Hardwin Jungclaussen

Kausale Informatik

Einführung in die Lehre vom aktiven sprachlichen Modellieren
durch Mensch und Computer

Dieses Buch widme ich dem Andenken an meine Freunde,
die Informatiker

Klaus Fritzsch
und
Eberhard Klett.

Geleitwort

Das vorliegende Buch “Kausale Informatik” stellt die Summa einer jahrzehntelangen Befassung des Autors mit der Informatik, insbesondere aus systemtheoretischer, aber auch aus physikalischer und philosophischer Sicht dar und “will den Weg zu einer kausalen Wissenschaft Informatik (in Analogie zur Physik) ebnen helfen”. Mit diesem Anspruch wendet es sich an einen breiten Leserkreis: Insider, Informatikinteressierte und Anwender, aber auch an Informatikstudenten. “Kausale Informatik” ist kein Textbuch oder Kompendium; der mathematische Apparat ist bewusst zurückgenommen eingesetzt, um den Ideengehalt, dargestellt auf der Basis meines Erachtens kreativer, aber gewöhnungsbedürftiger Begriffsbildungen, mehr hervortreten zu lassen.

Aus dieser Sicht werden die Hauptbereiche der Informatik (einschließlich Hardware) behandelt, ein gesonderter Abschnitt ist dabei der Künstlichen Intelligenz gewidmet. Den Abschluss bildet die Auseinandersetzung des Autors mit philosophischen und ethischen Problemen der Informatik und ihrer Anwendungsbereiche.

“Besinnungsbücher”, wie das vorliegende, sind für die in rasanter und inhomogener Entwicklung befindliche Wissenschaftsdisziplin Informatik sehr notwendig und von großem Nutzen. “Kausale Informatik” liefert neben einer Gesamtdarstellung der Informatik in diesem Sinne viele Denkanstöße, die zu Neuem führen können, und Ansatzpunkte für durchaus konträre Diskussionen, fordert auch den Widerspruch heraus. Das muss ein Buch dieser Art.

Erwin P. Stoschek

Vorwort

Was hast du aber, das du nicht empfangen hast.
1 Kor 4,7

Ich glaube, ich habe nie eine Gedankenbewegung erfunden, sondern sie wurde mir immer von jemand anderem gegeben. Ich habe sie nur sogleich leidenschaftlich zu meinem Klärungswerk aufgegriffen.
Ludwig Wittgenstein

Wittgensteins Satz trifft sicher auf jeden zu, der sich einem "Klärungswerk" verschreibt, sei es, weil er andere aufklären will, sei es, weil er, wie ich, zunächst einmal selber zur Klarheit kommen will. Darin, dass ich Wittgensteins Satz auf mich beziehe, liegt keine Anmaßung. Ich weiß, dass sich meine Überlegungen und Einsichten nicht an der Schärfe und Fundamentalität seiner Gedanken und Sätze messen können. Zudem muss ich in dem Zitat die Worte "sogleich leidenschaftlich" ersetzen durch "soweit ich es mit zähem Bemühen vermochte".

Antrieb meiner Bemühungen waren die Frage "Was ist Informatik?" und der Wunsch, ein in sich geschlossenes Gedanken- und Begriffsgebäude der Informatik zu erarbeiten. Ich habe den verschlungenen Weg zum Ziel nachträglich begradigt, um ihn für andere gangbar zu machen. Dabei ist ein Buch entstanden, das sich nicht so sehr durch markante, klärende Sätze, schon gar nicht durch mathematische Formulierungen auszeichnet, sondern eher durch wortreiche Erklärungen.

