. Mittels Mutationen und bei manchen Arten auch Rekombinationen wird der Bauplan für die Organismen einer Art über unzählige Generationen hinweg an die Umweltbedingungen angepasst. Bei Rekombinationen werden Erbmaterialien verschiedener Individuen miteinander verwoben, so entstehen neue Arrangements, jedoch keine neuen Abschnitte auf der DNA. Die eigentlichen Neuerungen bringen die Mutationen, und diese Veränderungen haben nur Bestand, wenn sie sich für den Organismus als nützlich erweisen und er sich erfolgreich fortpflanzen kann. Mutationen treten nicht beliebig häufig auf, und insbesondere Punktmutationen, bei denen nur ein Nucleotid verändert wird, lassen sich zeitlich recht gut abschätzen. Jetzt ist die Idee: Wir vergleichen die DNA von zwei Organismen und können so bestimmen, wie viele Punktmutationen notwendig wären, um das eine Genom aus dem anderen zu erzeugen (oder umgekehrt). Aus dieser Information kann man dann Rückschlüsse darauf ziehen, in welchem



Abb. 9.14 Lebensformen können über äußere und innere Merkmale miteinander in Beziehung gesetzt werden. (© pixelprof/Getty Images/iStock (links), GoodLifeStudio/Getty Images/iStock (rechts))

Zeitraum sich die Spezies aufgespalten und getrennt habt. Dieser Ansatz lässt sich noch erweitern, sodass mehrere Organismen in Zusammenhang gesetzt werden können.

Das Ganze setzt allerdings voraus, dass der Text auf der DNA bekannt ist, und diese Textfassung ist gar nicht so einfach zu bewerkstelligen. Zwar gibt es Sequenziermaschinen, die DNA-Moleküle biochemisch auslesen können, aber sie verarbeiten nur Stränge begrenzter Länge – wir sprechen hier von einigen Hundert Basenpaaren. Jetzt stellt sich die Frage, wie man DNA-Stränge mit vielen Millionen Basenpaaren sequenzieren kann, und da haben wir bereits das erste Informatikproblem. Beim Menschen war es dank massiven Computereinsatzes um die Jahrtausendwende so weit. Im Jahr 2000 verkündete der damalige Präsident der USA, Bill Clinton, zusammen mit zwei Wissenschaftlern, dass das menschliche Genom bald entschlüsselt sei. Es sollte zwar noch dauern, bis die Resultate veröffentlicht (2001) und die letzten Abschlussarbeiten durchgeführt waren (2003), doch selbstverständlich markiert dieses Ereignis einen Durchbruch in der Genetik: Das vollständige menschliche Genom mit über 3 Mrd. Basenpaaren lag als A-C-G-T-Buchstabenfolge vor. Welche Rolle Software-Systeme bei der Sequenzierung von DNA-Strängen spielen, erläutert Ihnen Bsp. 9.10.

Beispiel 9.10: Wie ein Genom ausgelesen wird

Die Idee ist, lange DNA-Stränge in kleine Stücke zu zerschneiden, die einzeln mittels Sequenziermaschinen ausgelesen werden können, und anschließend die Textbausteine im Computer wieder zusammenzusetzen, so wie bei einem Puzzle. Plastisch können Sie sich das so vorstellen, dass ein Foto in extrem kleine Fetzen zerrissen wird und es nun darum geht, aus den Fetzen das Originalfoto zu rekonstruieren (siehe Abb. 9.15). Beim Foto ist es etwas einfacher als bei der DNA, weil man noch die Form und die Anschlussstellen zur Verfügung hat. Und wenn für das Foto sogar mehrere identische Abzüge vorliegen und diese entsprechend zerrissen worden sind, dann erhöht sich die Wahrscheinlichkeit, eine Version des Originalfotos zusammensetzen zu können; der Grund ist, dass wir durch die Überlappungen von Fetzen Zusatzinformationen ausnutzen können.

Bei einem DNA-Strang ist die Umsetzung dieser Idee nicht trivial, so einfach sie auch klingen mag. Damit das Zusammensetzen funktioniert, müsste man beim Zerschneiden die Reihenfolge der Teilstränge erhalten. Das ist bei dieser molekularen Schnur nicht möglich, man kann sie auch nicht gezielt mit einer Schere an beliebigen Stellen zerschneiden. Was hingegen funktioniert, ist den DNA-Strang an zufälligen Stellen zu trennen, z. B. durch mechanische Einflüsse wie Ultraschall; allerdings weiß man nachher nicht, wie die Teilstücke zueinander angeordnet waren. Damit man die Reihenfolge trotzdem ermitteln kann, bedient man sich eines Tricks: Man vervielfältigt die DNA, bevor man sie zerschneidet, und führt dann die Zerschneidung und Sequenzierung für jede der Kopien separat durch. Weil das Zerschneiden nach dem Zufallsprinzip funktioniert, lassen sich aus den

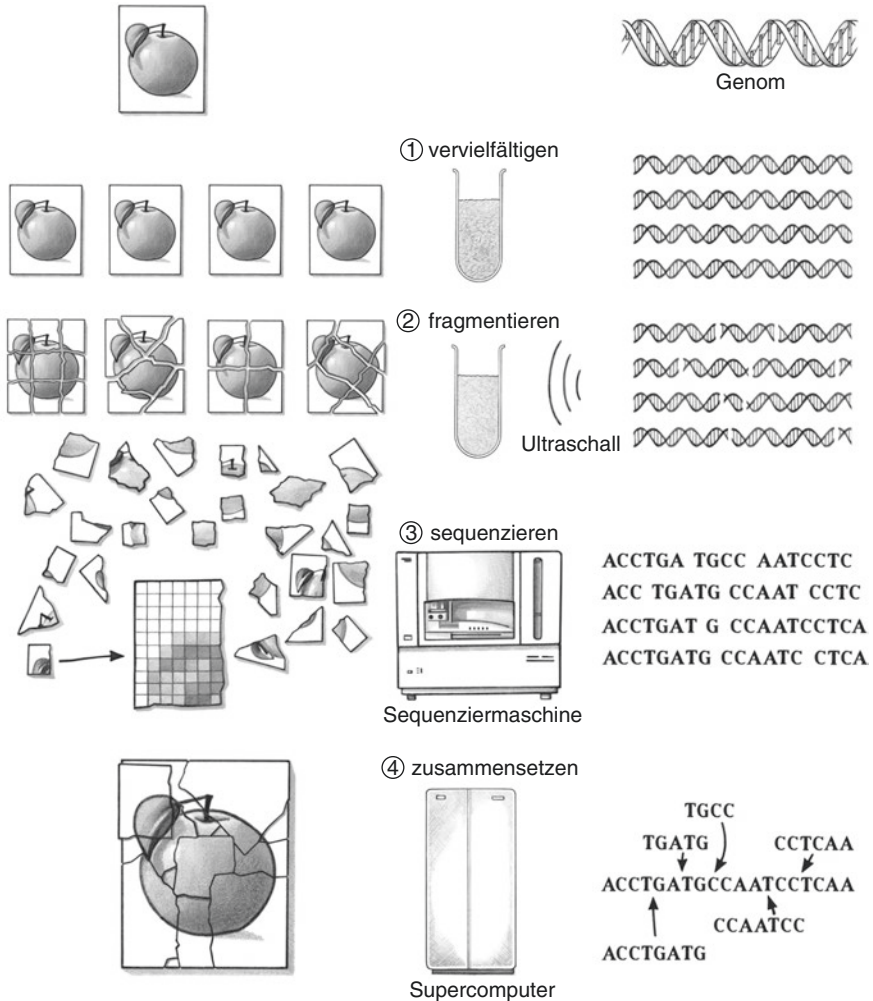


Abb. 9.15 Der Ablauf bei der Shotgun-Sequenzierung – der Rekonstruktion eines Fotos gegenübergestellt. (© Susanne Staubli, Eckart Zitzler)

Teilstücken aller DNA-Kopien überlappende „Wege“ konstruieren, vom Anfang bis zum Ende des gesamten DNA-Strangs. Dieses Prinzip nennt man Shotgun-Sequenzierung.

In Abb. 9.15 sind die einzelnen Schritte der Shotgun-Sequenzierung grafisch dargestellt. Ein längerer DNA-Strang wird zunächst vervielfältigt, und zwar mit ähnlichen Methoden, wie wir sie bereits in Kap. 1 im Zusammenhang mit dem DNA Computing kennengelernt haben. Anschließend wird jede einzelne Kopie in zufällig gewählte Teilstränge zerschnitten. Jeder dieser Teilstränge wird nun mittels der traditionellen aufwendigen Art der Sequenzierung ausgelesen.

Danach liegen viele Abschriften von Teilstücken der DNA vor. Nun kommt der Computer mit einem ausgeklügelten Software-System zum Einsatz: Es gilt, ein riesiges Puzzle zu lösen und eine möglichst lange Sequenz von sich möglichst stark überlappenden Teilstücken zu finden; diese Sequenz repräsentiert dann die gewünschte Information. Natürlich können hier Fehler passieren, aber letztlich kann das Verfahren die effektive A-C-G-T-Folge recht genau ermitteln.

Liegen die DNA-Sequenzen von mehreren Organismen vor, so kann man sich an die Rekonstruktion eines Stammbaums – im Fachjargon spricht man von einem phylogenetischen Baum – machen. Man konzentriert sich auf ausgewählte DNA-Fragmente, z. B. einzelne Gene, geht von einem gemeinsamen Urahn aus und fragt danach, wann sich die einzelnen Arten abgespalten haben. Dabei gilt es mehrere Teilfragen zu beantworten. Wir müssen ermitteln, welche Teile der DNA-Fragmente konserviert sind und von einem gemeinsamen Vorfahren stammen und welche durch Mutationen entstanden sind. Wir müssen die DNA-Fragmente in Beziehung zueinander setzen und in einer Stammbaumstruktur abbilden. Und schließlich müssen wir noch die zeitlichen Abstände abschätzen, also wann sich wer von wem abgespalten hat. Bei allen drei Fragen können wir nur Vermutungen anstellen, die auf bisherigem Wissen und modellhaften Annahmen beruhen.