Wenn ich das Ergebnis meiner Arbeit nun der Öffentlichkeit vorlege, wende ich mich in erster Linie an junge, vorwärtsstrebende Menschen, die sich anschicken, als Informatiker die technische Basis der sich herausbildenden Informationsgesellschaft mitzugestalten und die das erlernte und zu erlernende Wissen in einen größeren Zusammenhang stellen möchten. Je tiefer diese jungen Informatiker um die naturwissenschaftlichen und auch um die philosophischen Wurzeln ihres Faches wissen, umso sicherer werden sie die Entwicklung in eine Richtung lenken, die zu einer funktionierenden und menschenwürdigen Gesellschaft führt. Darüber hinaus hoffe ich, dass auch erfahrene Informatiker, die sich für das wissenschaftliche und philosophische Umfeld, insbesondere für die physikalischen Grundlagen ihres Arbeitsgebietes interessiern, diese oder jene Anregung erhalten. Schließlich wendet sich das Buch an Leser, die sich einen Überblick über das Gesamtgebiet oder tiefere Einblicke in verborgene Zusammenhänge verschaffen wollen und die verstehen möchten, was sich hinter der Informatik und insbesondere hinter der künstlichen Intelligenz verbirgt, ohne Informatik studieren zu müssen. An diese Leser habe ich vor allem beim Schreiben der Kapitel 1 bis 7 und 11 sowie des Teils 3 und des Kapitels 21.4 gedacht. Das Buch setzt keine mathematischen Kenntnisse, wohl aber Lust zum Denken voraus.

Ich habe die Hoffnung, dass das Buch zu einem breiteren Verständnis dessen, was Informatik ist, und gleichzeitig zur Klärung des Inhalts der Wissenschaft Informatik beiträgt, selbst wenn nicht alle Steine des Anstoßes - wie Unschärfe in den Formulierungen und Schlussfolgerungen oder mangelnde Klarheit der Darlegungen - aus dem Wege geräumt sind, den das Buch durch das Dickicht von tausend Fragen und tausend Ideen dorthin bahnt, wo sich ein freierer Blick über das weite Feld der Informatik bietet. Natürlich sind auch andere Wege zu anderen “Standpunkten” möglich, die andere “Ansichten” der Informatik bieten.

Ich bedanke mich bei all denen, die meine Arbeit an dem Buch gefördert haben, sei es durch menschliche oder fachliche Unterstützung, durch Motivation, durch Diskussion oder durch Kritik. Mein erster Dank gilt meiner Frau. Sie hat mir Ruhe zu ungestörtem Arbeiten in der heimischen Studierstube geschenkt. Mein zweiter Dank gilt Prof. Hans Heinold, Prof. Dietrich Schubert und Dr. med. Klaus Weinert für die mir zuteil gewordene Unterstützung, die ich brauchte, um trotz aller Forderungen und auch Überforderungen, mit denen ich in meiner beruflichen Umwelt vor 1989 konfrontiert war, unbeirrt an meinem Anliegen arbeiten zu können und die Grundlage zu diesem Buche zu legen.

Für das kritische Lesen des gesamten Manuskripts und für viele wichtige Hinweise sage ich Dr. Manfred Bonitz, Dr. Horst Piehler, Prof. Dietrich Schubert und Prof. Erwin Stoschek meinen tiefen Dank.

Für die Durchsicht von Teilen des Manuskripts und wertvolle Hinweise bedanke ich mich bei Prof. Gerhard Banse, Bernd Dupal, Prof. Peter Fleissner, Prof. Hartmut Fritzsche, Dr. Dietbert Gütter, Dr. Claude-Joachim Hamann, Prof. Rolf Hebenstreit, Prof. Gerhard Hertz†, Prof. Dieter Jungmann, Prof. Alexander Leitsch, Dr. Wolfgang Oertel, Prof. Jörg Pflüger, Dr. Michael Posegga, Dr. Heinz Rötger, Dr. Wolfgang Schwarz, Prof. Rainer G. Spallek und Prof. Helmut Thiele.

Aus der großen Zahl derer, denen ich für Diskussionen oder für Unterstützung in dieser oder jener Form dankbar bin, möchte ich einige nennen: Prof. Gerhard Diener, Prof. Volkmar Dienhold, Prof. Klaus Fritzsch†, Dr. Eberhard Klett†, Dr. Anatolij Korneitschuk, Prof. Klaus Meißner, Dr. Wolfgang Oertel, Prof Hans-Georg Schöpf, Dr. Wolfgang Schubert und Prof. Paul Ziesche.

Mein besonderer Dank gilt Reinhart Schmidt für die ständige, selbstlose Unterstützung bei der Arbeit mit der Technik und bei der Erstellung des Schriftsatzes und der Bilder, sowie Bernd Dupal und Dr. Wolfgang Oertel für die Erstellung der Programme.