Aus Sicht der Informatik ergibt sich damit ein hochgradig komplexes Optimierungsproblem. Die Eingabe besteht aus den ausgesuchten DNA-Abschnitten der verschiedenen Organismen. Die Ausgabe ist ein Stammbaum, der als Graph repräsentiert werden kann. Die Knoten des Graphen sind mit einzelnen Arten assoziiert, die Kanten stehen für Vorfahren-Beziehungen und sind mit Zeitspannen angeschrieben. Natürlich darf dieser Graph keine Kreise enthalten – keine Art kann Urahn von sich selbst sein – und muss zusammenhängend sein – keine Art ist vom Himmel gefallen. Gesucht ist der Graph, der die zeitliche Entwicklung mit der größten Wahrscheinlichkeit wiedergibt. Genau diesen Graphen zu ermitteln, ist unmöglich, zu komplex ist das Ganze. Doch es lässt sich ein möglichst wahrscheinlicher Stammbaum finden, wenn man das Rekonstruktionsproblem in handhabbare Einzelprobleme zerlegt. Jedes dieser kleineren Probleme ist für sich genommen noch schwierig genug, die Entscheidungsvarianten sind NP-vollständig – da wissen wir, welche Herausforderungen uns hier erwarten. Das erste Problem besteht darin, die DNA-Abschnitte aufeinander auszurichten; denn sie sind in der Regel verschieden lang, und wie berechnet man da die Unterschiedlichkeit bzw. eine Distanz zwischen DNA-Abschnitten? Das zweite Problem dreht sich um die Abstammungsreihenfolge; es ist ein Graph zu rekonstruieren, der die Abspaltungen aufzeigt, aber noch keine Zeitangaben aufweist. Und das dritte Problem heißt, Abschätzungen

vorzunehmen und zu berechnen, was zuerst war; der gesamte Graph ist in einen Stammbaum mit einer zeitlichen Anordnung zu überführen. Bsp. 9.11 zeigt auf, wie dieses Problem mittels einer Programmkollektion gelöst wird.

Beispiel 9.11: Wie Stammbäume rekonstruiert werden

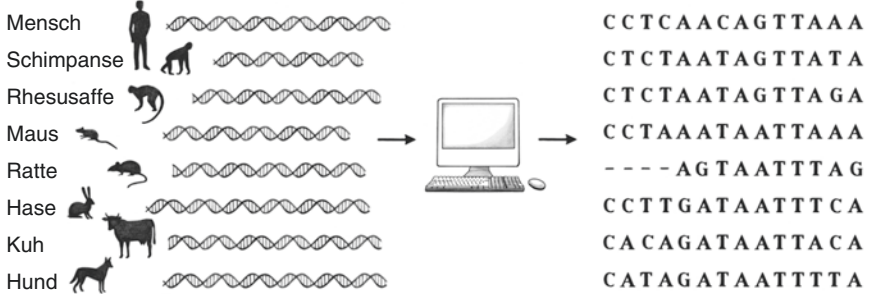
Das erste Problem wird als Sequenzalignment bezeichnet und hat zum Ziel, die vorgegebenen DNA-Abschnitte auf die gleiche Länge zu bringen. Im einfachsten Fall gilt es, zwei DNA-Sequenzen so zueinander auszurichten, dass eine möglichst große Übereinstimmung entsteht und die Paarung eine hohe Wahrscheinlichkeit besitzt, durch Evolution entstanden zu sein. Dazu dürfen Lücken, mit einem Strich gekennzeichnet, in die Sequenzen eingeführt werden, die entweder für den Verlust oder das Hinzukommen eines Nucleotids stehen. Auf diese Weise lässt sich die Übereinstimmung erhöhen. Müssen mehrere DNA-Sequenzen zueinander ausgerichtet werden, wie das in Abb. 9.16 der Fall ist, spricht man von einem multiplen Sequenzalignment – und gleich ist das Problem noch etwas komplizierter.

Das zweite Problem besteht darin, einen Graphen zu konstruieren, der die Abstammungsbeziehungen aufzeigt. Mittels einer gierigen Strategie lässt sich das wie folgt erreichen. Wir gehen von einem Graphen aus, bei dem alle Arten von genau einem Urahn abstammen; dieser fiktive Urahn steht in der Mitte (vgl. Abb. 9.16). Anschließend werden schrittweise neue fiktive Zwischenvorfahren eingeführt. Die zwei DNA-Sequenzen, die am nächsten zueinander stehen, d. h. zwischen denen am wenigsten Punktmutationen stehen, werden unter einem neuen gemeinsamen Vorfahren zusammengefasst; der Vorfahr bekommt die DNA-Sequenz, die quasi zwischen denen der beiden Organismen steht. Das Prozedere wird wiederholt, bis schließlich die Wurzel nur noch drei direkte Kinder hat.

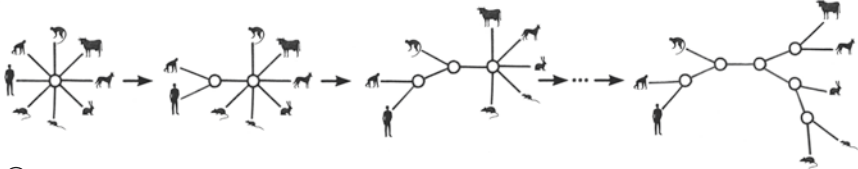
Zum Schluss geht es beim dritten Problem darum, die zeitliche Reihenfolge zu ermitteln. Zunächst muss man bestimmen, wer von all den aufgeführten Organismen eigentlich der älteste ist; der fiktive Vorfahr, von dem ich oben sprach, ist ja nur eine Konvention ohne biologische Bedeutung. Wenn man hingegen aufgrund von Zusatzinformationen weiß, welcher Organismus am ältesten ist, lässt sich eine Wurzel des Baums bestimmen und der entsprechende Stammbaum aufschreiben. Schließlich folgt noch der letzte Schritt, nämlich aufgrund der Punktmutationen abzuschätzen, wie viel Zeit jeweils verging, bis sich ein Nachfahre aus seinem Vorfahren entwickelt hatte. Das lässt sich z. B. mittels der Längen der Kanten grafisch darstellen, wie in Abb. 9.16 unten gezeigt ist.

Wie verlässlich ist nun der rekonstruierte Stammbaum? Zuerst einmal müssen wir uns vor Augen halten, dass im Software-System, das den Stammbaum ermittelt, verschiedene biologische Annahmen eingeflossen sind. Entsprechend handelt es sich beim berechneten Graphen um eine Hypothese, genauso wie Verhaltensforscher Hypothesen aufgrund einer Menge von Beobachtungen anstellen. Des Weiteren müssen wir uns bewusst sein, dass hier keine exakten Algorithmen, sondern Heuristiken zum Einsatz kommen. Insbesondere bei einer gierigen Strategie erhalten wir bekanntlich

① multiples Sequenzalignment



② Baumkonstruktion



③ Zeitabschätzung

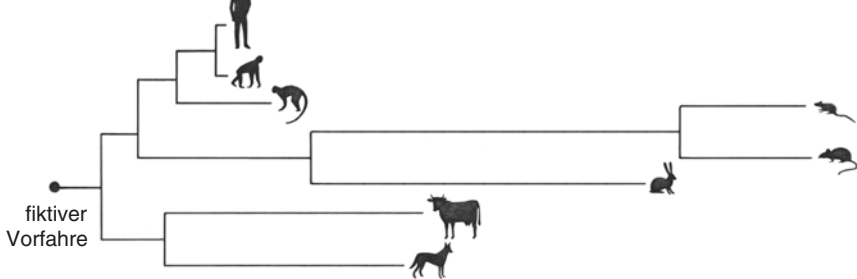


Abb. 9.16 Drei Schritte in einer computerbasierten Stammbaumrekonstruktion: 1. Ausrichtung der DNA-Abschnitte, 2. Baumkonstruktion und 3. Zeitabschätzung. (© Susanne Staubli, Eckart Zitzler)

nur eine Näherungslösung. Das ist generell bei der Verwendung von computerbasierten Auswertungstools zu beachten, auch wenn sich die Güte einer Näherungslösung mittels statistischer Methoden abschätzen lässt. Insofern muss die vom Computer generierte Hypothese weiter validiert und mittels anderer Beobachtungen ergänzt und verfeinert werden.