Alle, die zu nennen ich versäumt habe, bitte ich um Entschuldigung; sie mögen meiner Dankbarkeit gewiss sein. Ich habe nicht alle Bemerkungen und Einwände berücksichtigt. Für die Richtigkeit der Aussagen des Buches trage ich die alleinige Verantwortung. Ich werde jedem dankbar sein, der mich auf Fehler hinweist.

Schließlich möchte ich mich beim Deutschen Universitäts-Verlag und bei Dr. Reinald Klockenbusch bedanken, der in einem Gespräch betreffs des Buches im

Jahre 1991 die Grundrichtung mit sinngemäß folgenden Worten vorgezeichnet hat: Es ist das Allgemeine mit dem Detail in ausgewogenem Verhältnis zu verbinden. Ich bedanke mich für die mir entgegengebrachte Nachsicht und Geduld.

Ich bin dankbar für die Freude, die mir die Arbeit an dem Buch bereitet hat.

Dresden, August 2000 Hardwin Jungclaussen

Inhaltsverzeichnis

Geleitwort **VII**

Vorwort **IX**

Inhaltsverzeichnis **XIII**

Symbole und Abkürzungen **XIX**

Einleitung **1**
Anliegen und Besonderheiten des Buches 1
Hinweise zum Lesen des Buches 8

Teil 1 Grundbegriffe und Grundideen

Zusammenfassung der Kapitel 1 bis 3 **11**

1 Codierung, Evolution und Information **13**

2 Gedanke und Sprache **19**

3 Informatik - Lehre vom aktiven sprachlichen Modellieren **25**
3.1 Modellklassen. Begriffsbestimmung der Informatik 25
3.2 Fundamente der Informatik 28

4 Analoges und digitales Modellieren **31**
Zusammenfassung 31
4.1 Messen und reelle Zahlen 32
4.2 Analoges Rechnen 35

5 Syntax und Semantik **43**
Zusammenfassung 43
5.1 Sprache und Evolution 44
5.2 Hierarchische Komponierung 47
5.3. Syntaxregeln 48
5.4 Externe, interne und formale Semantik 51
5.5 Begriffsbildung 55
5.6* Umcodierungseffizienz und syntaktische Information 61

6 Arbitrarität und Zirkularität **69**
Zusammenfassung 69
6.1 Arbitrarität des Artikulierens 69
6.2 Referenzielle Zirkularität 70
6.3 Operationale Zirkularität 72

7 Evolution der Intelligenz **75**
Zusammenfassung 75

7.1 Deduktive, assoziative und intuitive Intelligenz. 76
7.2 Beschriftung der tabula rasa. Algorithmenbegriff 81

8 Formale Grundbegriffe und Methoden 87
Zusammenfassung 87
8.1 Uniforme Systembeschreibung (USB) 88
8.2 Kausaldiskrete Prozessbeschreibung 95
8.2.1 Reale und sprachliche Operatoren 95
8.2.2 Petrinetze 101
8.2.3 Abstrakter Automat 105
8.2.4 Elementare kausaldiskrete Operatoren 108
8.2.5 Logisch-kausale Äquivalenz 111
8.3 Operation, Funktion, Prädikat, Berechenbarkeit 114
8.4* Universelle algorithmische Systeme 125
8.4.1 Problemstellung 125
8.4.2 Turingmaschine 127
8.4.3 Unbeschränkte Registermaschine (URM) 129
8.4.4 Markovalgorithmus 130
8.4.5 Rekursive Funktionen und USB-Funktionen 131
8.4.6 Einschub: Rekursives Berechnen 142
8.4.7 Lambda-Kalkül 146
8.5* CHURCH‘sche These und Kalkültransformation 151
8.6 Vier Grundideen des elektronischen Rechnens 158

Teil 2 Vom Bit zur Maschinensprache

9 Grundlagen der Komponierung informationeller Operatoren 167
Zusammenfassung 167
9.1 Statische und dynamische Codierung 168
9.2 Elementare informationelle Operatoren 170
9.2.1 Elementare boolesche Operatoren und die erste Grundidee des elektronischen Rechnens 170
9.2.2 Künstliche Neuronen 176
9.3* Ergänzung der formalen Methoden: Boolesche Algebra 178
9.4 Netzklassen 188
9.5 Ein-Bit-Speicher 197
9.6 Einschub: Magnetische und optische Speicherverfahren 199