Dieses Beispiel zeigt auf, wie sich ein neues Gebiet an der Schnittstelle zwischen Informatik und Biologie etabliert hat: die Bioinformatik. Sie beschäftigt sich damit, biologische Fragestellungen mithilfe informatischer Methoden zu lösen. Das heißt aber nicht, dass da einfach nur auf der Basis einer Standard-Software Daten auszuwerten wären. Häufig ist ein auf eine biologische Fragestellung speziell zugeschnittenes Programmsystem zu entwickeln. Daraus ergeben sich vielfach rechenintensive Berechnungsprobleme,

die auch die Informatik vor große Herausforderungen stellen. Insofern braucht es auf dem Gebiet der Bioinformatik sowohl tiefgehendes Wissen in der Biologie als auch in der Informatik. Häufig arbeiten Expertinnen und Experten aus beiden Gebieten interdisziplinär zusammen. Ein Biologe und eine Informatikerin müssen also ihr Know-how zusammenbringen, wollen sie solch eine Abstammungsfrage beantworten (siehe Abb. 9.17). Sie entwerfen

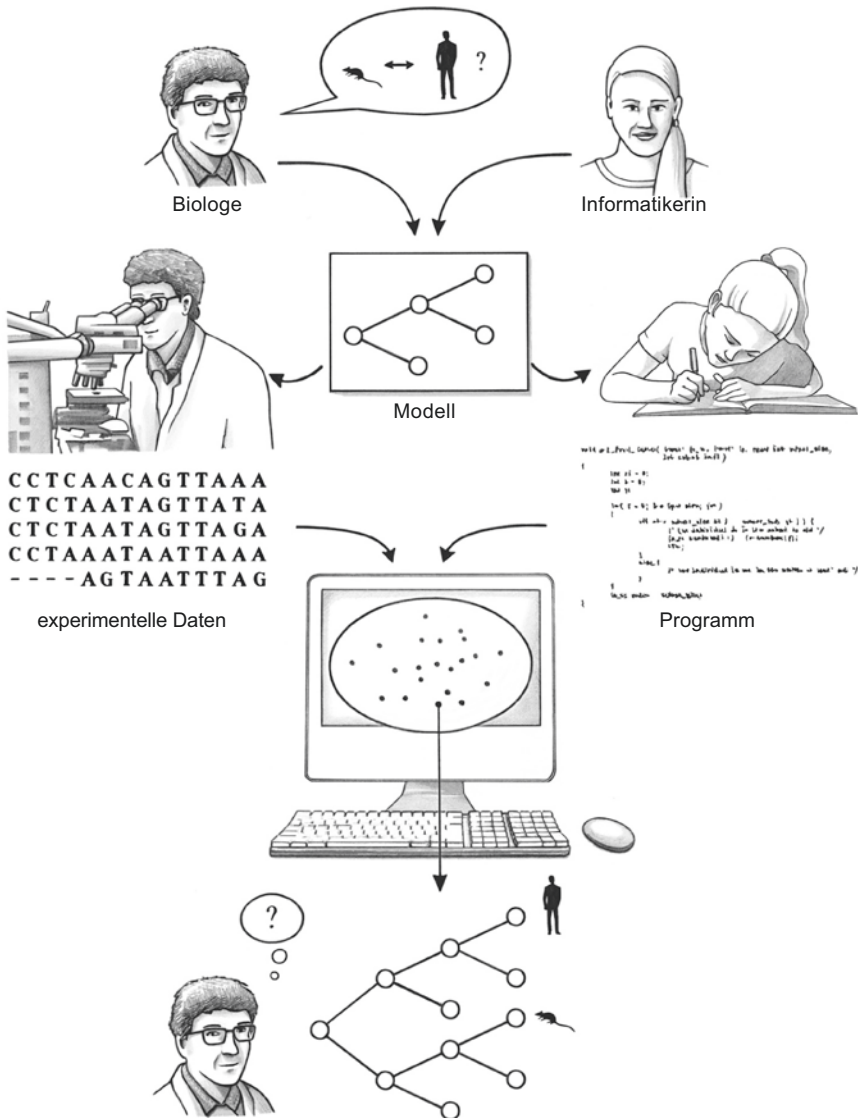


Abb. 9.17 Schematische Darstellung, wie biologische Fragestellungen interdisziplinär mithilfe der Informatik angegangen werden können. (© Susanne Staubli, Eckart Zitzler)

gemeinsam ein Modell, auf dessen Basis anschließend die Informatikerin ein Programm konzipiert und der Biologe Experimente entwirft und durchführt. Die experimentellen, quantifizierten Daten schließlich dienen als Eingabe für das Programm, das nun versucht, den Raum der möglichen Hypothesen mit den experimentellen Daten abzugleichen und die wahrscheinlichste oder vielversprechendste zu ermitteln.

Evolution imitieren

Der Evolution im Zeitraffer zuschauen zu können, wünschen sich wohl viele. Entsprechend überlegten sich Biologen mit dem Aufkommen der ersten Computer, Teilaspekte der Evolution zu simulieren, um Hypothesen zu prüfen und zu neuen Erkenntnissen zu gelangen. Ihre Hoffnung: Indem man die Entstehungsprozesse virtuell beobachten und die Umweltbedingungen gezielt verändern kann, ergeben sich neue Einsichten, wie es sich wirklich zugetragen haben könnte. Es brauchte nicht lange, bis Ingenieure ähnliche Ideen hatten – nur mit einer anderen Zielsetzung. Sie wollten die Evolution nachahmen, um damit innovative Lösungen für technische Probleme zu finden. Und wie so häufig, wenn die Zeit reif ist für gute Ideen, kommen sie fast zeitgleich unabhängig voneinander an verschiedenen Orten zum Vorschein.

In Deutschland waren es Ingo Rechenberg und Hans-Paul Schwefel, die diese Idee wortwörtlich zum Fliegen brachten. Einen durchschlagenden Erfolg erzielten sie, als sie eine unscheinbare Düse mittels simulierter Evolution verbessern konnten. Bei dieser Düse handelte es sich um eine Überschalldüse, wie sie im Flugzeugbau eingesetzt wird. Ihre Aufgabe ist es, einen Luftstrom zu bündeln und anschließend zu beschleunigen. In den 1960er-Jahren kannte man nur eine sehr einfache Form, wie sie in Abb. 9.18 oben zu sehen ist: eine Öffnung, die sich erst verjüngt und anschließend wieder verbreitert. Die spannende Frage war natürlich, ob sich mit anderen Formen die Ausflussgeschwindigkeit der Luft vergrößern lässt. Doch man tappte damals im Dunkeln – zu jener Zeit waren die mathematischen Methoden noch nicht ausgereift genug, um genau zu verstehen, welchen Einfluss die Form auf den Luftfluss hat. Und nun entwickelten die beiden Deutschen eben die Idee, die zwei Prinzipien der Evolution stark vereinfacht anzuwenden, d. h. die Form zufällig zu variieren und immer die

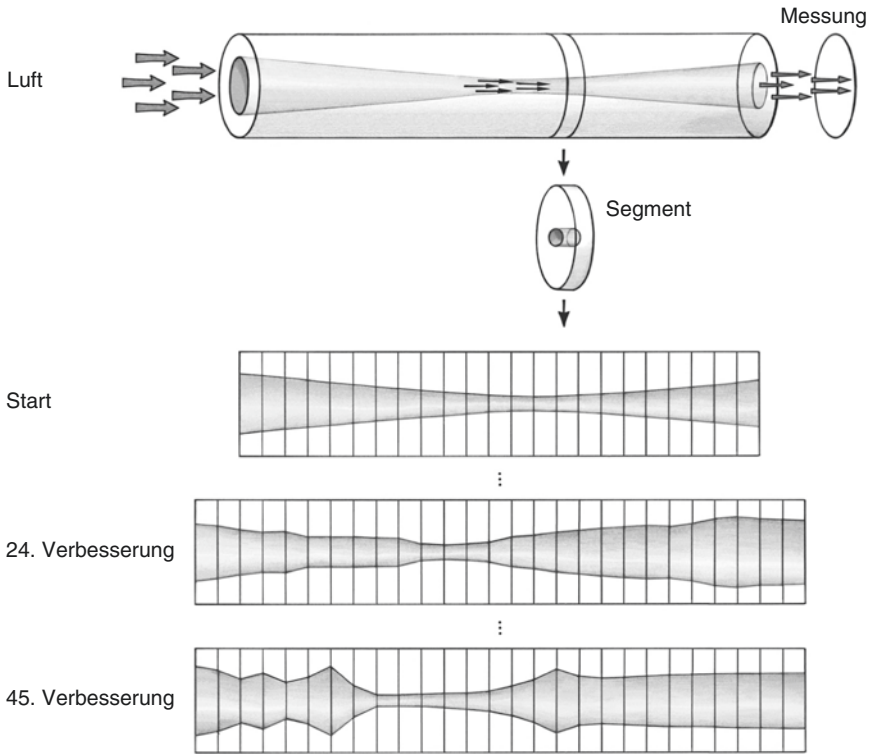


Abb. 9.18 Wie eine Düse durch Imitation des Evolutionsprozesses verbessert werden konnte. (© Susanne Staubli, Eckart Zitzler)

weniger effektive Form durch die erfolgreichere zu ersetzen. Auf diese Weise konnten sie nach 45 Verbesserungsschritten eine Form finden, die äußerst unkonventionell aussah, einen um 30 % höheren Wirkungsgrad als die altbekannte Form aufwies und erst 30 Jahre später auch theoretisch verstanden wurde. Bsp. [9.12](#) erklärt, wie das Experiment im Detail ablief.

Beispiel 9.12: Eine Evolution der Düse

Konkret haben Hans-Paul Schwefel und Ingo Rechenberg folgenden experimentellen Aufbau konzipiert: Die Düse bestand aus mehreren einzelnen Scheiben, jede von ihnen mit einem Loch bestimmter Größe versehen (vgl. Abb. [9.18](#)). Zu Anfang waren die Scheiben so gewählt, dass daraus genau die

altbekannte Düsenform resultierte; die Düse konnte mittels eines Luftstrahls getestet werden, indem die Geschwindigkeit der austretenden Luft gemessen wurde.

Nun wurde die Form zufällig verändert, also mutiert. Dazu wurden Zufallszahlen ausgewürfelt, für jede Scheibe einmal, und die Zufallszahl bestimmte, wie stark sich der Lochdurchmesser in der Scheibe veränderte. Darüber hinaus wurde noch ausgewürfelt, ob Scheiben verschwanden oder hinzukamen. Anschließend wurde die mutierte Form gebaut und die resultierende Luftgeschwindigkeit gemessen. Das Ergebnis wurde verglichen mit der Originalform. Konnte eine Verbesserung erzielt werden, so wurde fortan mit der besseren neuen Form weitergearbeitet in der gleichen Weise – bis wiederum ein noch besseres Design zum Vorschein kam. In Abb. 9.18 ist zu sehen, wie die Form nach der 24. und nach der 45. Verbesserung aussah.

Eigentümlich an Schwefels und Rechenbergs Idee war, dass sie diese ohne Einsatz eines Computers umsetzten – Computer waren damals Mangelware und horrend teuer. Die beiden mussten einige Jahre warten und auf anderen Gebieten forschen, bevor sie den Evolutionären Algorithmen – so heißt das Forschungsfeld, das sie dann in den 1980er-Jahren zu etablieren begannen – zum Durchbruch verhalfen (unabhängig von ihnen kamen Forscher in den USA auf ähnliche Gedanken). Wie funktioniert nun so ein Evolutionärer Algorithmus? Er ist im Prinzip nichts anderes als ein Suchalgorithmus analog zum Prinzip „Versuch und Irrtum“, nur dass der Suchraum nicht systematisch durchkämmt, sondern unter Einbezug des Zufalls abgesucht wird. Er arbeitet mit einer Menge von Lösungskandidaten, die anfangs zufällig generiert werden. Aus dieser Menge wird eine neue Menge von Lösungskandidaten erzeugt, erneut durch zufällige Prozesse, und dabei kommt dann die Idee der Evolution zum Tragen: Selektion und Variation.

Wenn wir einen Evolutionären Algorithmus einsetzen wollen, müssen wir uns zuerst überlegen, wie ein Lösungskandidat für das Problem zu codieren ist, also wie quasi das Genom aussehen soll. Dann müssen wir uns überlegen, wie Lösungskandidaten mutiert bzw. rekombiniert, also variiert werden können. Und schließlich braucht es noch eine Funktion, die jeden Lösungskandidaten bewertet und angibt, wie „gut“ das Individuum ist. Auf dieser Basis lässt sich ein Evolutionärer Algorithmus einfach programmieren. Zuerst wird zufällig eine Menge von Lösungskandidaten generiert, Population genannt. Dann werden neue Lösungskandidaten kreiert, indem in der Population vorhandene Individuen mutiert bzw. miteinander rekombiniert werden. Schließlich werden alle alten und die neuen Lösungskandidaten bewertet und nur die besten werden in der Population behalten. Dann beginnt das Spiel wieder von vorne, bis eine vorgegebene

Anzahl von Generationen durchexerziert wurde. Am Schluss ist der beste Lösungskandidat in der letzten Population die Lösung, die der Evolutionäre Algorithmus produziert hat. Wie sich auf diese Art das Rucksackproblem aus Kap. 8 lösen lässt, können Sie in Bsp. 9.13 nachlesen.

Beispiel 9.13: Evolutionär Rucksäcke packen

Zunächst einmal müssen wir uns überlegen, wie wir eine Auswahl an Gewichten als Lösungskandidat codieren. Das ist in diesem Fall einfach, wie nehmen einfach eine Sequenz von 0 und 1, die genauso viele Zeichen hat, wie Gewichte zur Verfügung stehen. Die Auswahl eines Gewichts wird durch eine 1 an der entsprechenden Stelle in der Sequenz markiert (siehe Abb. 9.19). Am Anfang würfeln wir mehrere dieser Auswahlen aus, sie bilden die Ursprungspopulation. Dann wenden wir die Variation an. Einen neuen Lösungskandidaten kreieren wir, indem wir zwei Auswahlen zufällig ziehen, beide mutieren und anschließend noch miteinander rekombinieren. Mutieren heißt in diesem Fall: Wir gehen die einzelnen Zeichen in der 0-1-Sequenz nacheinander durch und würfeln aus, ob an dieser Stelle das jeweils andere Zeichen

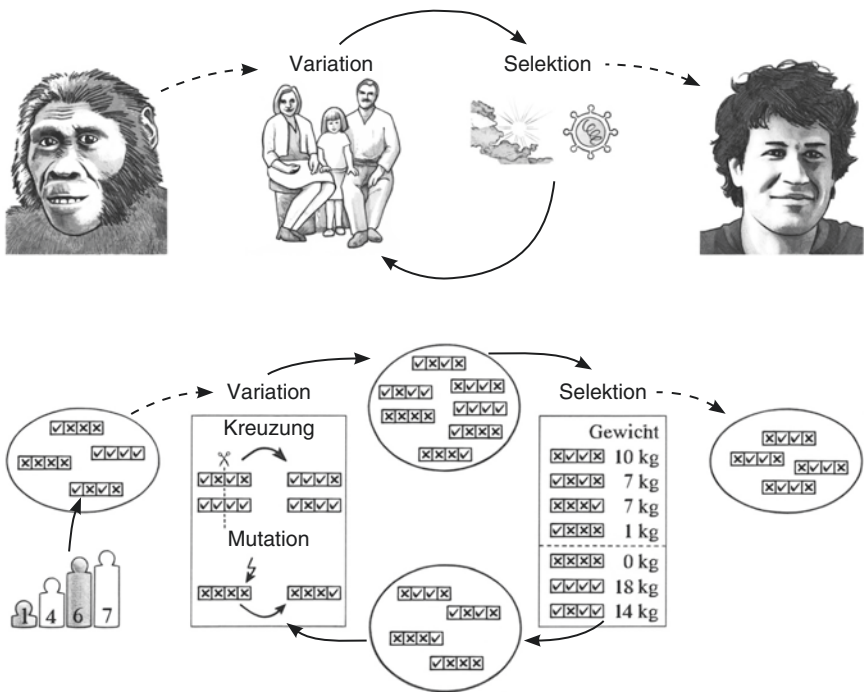


Abb. 9.19 Wie das Rucksackproblem mittels eines Evolutionären Algorithmus gelöst werden kann. (© Susanne Staubli, Eckart Zitzler)

eingesetzt wird oder keine Änderungen stattfindet – die Wahrscheinlichkeit eines Zeichenwechsel ist dabei sehr klein. Und Rekombinieren heißt, ein zufällig gewählter Teil eines Elternteils wird übernommen, der Rest wird vom anderen Elternteil übernommen. Anschließend werden alle Auswahlen, die alten und die neuen, bewertet, sprich: Es wird ausgerechnet, wie gut die einzelnen Lösungskandidaten sind. Gut bedeutet hier, möglichst nahe beim maximalen Gewichtswert zu sein, also die Differenz von ausgewählten Gewichten und dem erlaubten Maximum ist zu minimieren. Die Lösungskandidaten werden entsprechend der Differenz sortiert und nur die besten 50 % bilden die neue Population, die restlichen Auswahlen werden verworfen. Dieser Kreislauf von Variation und Selektion wird anschließend wiederholt – die genaue Zahl der Wiederholungen muss der Anwender wählen –, und die beste Auswahl in der Population ist die Ausgabe des Evolutionären Algorithmus.

Eine häufig gestellte Frage im Zusammenhang mit Evolutionären Algorithmen lautet: Wann weiß das Verfahren, dass eine optimale Lösung gefunden wurde? Die Antwort kann irritieren: Es weiß es nicht, kann es nicht wissen. Evolutionäre Algorithmen gehören zu den randomisierten Algorithmen, die gezielt den Zufall einsetzen, und stellen Heuristiken dar. Sie sind darauf angelegt, bestehende Lösungskandidaten zu verbessern, und nicht darauf, nachweislich die beste Lösung zu generieren. „Aber welchen Zweck hat das Verfahren dann, wenn man nichts über die Güte aussagen kann?“, könnten Sie zu Recht fragen. Nun, es wird immer dann eingesetzt, wenn man keinen besseren Algorithmus kennt, wenn man tatsächlich im Nebel stochert – und das tun Evolutionäre Algorithmen, nur auf geschickte Art und Weise. Ein großer Anwendungsbereich von Evolutionären Algorithmen sind Entwurfsprobleme, z. B. um den Kabelbaum in einem Auto zu verlegen, wobei naturgemäß mehrere Bewertungskriterien zum Einsatz kommen. Weil die Entwurfsprobleme so komplex sind, sind Ingenieure bereits froh, wenn sie ein Tool an der Hand haben, das ihnen Verbesserungsvorschläge macht und Anregungen liefert. Und dafür eignen sich Evolutionäre Algorithmen bestens.

Ist das bereits Kreativität? Spannend ist sicherlich, dass so innovative Lösungen kreiert werden können, auf die Menschen evtl. noch nicht gekommen sind. Aber es ist Vorsicht angebracht. In der ersten Euphorie verfällt man bei solchen Verfahren häufig der verlockenden Variante, simulierte Evolution mit Evolution gleichzusetzen und überall Parallelen zu sehen und sie entsprechend zu interpretieren. Das ist gefährlich, denn bei allem Potenzial: Evolutionäre Algorithmen sind ein geradezu lächerlich simples Modell der natürlichen Evolution. Aussagen vom Algorithmus zur Natur und umgekehrt sind nur in äußerst beschränktem Maße möglich.

Macht auch nichts. Trotzdem ergeben sich auf beiden Seiten spannenden Fragestellungen, die wiederum auch die Gegenseite inspirieren können.