10 Realisierungsprinzip zirkelfreier boolescher Netze 201
Zusammenfassung der Kapitel 10 bis 12 201
10.1 Die zweite Grundidee des elektronischen Rechnens 203
10.2 Zirkelfreie Schalternetze 205

11 Historischer Einschub: Entwicklung der begrifflichen und physikalischen Basis der Rechentechnik 207
11.1 Von der Syllogistik des Aristoteles zur Schaltalgebra 207
11.2 Rechnergenerationen 213

12 Technische zirkelfreie boolesche Netze 219
12.1 Codeumsetzer und elektronischer Festwertspeicher 219
12.2 Herstellung von Diodenmatrizen 225
12.3 Anwendungen von Diodenmatrizen 227
12.3.1 Elementare Hardware-Software-Schnittstelle 227
12.3.2 Multiplexer, Demultiplexer und Bus 228
12.3.3 Funktionsgeneratoren 234
12.3.4 Steuermatrix 234

13 Von der Kombinationsschaltung zum Von-Neumann-Rechner 237
Zusammenfassung 237
13.1 Die dritte Grundidee des elektronischen Rechnens 238
13.2 Elektronische Speicher 243
13.2.1 Register 243
13.2.2 Adressierbarer elektronischer Speicher 244
13.3* Einschub: Berechenbarkeits-Äquivalenzsatz 246
13.4 Taschenrechner 250
13.5 Prozessor 253
13.5.1 Idee des Prozessors und seiner Programmierung 253
13.5.2 Maschinensprache 257
13.5.3 Arbeitsweise der RALU 261
13.5.4 Sprungbefehl 266
13.5.5 Matrixsteuerwerk 269
13.5.6 Operatorenkomponierung mittels Firmware 270
13.5.7 Mikroprozessor 271
13.6 Von-Neumann-Rechner 273
13.7 Netzparadigma und imperatives Paradigma 274

Teil 3 Von der Maschinensprache zur künstlichen Intelligenz

14 Zwischenbilanz 283
Zusammenfassung 283
14.1 Probleme des Softwareweges zur KI 284
14.2 Drei Rollen des Computers als eines intelligenten Partners des Menschen 286

15 Lösen mathematischer Probleme 289
Zusammenfassung 289
15.1 Funktionales Programmieren 291

15.2 Imperatives Programmieren 294
15.3 Näherungsverfahren 299
15.4 Assembler, Binder, Lader 300
15.5 Semantische Lücke 303
15.6 Numerisches Rechnen 305
15.7 Programmierung von Operatorenhierarchien 309
15.8 Analytisches Rechnen 312
15.9 Bemerkung zur Programmübersetzung 320

16 Lösen nichtmathematischer Probleme 323
Zusammenfassung 323
16.1 Schlussfolgern 324
16.2 Wissensverarbeitung 335
16.3 Erfinden 345
16.4 Übersetzen und die vierte Grundidee des elektronischen Rechnens 353
16.5* Theorie formaler Sprachen 364

17 Unterhaltung 373
Zusammenfassung 373
17.1 Der Computer als Gesprächspartner 374
17.2 Der Computer als Unterhalter 381
17.3 Der Computer als Schachpartner 385

18 Evolution der Programmiersprachen 399
Zusammenfassung 399
18.1 Der Flaschenhals des linearsprachlichen Modellierens 400
18.2 Semantische Verdichtung 407
18.3 Technisches Semantikproblem und Programmierparadigmen . . . 416

Teil 4 Vertiefende Ergänzungen

19 Ergänzungen zum Problemlösungsweg der Rechentechnik 429
Zusammenfassung 429
19.1 Vorbemerkung 430
19.2 Hardwarearchitektur unterhalb der Prozessorebene 431
19.2.1 Befehlssatz und Speicherhierarchie 431
19.2.2 Pipelining und Vektorrechner. 437
19.2.3 ALU-Array. Simulation von Nahwirkungen 440
19.3 Hardwarearchitektur oberhalb der Prozessorebene 444
19.4 Architektur kausaldiskreter Systeme 452
19.5 Betriebssystem 461
19.5.1 Anwendungsprogramme und Organisationsprogramme 461
19.5.2 Prozessbegriff und geteilte Nutzung von Hardwareoperatoren . 463
19.5.3 Geteilte Nutzung von Speicherplätzen, Daten und Programmen 470