Fazit

In diesem Kapitel haben wir uns von den reinen Analogien zwischen Informatik und Biologie gelöst und untersucht, wo beide Gebiete tatsächlich ineinandergreifen. Primär ging es jedoch um Software-Technik. Wenn wir uns die Prinzipien anschauen, nach denen Software-Systeme entwickelt werden, dann fällt auf, dass Natur und Informatik ähnliche Lösungsprinzipien gefunden habe, um immer neue Schichten der Komplexität aufzusetzen. Des Weiteren unterliegt die Software, ebenso wie die Hardware, einer ganz eigenen Evolution, auch sie baut immer auf Altbekanntem auf und ergänzt es. Und die Software eröffnet der Informatik neue Anwendungsfelder. Sie haben das Gebiet der Bioinformatik kennengelernt, das sich mit dem Einsatz informatischer Methoden beschäftigt, um Probleme in der Biologie zu lösen. Grundlage dafür ist, dass ein geeignetes Modell formuliert werden kann, mittels dessen die Hypothese mathematisch präzise zu formulieren ist. Ein Software-System kann schließlich in einem riesigen Raum an potenziellen Hypothesen nach vielversprechenden suchen und somit dem Biologen Anhaltspunkte liefern, was aus bestimmten experimentellen Daten gewonnen werden kann. Doch auch die andere Richtung gibt es, wo die Biologie für die Informatik Pate steht. Biologische Modelle der Wirklichkeit werden in informatischen Algorithmen übersetzt, um sie zur Lösung technischer Probleme einzusetzen. Wir sehen also: Die Verbindung von Informatik und Biologie ist an sich eine natürliche.

Zum Nach- und Weiterlesen

- Deutscher, G. (2011). *Du Jane, ich Goethe: Eine Geschichte der Sprache*. München: Deutscher Taschenbuch Verlag.
- Fogel, D. B. (1998). *Evolutionary computation: A fossil record*. Hoboken: Wiley-IEEE Press.
- Gombrich, E. H. (2000). *Die Geschichte der Kunst*. Berlin: Phaidon.
- Gusfield, D. (1997). *Algorithms on strings, trees and sequences: Computer science and computational biology*. Cambridge: Cambridge University Press.
- Hölldobler, B., & Wilson, E. (2010). *Der Superorganismus: Der Erfolg von Ameisen, Bienen, Wespen und Termiten*. Berlin: Springer Spektrum.

- Neffe, J. (2010). *Darwin: Das Abenteuer des Lebens*. München: Goldmann.
- Sommerville, I. (2012). *Software Engineering* (9. Aufl.). München: Pearson Studium.
- Stallings, W. (2014). *Network security essentials: Applications and standards* (5. Aufl.). Essex: Pearson Education Limited.

Bildquellen

Abb. 9.14: pixelprof/Getty Images/iStock, Bildnummer 511121338, <http://www.istockphoto.com> (links, abgerufen am 30.05.2017); GoodLifeStudio/Getty Images/iStock, Bildnummer 515002420, <http://www.istockphoto.com> (rechts, abgerufen am 30.05.2017).

Epilog: Zur Biologie der Informatik

Sie haben nun in des Computers Hirn hineingeschaut und erfahren, nach welchen Prinzipien dort Informationen umgewandelt werden – und natürlich wissen Sie auch, dass ein Computer weder ein Gehirn hat, noch selbst eines ist, genauso wenig wie ein Gehirn ein Computer ist. Der Buchtitel spielt auf die Analogien zwischen künstlicher und natürlicher Informationsverarbeitung an, von denen Sie in den vorangegangenen neun Kapiteln verschiedenste kennengelernt haben. Die Informatik als eigenständige Disziplin hat also zahlreiche Berührungspunkte mit anderen Gebieten. Und wenn wir uns vergegenwärtigen, wie grundlegend der Begriff der Information für alle Fachrichtungen ist, müssen wir eigentlich sagen: Die Natur der Informatik ist interdisziplinär.

Die Natur der Informatik ist interdisziplinär

In meiner eigenen Forschungstätigkeit war es vor allem die Beschäftigung mit der Biologie, die mir ein tiefer gehendes Verständnis für die Informatik vermittelte, die neue Gedankengänge anregte und mir darüber hinaus vor Augen führte, wie wesentliche Fragen die Disziplinen überspannen: Was ist Information, was heißt Rechnen, wie funktioniert Kommunikation, was macht ein gutes Gedächtnis aus, was verstehen wir unter Intelligenz, was bedeutet Lernen, was ist geschickte Arbeitsteilung usw.? Da wird aus Informatik nicht nur Biologie, sondern Physik, Mathematik, Psychologie, Soziologie, Politik und letztlich Philosophie. Auf einmal geht es um das große

Ganze, den großen Bogen. Und ich hätte zu Beginn meines Informatikstudiums kaum vermutet, wie sehr die Informatik von anderen Disziplinen beeinflusst wird, vor allem von der Biologie und den Lebenswissenschaften allgemein, und wie sehr sie selbst wiederum zurückwirkt. Dabei, wir haben es gesehen, dachten Informatikpioniere wie John von Neumann und Alan Turing bereits von Anfang an interdisziplinär und betätigten sich als Wandler zwischen den Fachwelten. Ersterer war fasziniert von der Neurologie und versuchte immer wieder gedankliche Brücken zwischen Rechenmaschine und Gehirn zu schlagen; darüber hinaus beschäftigte ihn die Frage, wie sich die Fähigkeit des Lebens, sich selbst zu vervielfältigen, auf technische Systeme übertragen ließe. Letzterer setzte sich mit dem Thema der Intelligenz auseinander und leistete zudem Beiträge zur Theoretischen Biologie, indem er Erklärungsansätze entwickelte, wie Organismen Formen wie Tannenzapfen und Muster wie Zebrastrifen hervorbringen können.

Heutzutage lässt sich der interdisziplinäre Charakter der Informatik vor allem an den Namen der unzähligen Forschungsfelder ablesen, die mit ihr eine Symbiose bilden. Im Englischen wird häufig der Begriff *Computational* vorangestellt, um die informatischen Aspekte innerhalb einer Naturwissenschaft zu bezeichnen: Das Spektrum reicht von *Computational Biology* über *Computational Physics* und *Computational Chemistry* bis zu *Computational Science*, also wissenschaftlichem Rechnen ganz allgemein. Eine bemerkenswerte Vielfalt gibt es an der Schnittstelle zwischen Informatik und Biologie. Während Bioinformatik und *Bio-inspired Computing* jeweils für ein weit gefasstes Themenfeld stehen, beschreiben neuronale Netze, Evolutionäre Algorithmen, Ameisenalgorithmen und Schwarmalgorithmen enger eingegrenzte Gebiete; auch die Künstliche Intelligenz lässt sich im weitesten Sinne dazuzählen. Am Ende geht es aber auch um die Rolle der Informatik in der Wissenschaft an sich. Der bekannte Informatiker Christos Papadimitriou hat 2008 an der Eidgenössischen Technischen Hochschule Zürich eine interessante These vorgetragen: Die Informatik sei die neue Mathematik. Was er damit meint? Dass der Informatik heutzutage eine ähnliche Rolle wie der Mathematik zukommt. Sie fungiert in vielen Gebieten als Hilfswissenschaft und treibt damit den Fortschritt in diesen voran. Neue Erkenntnisse in den Naturwissenschaften und den Ingenieurdisziplinen werden häufig unter Zuhilfenahme des Computers gewonnen; vielfach ist er als Werkzeug gar nicht mehr wegzudenken, beispielsweise in der Molekularbiologie oder der Klimaforschung. Es ist aber auch die Perspektive der Informatik, die das wissenschaftliche Denken in verschiedenen Disziplinen beeinflusst und verändert. Das heißt, genauso wie in vielen Fächern Mathematikkenntnisse

vorausgesetzt werden, braucht es mittlerweile auch ein Grundlagenwissen in Informatik, um fachspezifisch arbeiten zu können.

Ich bin überzeugt, diese Anknüpfungspunkte aufzuzeigen und in den Fokus zu rücken, gerade zur Biologie, kann die abstrakten Informatikinhalte mit Leben füllen und sie um Sinnfragen bereichern – letztendlich eben Technik mit Natur verbinden. Die Interdisziplinarität der Informatik ist auch ein vielversprechender Ausgangspunkt, um über informatische Allgemeinbildung nachzudenken. Dieser im Zusammenhang mit Wissensvermittlung wenig beachtete Wesenszug birgt ein besonderes Lehr- und Lernpotenzial in sich und kann neue Perspektiven eröffnen. Der Erfolg bekannter Longseller wie *Gödel, Escher, Bach* von Douglas Hofstadter aus dem Jahr 1979 oder *Computerdenken* von Roger Penrose aus dem Jahr 1989, die Facetten der Computerwissenschaft in eine fachübergreifende Gesamtsicht einbetten, spricht dafür. Und zu guter Letzt: Das Bauen und das Verstehen liegen eng beieinander, oder wie es der berühmte Physiker und Nobelpreisträger Richard Feynman gesagt haben soll: „What I cannot create, I do not understand.“ Insofern bin ich überzeugt davon, dass die Auseinandersetzung mit der automatisierten Informationsverarbeitung wertvolle Einsichten und Anregungen zum Verständnis der natürlichen Informationsverarbeitung generieren kann – und umgekehrt.

Dank

Zunächst bedanke ich mich bei Ihnen, dass Sie sich auf dieses Buch und meine Gedankengänge eingelassen haben. Wenn Sie etwas für sich mitnehmen und neue Einsichten gewinnen konnten, freut mich das sehr. Auch ich habe während der Entstehung des Buchs viel gelernt, und jedes Kapitel hat über die Zeit einen erstaunlichen Wandel durchlebt. Glücklicherweise wusste ich nicht, wie viel Arbeit da auf mich wartete, als ich enthusiastisch die ersten Ideen zu diesem Buch aufschrieb ... Ich habe davon profitiert, dass sich viele Menschen vor mir die Mühe gemacht haben, Wissen zusammenzutragen, aufzubereiten und verständlich darzustellen. Die meisten dieser Quellen sind am Ende der Kapitel aufgeführt, und besonders möchte ich den Autorinnen und Autoren der Zeitschrift *GEO* danken, in deren fundiert recherchierten und ansprechend geschriebenen Artikeln ich immer wieder wertvolle Hinweise und Anregungen fand.

Zudem wurde ich bei diesem Buchprojekt von verschiedenen Seiten tatkräftig unterstützt. Mein herzlicher Dank gilt (in alphabetischer Reihenfolge): **Sven Bergmann** für spannende Diskussionen rund um die

Bioinformatik; **Camille Büsser** für das motivierende Anspornen, das interessierte Nachfragen und vor allem dafür, dass er von ausgewählten Informatikpionieren Porträts zeichnete und sie mir für dieses Buch zur Verfügung stellte; **Jeannette Büsser** für die gute Laune, das Mitfühlen und das Daran-Glauben; meiner Frau **Judith Büsser** für die wunderbaren Pings und Pongs, die schönen Fotos und ihre Liebe in allen Gemütslagen des Autorendaseins; **Uwe Dirksen** für die kompetenten fachlichen Rückmeldungen, die witzigen Kommentare zu meinen Stilblüten und die regelmäßigen „Aufsteller“; **Felix Etter** für die zahlreichen Gespräche über Mittag, nach denen die (Buch)Welt jedes Mal wieder im Lot war, und die lehrreichen Einblicke in das Berufsleben eines Computerdoktors; **Helene Fischnaller, Harald Seppi, Leo Seppi** und **Philipp Seppi** für zwei ungemein kurzweilige, lustige und produktive Schreiburlaube in Südtirol und ihre grandiose Gastfreundschaft; **Silvia Gfeller** für ihre didaktischen Anregungen und das gemeinsame Fernsehschauen vor dem Magnolienbaum; **Werner Hartmann** für das genaue Lesen des Manuskripts, die Denkanstöße und die differenzierten, unverblünten und umso hilfreicheren Feedbacks; **Gottfried Hodel** für sein Wohlwollen und das Gewähren der Freiräume, die für den Abschluss des Projekts nötig waren; **Verena Hunziker** für Ohr, Klingel, Dasein und Mutmachen von Anfang bis Ende; meinen **Kolleginnen** und **Kollegen** und meinem **Team** an der Pädagogischen Hochschule Bern, die mit großer Nachsicht und Verständnis dem Gedeihen dieses Werkes entgegen sahen; **Christoph Kläy** für das kabarettistische Intermezzo, das mir gezeigt hat, dass es auch nach dem Buch noch viele aufregende Projekte geben wird; **Verena Kovatsch-Guldimann** für ihre Argusaugen und das Bewusstmachen, wie gerne ich doch eigentlich schon überflüssige Füllwörter benutze; **Christian Kropf** für seine inspirierende Spinnenexkursion und die umfassenden Informationen zu den Bola-Spinnen; **Barbara Lühker** für ihre fachkundige und zuverlässige Begleitung in allen Lektoratsfragen; **Martin Merz** für seinen Lehrer- und Übersetzerblick, der ganz neue Aspekte zutage förderte; **Michèle Minelli** für die wertvollen Schreibtipps und ihre Frage: „Wo ist das Problem, Du bist doch nahezu fertig?“, als ich das Licht am Ende des Tunnels nicht sah; **Frithjof Nolting** für sein Brudersein und all sein Dazutun; **Marion Nitsch** für die beherzte und gleichsam geduldige Geburtshilfe, als die Ideen noch wild und der Weg zur Erstfassung im dichten Nebel eingehüllt waren; **Kurt Reber** für seine unkomplizierte Art, die positive Einstellung und dafür, dass er mir den Rücken freigehalten hat; **Andreas Rüdinger** für die gute, langjährige Zusammenarbeit, seinen Einsatz für dieses Projekt und das Vertrauen, das selbst Bücher, deren Entstehung in den Zeitdimensionen des Verlagswesens eine Ewigkeit beansprucht, irgendwann



Abb. E.1 Dem Autor ins Manuskript geschaut. (© Judith Büsser)

einmal fertiggestellt sein können; **Yvonne Ruoss** für's Mitfiebern, Mitdenken und all die Ermunterungen; **Martin Schäfer** und der **Pädagogischen Hochschule Bern** für den Bildungsurlaub, der mir erlaubte, den entscheidenden Schritt vorwärtszukommen; meiner Schwester **Jana Schlegel** dafür, dass sie sonntags mit mir geduldig durch das Arithmeum in Bonn schlenderte, damit ich Fotos für dieses Buch machen konnte; **Susanne Staubli** für ihre großes Engagement, die Bereitschaft, sich auf ein so langwieriges und arbeitsintensives Experiment einzulassen, ihre Geduld mit mir und ihre tollen Zeichnungen, ohne die das Buch gar nicht denkbar wäre; meiner Schwester **Kristina von Rintelen** für den Anstupser, das Naturkundemuseum in Berlin zu besuchen, was mir eine Fülle an neuen Ideen einbrachte. Schließlich danke ich allen Cafés, die ich zu meinem Zweitarbeitszimmer machen durfte und in denen unzählige handgeschriebene Puzzlestücke für das Manuskript entstanden (Abb. E.1), ganz besonders dem **Gnädinger** in Zürich, dem **Sattler** in Bern, dem **Città** in Bozen und dem **Tibits** in Bern.

Stichwortverzeichnis

A

- Abakus 5
- Addition 7
- Ablaufplanung 181
- address resolution protocol* 260
- Adleman, Leonard 37, 330
- Advanced Encryption Standard* 319
- AES (*Advanced Encryption Standard*) 319
- agile Methode 426
- Agiles Manifest 427
- Aktor 155
- ALAP (*as least as possible*) 180
- al-Chwarizmi 349
- Algorithmus 348
 - effizienter 389
 - evolutionärer 438
 - pseudopolynomieller 356
 - randomisierter 349
- Alphabet 372
- alpha-beta pruning* 371
- Amdahl, Gene 172
- Amdahlsches Gesetz 172
- Ameisen 278
- Ameisen-Algorithmus 279
- Aminosäure 31
- Amplitudenmodulation 242
- analog 45
- Analog-Digital-Wandlung 152, 154
- Analytical Engine 11
- Anpassungsfähigkeit 229, 360
- Antizipation 363
- Apple 406
- ARP (*address resolution protocol*) 260
- array(Feld) 212
- ASAP (*as soon as possible*) 180
- ASCII (*American Standard Code for Information Interchange*) 290
- Assembler 127
- Assemblersprache 127
- Assoziation 230
- assoziativer Speicher 233
- asymmetrische Verschlüsselung 329
- asynchroner Datenaustausch 157
- Aufmerksamkeit 167, 177
- Ausgabe 142, 148, 151
 - Aktivitätsmuster 54
 - Aktor 155
 - Muster 54
 - Nervenzelle 88
 - Stromkreis 55, 61
- Automat
 - endlicher 377
- Axon 88

B

- Babbage, Charles 11
- back propagation* 367
- backtracking* 351
- Batch-Betrieb 175
- Baum 216
 - Wurzel 216
- Baustein
 - Speicherelement 83
- Bayessches Netz 364
- Berners-Lee, Tim 267
- Betriebssystem 160, 184, 403
 - Aufgaben 185
- Bilddarstellung 292
 - Rasterbild 292
 - Vektorgrafik 294
- Binärbaum 217
- binäre Suche 213
- Binärzahl 49
- Bioinformatik 434
- Biologie viii, 402
- biologische Evolution 401
- biologischer Code 296
- Bit 109, 198, 299
- Bitcoin 271
- Bitmap 292, 316
- Bitübertragungsschicht 260
- Blockchain 271
- Block-Code 241
- Bola-Spinne 335
- Boole, George 73
- Boolesche Algebra 75
- Booten 186
- Botnet 410
- branch and bound* 353
- Bridge 248
- brute force* 37, 350
- Buch 192
- Bug 415
- Bus 148, 156
 - Adressleitung 156
 - Datenleitung 156
 - Hierarchie 163
 - Northbridge 164
 - Protokoll 157
 - Southbridge 164
 - Transaktion 157
- Busy-Beaver-Funktion 384
- Byte 198

C

- C (Programmiersprache) 417
- Cache 206
- Cäsar-Chiffre 313
- Cerf, Vinton G. 262
- Chiffre 313
- Chiffrierscheibe 314
- Chomsky, Noam 374
- Chomsky-Hierarchie 374
- Church, Alonzo 380, 384
- Churchsche These 24
- Client 264
- Client-Server-Modell 264
- CMYK (*cyan, magenta, yellow, black*) 293
- Code 287, 296
 - ASCII 290
 - biologischer 296
 - Block-Code 241
 - fehlererkennender 308
 - fehlerkorrigierender 308
 - Geheimcode 312
 - genetischer 297
 - Kriterien 289
 - Leitungscode 241
 - Manchester-Code 241
 - Maschinencode 291
 - Unicode 290
- Codewort 287
- Codewortlänge 299

Compiler 405, 416
Computer
 Definition 24
 elektronischer 12
 Kerneigenschaften 14
 menschlicher 146
 Von-Neumann-Architektur 148, 150
Computermodell 26
Computernetzwerk 238
Computer-Schichtenmodell 404
Cook, Stephen A. 394
Core 171
CPU (*central processing unit*) 96, 170
CSMA/CA (*carrier sense multiple access, collision avoidance*) 246
CSMA/CD (*carrier sense multiple access, collision detection*) 246

D

daisy chain 169
Data Encryption Standard 317
Datagramm 256
Datei 201
Dateiserver 265
Dateisystem 201, 265
Daten 286
Datenaustausch
 asynchroner 157
 synchroner 157
Datenbank 225
Datenfluss 116
Datenformat 296
Datenkomprimierung 303
Datenpfad 99, 106
 Statusleitung 108
 Steuerleitung 108
Datenstruktur 208
 Baum 216
 dynamische 218
 Feld 212
 Graph 221
 hybride 225

Liste 221
Speicherbedarf 224
Stapel 210
statische 218
Zeiger 219
Zugriffszeit 223
Datenträger 286
Datentyp 416
Datenübertragung
 elektrische 240
 Funk 240
 optische 240
da Vinci, Leonardo 141
Deadlock 184
Deep Blue 369
Deklaration 417
deklarative Programmierung 419
Dendrit 88
denial of service 411
DES (*Data Encryption Standard*) 317
Determiniertheit 349
Determinismus 349, 392
Deutscher, Guy 421
Dezimalsystem 47
D-Flipflop 83
Diffie, Whitfield 320, 330
digital 45
Digital-Analog-Wandlung 154
digitale Schaltung 55
direct memory access 168
 Controller 168
disjunktive Normalform 75
distributed denial of service 411
DMA (*direct memory access*) 357
 Controller 168
DNA
 Doppelkettenmolekül 64
DNA (*deoxyribonucleic acid*) 27, 227
DNA-Computing 36
 Ablauf 38
DNA-Reparatur 311
DNA-Speicher 230
DNS (*domain name system*) 258

DNS-Server 258, 265
 Domain 258
domain name system 258
 Dualsystem 48
 Dualzahl 49
 Addition 50
 Multiplikation 112
 Plus-Baustein 66
 Duft 334
 Rezeptor 334
 dynamische Finitheit 349
 dynamische Programmierung 355

E

EEPROM (*electrically erasable program-
 mable read-only memory*) 199
 Effizienz 389
 Eingabe 142, 148, 151
 Aktivitätsmuster 54
 Muster 54
 Nervenzelle 88
 Sensor 155
 Stromkreis 55, 61
 eingebettetes System 3, 155
 Einwegfunktion 322
 Elektronenröhre 56
 elektronischer Computer 12
 E-Mail 266
 endlicher Automat 377
 Endlosschleife 385
 ENIAC (*Electronic Numerical Integrator
 and Computer*) 12, 149
 Enigma 319
 Entropie 301
 Entscheidungsproblem 345
 Entwurfsprinzip 402
 Enzym 29
 Epigenetik 228
 Erinnerung 230
 Erosion 421
 Eselsbrücke 232
 Ethernet 243, 245, 247, 248
 EUI-64-Adresse 247

Evolution 400
 biologische 401
 kulturelle 401
 Evolutionärer Algorithmus 438
 Exception 165
 exponentielles Wachstum 389

F

Farbdarstellung 293
 Fehlererkennung 308
 Fehlerkorrektur 202, 308
 Feld 212
 Festplatte 200
 RAID 202
File Transfer Protocol 264
 Finitheit 349
 Firewall 412
 Fixkommadarstellung 289
 Flaschenhals 158
 Flash-Speicher 196
 Flipflop 81, 104
 biologisches 86
 flankengesteuertes 83
 Grundbaustein 82
 Prinzip 78
 formale Sprache 372
 Formatierung 201
 Fragmentierung 179
 Frame 243
 Frequenzcodierung 154
 Frequenzmodulation 242
 FTP (*File Transfer Protocol*) 264
 Funk 240, 242

G

Gates, Bill 405
 Gatter 69
 biologische 75
 Neuronen 90
 NICHT-Gatter 69
 ODER-Gatter 69
 UND-Gatter 69

Gedächtnis 361
gegenseitiger Ausschluss 182
Geheimcode 312
Gehirn 87, 360
 Großhirn 167
 Großhirnrinde 91, 174
 Nervenzelle 88
 Zwischenhirn 167
Gen 29, 64
 Aktivitätsmuster 53
 Binderegion 64
 Promotor 64
 Spleißen 310
 Überlagerung 310
genetischer Code 297
Genom 429
Genregulation 30
Geruch 334
Geruchsrezeptor 334
gierige Strategie 354
Gleitkommazahl 289
Go 371
Gödel, Kurt 381
Gödel-Nummerierung 381
Gotik 427
GPU (*graphical processing unit*) 170
Grafikprozessor 170
Grammatik 194, 372, 374
Graph 221, 432
greedy strategy 354
Großhirn 167
Großhirnrinde 91, 174
größter gemeinsamer Teiler 347

H

Halteproblem 382
Hamilton-Pfad-Problem 37, 279, 344, 351, 390
Handlungsreisenden-Problem 390
Handschrifterkennung 365
Handshaking-Protokoll 158
Hardware 120

Hashing 225
Hauptspeicher 99, 102, 148
 Adresse 102
Heartbleed-Bug 413
Hellman, Martin 320, 329, 330
Heuristik 354, 440
Hierarchisierung 403
Hilbert, David 380
Hormonsystem 275
HTML (*Hypertext Markup Language*) 267, 294
HTTP (*Hypertext Transfer Protocol*) 267
Huffman-Baum 303, 305
Huffman-Code 305
Hypertext 267
Hypertext Markup Language 267, 294
Hypertext Transfer Protocol 267

I

IBM (*International Business Machines Corporation*) 405
imperative Programmierung 418
Implementierung 424
Informatik vii
Information 284
 physikalische Repräsentation 286
 symbolische Repräsentation 286
Informationsgehalt 301
Innovation 421
Instruktion 123
Instruktionssatz 123
Intelligenz 360
 künstliche 360
Internet 252
Internetadresse 258
Internet-Domain 258
Internet-Protokoll 256
Interpreter 405, 416
Interrupt 165, 176
 Maskierung 167
IP (Internet-Protokoll) 256
IP-Adresse 258

J

Java 404
Jobs, Steve 406

K

Kahn, Robert E. 262
Kalkül 379
Kapselung 403
Kasparow, Garri 369
Kern
 suprachiasmatischer 164
Kleene, Stephen 372, 380
Kleenesche Hülle 372
Kognition 361
Kolibakterie 26, 85
Kollisionsdetektion 246
Kombinations-Verschaltung 60
Komplexität 400
Komplexitätsklasse 388
Komprimierung 303
 verlustbehaftete 307
 verlustfreie 306
kontextfreie Grammatik 374
kontextsensitive Grammatik 374
Kontrollfluss 116
Kontrollleitung 156
Kontrollpfad 99, 109, 115
Kreativität 440
Kryptografie 312
 visuelle 316
kubisches Wachstum 389
Kuchenproblem 347, 357
kulturelle Evolution 401
künstliche Intelligenz 360
künstliches neuronales Netz 364

L

Lactose 30
Lambda-Kalkül 380
Lambda-Phage 85
LAN (*local area network*) 248
Lauf längencodierung 303

Laufzeit 386
Leibniz, Gottfried Wilhelm 49
Leitungscode 241
Lernen 229, 361, 363
Lichtwellenleiter 240
lineares Wachstum 388
Linux 185
Liste 221
local area network 248
Lochkarte 20
logische Adresse 259
Lokalitätsprinzip 204
Lovelace, Ada 11

M

MAC-Adresse 247
magnetischer Speicher 195
Manchester-Code 241
Maschinencode 126, 291
Maschinensprache 126, 416
Massenspeicher 192
 Festplatte 200
mechanische Rechenmaschine 8
 Analytical Engine 11
 Pascaline 10
 Stephenson-Addierer 8
Mehrband-Turing-Maschine 32
Mehrzeller 227
memory management unit 204
Mergesort 357
Merkle, Ralph 329
Microarray 53
Microsoft 405
Milchzucker 30
Mimikry 336
Minmax-Strategie 370
MMU (*memory management unit*) 204
Modell 346
Modularisierung 403
Modulo-Arithmetik 323
Modulo-Funktion 323
Moore, Gordon 207

Mooresches Gesetz 207
Morse-Code 288
mother board 143
Multicore-Prozessor 171
multiples Sequenzalignment 433
Multiplexer 107
Multiprozessor-System 171
Murray Hopper, Grace 415
Muster, Zuordnung 54, 69

N

Nachrichtentrommel 239
Napster 269
nearest neighbor 364
Nerv 159
Nervenimpuls 89, 341
Nervensystem 87, 159, 229, 275, 360
 vegetatives 161, 341
Nervenzelle 88
Netz
 neuronales, künstliches 364
Netzwerk 238
 Bridge 248
 Bustopologie 245
 hierarchisches 250
 Kollisionsdetektion 246
 Ringtopologie 245
 vollvermaschtes 244
Netzwerkadapter 244
Netzwerkadresse
 Internet 258
 logische 259
 MAC-Adresse 247
 physikalische 259
Netzwerkprotokoll 243
Netzwerksegment 250, 253
Netzwerk-Switch 250
Neuron 88
neuronales Netz 364
Neurotransmitter 89
Nichtdeterminismus 392
NICHT-Gatter 69

Nichtterminalsymbol 373
nop-Befehl 123
Normalform
 disjunktive 75
Northbridge 164
NP (nichtdeterministisch polynomielle
 Zeit) 393
NP-Vollständigkeit 394
Nucleotid 28

O

objektorientierte Programmierung 419
ODER-Gatter 69
öffentlicher Schlüssel 325
One-Time-Pad 315
O-Notation 389
Ontogenese 187
Optimierungsproblem 345
optischer Speicher 196
optischer Telegraf 239
Orca 361
OSI-Schichtenmodell 260, 403

P

P (Menge aller lösbaren Probleme) 393
Parallelisierung, Speedup 172
Parallelverarbeitung 172
Parallelverschaltung 58
Parität 203
Pascal, Blaise 11
Pascaline 10
Pawlow, Iwan 231
Peer-to-Peer-Modell 268
Pfeifsprache 239
Phasenmodulation 242
Pheromon 279, 335
phylogenetischer Baum 432
physikalische Adresse 259
Pixel 292
PLA (*programmable logic array*) 92
Plus-Baustein 72
pointer 219

Polling 165
 polynomielle Reduktion 391
 polynomielles Wachstum 388
 Population 438
 Port 265
 Post 252
 Primfaktorzerlegung 330
 privater Schlüssel 325
 Problem 343
 des Handlungsreisenden 390
 kombinatorisches 345
 Probleminstanz 344
 Programm 24, 125, 350
programmable logic array 92
 Programmausführung 127
 Programmierparadigma 418
 Programmiersprache 372, 403, 416, 419
 Programmierung
 deklarative 419
 imperative 418
 objektorientierte 419
 strukturierte 419
 Promotor 64
 Protein 29, 227
 Bauplan 29, 64
 Schalterprotein 31, 63
 Protokoll 157, 243
 Handshaking 158
 Prozess 177
 Deadlock 184
 gegenseitiger Ausschluss 182
 Kontext 177
 Umschaltung 176
 Prozessor 96, 98, 142
 Befehle 120
 Befehlscode 121
 Datenpfad 99
 Hauptspeicher 99, 102
 Kern 171
 Kontrollpfad 99
 Programmzähler 128
 Rechenwerk 99, 100

Register 99, 102
 Steuerwerk 99, 115, 128
 pseudopolynominaler Algorithmus 356
 Public-Key-Kryptografie 325, 330
 RSA 330

Q

quadratisches Wachstum 388

R

RAID (*redundant array of independent disc*) 202
 RAM (*random access memory*) 199
random access memory 199
 randomisierter Algorithmus 349, 440
 Ransomware 412
 Rasterbild 292
 Rauchzeichen 239
 Rechenberg, Ingo 436
 Rechenmaschine
 elektrische 51
 mechanische 8
 Rechenoperation 19
 Rechenwerk 99, 100, 147
 Rechenzentrum 146
 Rechnen 15, 62
 Automatisierung 17, 112
 Textbearbeitung 18
 Rechner 5
 Reduktion 391
redundant array of independent disc 203
 Redundanz 302
 Reflex 169
regional internet registry 257
 Register 99, 102
 reguläre Grammatik 374
 Reihen-Verschaltung 58
 Rekursion 357
 Relais 55
 Geschichte 56
 öffnendes 55
 schließendes 55

remote procedure call 271
RGB (*red, green, blue*) 293
Rice, Henry Gordon 385
Ritchie, Dennis 417
Rivest, Ron 330
RNA (*ribonucleic acid*) 227
RNA-Polymerase 28
ROM (*read only memory*) 199
Router 253
Routing-Tabelle 254
RPC (*remote procedure call*) 271
RSA (Rivest, Shamir, Adleman) 330, 396
Rückenmark 159
Rückkopplung
 Neuronen 92
 Proteine 86
 Stromkreis 78
Rucksackproblem 325, 354, 439
Rückverfolgung 351
Russell, Bertrand 380

S

SAT-Problem 394
Schach 172, 367
Schad-Software 408
Schalter 44
 Neuron 88
 Relais 55
 Transistor 56
Schalterprotein 31
Schaltung 55
 Aussagenlogik 74
 Baustein 65
 digitale 55
 disjunktive Normalform 75
 Entwurf 65, 69, 72
 Gatter 69
 Gene 65
 Kombinations-Verschaltung 60
 Minimierung 75
 Musterzuordnung 69

Neuronen 90
Parallelverschaltung 58
Proteine 65
Reihen-Verschaltung 58
Scheduling 179
Schleife 416
Schlüssel 314, 317
 öffentlicher 325
 privater 325
Schlüsselaustausch 320
Schlüssel-Schloss-Prinzip 334
Schmerz 169
Schnittstelle 145, 425
Schrifterkennung 365
Schwefel, Hans-Paul 436
Secure Shell 265
Selektion 428, 438
Sensor 155
Sequenzalignment 433
Sequenzierung 431
Server 264
Shamir, Adi 330
Shannon, Claude 302
Shotgun-Sequenzierung 431
Sicherheitsschicht 261
Sinneswahrnehmung 154, 159, 167
Skalierbarkeit 174
Skytale 313
Software 120
Software-Architektur 423, 424
Software Engineering 400, 423
Software-Entwicklung 402
Software-System 400
Software-Technik 400, 423
solid state disc 201
Southbridge 164
Speedup 172
Speicher 192
 assoziativer 233
 EEPROM 199
 Hauptspeicher 102
 nichtflüchtiger 196
 RAM 199

- Register 102
- ROM 199
- WORM 199
- Speicherelement 83
- Speicherhierarchie 205, 403
- Speicherkapazität 197
- Speichertechnologie 195, 227
- Spezifikation 423
- Spielbaum 369
- Spielen 361, 367
- Spiralmodell 426
- Sprache 362, 371, 420
 - formale 372
- Spracherkennung 371
- Spyware 412
- SSD (*solid state disc*) 201
- SSH (*Secure Shell*) 265
- Stammbaum 218, 429
- Stammbaum-Rekonstruktion 432
- Standardisierung 403
- Stapel 210
- statische Finitheit 349
- Statusleitung 108, 112
- Steganografie 308
- Stellenwertsystem 47
 - Dualsystem 48
- Stephenson, Archibald M. 8
- Stephenson-Addierer 8
- Steuerleitung 108
- Steuerleitungsbelegung 110
- Steuerwerk 99, 115, 128, 147
- Stoffwechselwege 133
- Strategie
 - gierige 354
- String 416
- Stromkreis 44
 - Aktivitätsmuster 46
 - Musterzuordnung 54
 - Zustand 45
- strukturierte Programmierung 419
- Suche
 - Baum 216
 - sortiertes Feld 213

- Suchproblem 345
- Suchraum 345
- suprachiasmatischer Kern 164
- Switch 250
- symmetrische Verschlüsselung 329
- Synapse 89
- synchroner Datenaustausch 157
- Syntaxbaum 373
- Synthetische Biologie 75
- System 145

T

- Takt 103, 110, 117, 157
- Taktfrequenz 118
- Taktgeber 157, 164
- Tarnung 312
- Taschenrechner 62, 68, 72
 - mit Speicher 76, 84
- TCP (*Transmission Control Protocol*) 262
- Teile und herrsche 357
- Telnet 265
- Terminalsymbol 373
- Testen 424
- Textdarstellung 290, 294
- Tic-Tac-Toe 370
- Top-Level-Domain 258
- Torvalds, Linus 185
- Transistor 13, 56
- Transmission Control Protocol* 262
- Transport-Protokoll 262
- Transportschicht 261
- trap door function* 322
- trial and error* 350
- Trojanisches Pferd 412
- Turing, Alan 15, 319, 349, 360, 381
- Turing-Award 26
- Turing-Maschine 16, 349
 - Addition 19, 21
 - Arbeitsweise 17
 - deterministische 392
 - nichtdeterministische 392
 - universelle 32

Turing-Test 360

Türme von Hanoi 357

U

Überschalldüse 436

Uhrenarithmetik 323

Uhrenfunktion 323, 327

UND-Gatter 69

Unentscheidbarkeit 384

Unicode 290

UNIX 185

Unterbrechung 165, 176

Maskierung 167

Unterbrechungsbehandlung 165

Unterprogramm 161, 416

Uruk 263

V

Variable 417

Variation 428, 438

vegetatives Nervensystem 161, 342

Vektorgrafik 294

Verarbeitungseinheit

zentrale 96

Verhalten 143, 341, 360

Verhaltensmuster 342

Vermittlungsschicht 261

Verschlüsselung 313

asymmetrische 329

Schlüssel 314

symmetrische 329

Versuch und Irrtum 350, 386, 392, 438

Verzweige und begrenze 353

Vigenère-Chiffre 315

Virus 85, 408

von Neumann, John 149

Von-Neumann-Architektur 148, 150

W

Wachstum 388

Wachstumsordnung 223, 387

Wahrheitstabelle 69, 117, 131

Warteschlange 221

Wasserfallmodell 426

Wasserzeichen 309

Web 266

Winkeralphabet 239

wireless local area networks 248

WLAN (*wireless local area networks*) 248

World Wide Web 266

WORM (*write once read many*) 199

Wort 372

write once read many 199

Wurm 410

WWW (*World Wide Web*) 266

Z

Zahlendarstellung 49, 289

Zählproblem 345

Zeichen 287

Zeichenvorrat 287

Zeiger 219

zentrale Verarbeitungseinheit 96

Zertifikat 331

Zertifizierungsstelle 331

Zwischenhirn 167



Willkommen zu den Springer Alerts

Jetzt
anmelden!

- Unser Neuerscheinungs-Service für Sie:
aktuell *** kostenlos *** passgenau *** flexibel

Springer veröffentlicht mehr als 5.500 wissenschaftliche Bücher jährlich in gedruckter Form. Mehr als 2.200 englischsprachige Zeitschriften und mehr als 120.000 eBooks und Referenzwerke sind auf unserer Online Plattform SpringerLink verfügbar. Seit seiner Gründung 1842 arbeitet Springer weltweit mit den hervorragendsten und anerkanntesten Wissenschaftlern zusammen, eine Partnerschaft, die auf Offenheit und gegenseitigem Vertrauen beruht.

Die SpringerAlerts sind der beste Weg, um über Neuentwicklungen im eigenen Fachgebiet auf dem Laufenden zu sein. Sie sind der/die Erste, der/die über neu erschienene Bücher informiert ist oder das Inhaltsverzeichnis des neuesten Zeitschriftenheftes erhält. Unser Service ist kostenlos, schnell und vor allem flexibel. Passen Sie die SpringerAlerts genau an Ihre Interessen und Ihren Bedarf an, um nur diejenigen Information zu erhalten, die Sie wirklich benötigen.

Mehr Infos unter: springer.com/alert