19.5.4 Verteilte Systeme 472
19.5.5 Systemrufe und geschützte Prozesskomponierung 480

20 Programmbeispiele 485
Zusammenfassung 485
20.1 Imperative Programme 486
20.2 CommonLisp-Programme 488
20.2.1 Funktionale Programme 488
20.2.2 Imperatives Programm 491
20.2.3 Objektorientiertes Programm 492
20.2.4 Logisches Programm 494
20.3 Objektorientiertes Programm in Borland-Pascal 499
20.4 Netzprogrammierung und Software-Lebenszyklus 515

21 Komplexität 519
Zusammenfassung der Kapitel 21 und 22 519
21.1 Zum Begriff der Komplexität 520
21.2* Berechnungskomplexität 524
21.3 Modellierung komplexer Systeme und Prozesse 528
21.3.1 Strukturelle und nichtlineare Komplexität 528
21.3.2* Fuzzy-Kalkül 534
21.4 Offene Fragen und die Komplexität des Denkens 545

22 Resümee und Perspektiven 557

Schlusswort 563
Zur gesellschaftlichen Bedeutung der Informatik 563

Glossar 577

Literatur 597

Namen- und Sachverzeichnis 607

Symbole und Abkürzungen

Symbole

- ^ (Dach) 1. (in Formeln): Potenzoperator; 2. (im Glossar vor einem Begriff): zu lesen als “siehe unter...”
- Symbole der booleschen Algebra siehe Bild 9.1.
- Abkürzungen und Symbole der USB-Methode siehe die Bilder 8.1 und 8.2.
- Abkürzungen für die Bausteine von Prozessoren und Computern siehe die Bilder 13.7. und 19.1.
- Maßeinheiten siehe Bild 19.1

Abkürzungen

AC	Akkumulator
ADR-Matrix	Adressiermatrix
ALU	arithmetisch-logische Einheit
aop	Arbeitsoperator
BNF	Backus-Naur-Form
BR	Befehlsregister
BZ	Befehlszähler
CD	Compact Disk
CD-RW	ReWritable (wiederholt beschreibbare) CD
Computer-IV	Informationsverarbeitung durch den Computer
CPU	Central Processing Unit
DIN	Deutsches Institut für Normung
DNF	disjunktive Normalform
DR	Datenregister
DRAM	Dynamischer RAM
DVD	Digital Versatile Disk
E/A	Eingabe/Ausgabe
EPROM	wiederholt programmierbarer ROM
FIFO	First In First Out
HK	Halbkommutator
HS	Hauptspeicher
Human-IV	Informationsverarbeitung durch den Menschen
i.d.R.	in der Regel
i.e.S.	im engen Sinne
INC	Inkrementierer
IV	Informationsverarbeitung
IV-System	Information verarbeitendes System, informationelles System
i.w.S.	im weiten Sinne
K	1.Kombinationsschaltung; 2. Kommutator

KDNF	kanonische disjunktive Normalform
KKNF	kanonische konjunktive Normalform
KR-Netz	Netz aus Kombinationsschaltungen und Registern
ld	Duallogarithmus (Logarithmus zur Basis 2)
LSI	Large Scale Integration
LIFO	Last In First Out
MByte	Megabyte, $2^{20} \approx 10^6$ Byte
OC	Operationscode
od	Operand
on	Operation
ON	Operatorennetz
op	Operator
PAL	Programmable Array Logic
PAP	Programmablaufplan
PLA	Programmable Logic Array
PROM	programmierbarer ROM
PS-Netz	Netz aus Prozessoren und Speichern
R	Register (in Kap.10 Widerstand)
RALU	ALU mit ihren Arbeitsregistern
RAM	Random Access Memory, Schreib-Lese-Speicher
ROM	Read Only Memory, Lesespeicher
S	Sammelweiche (in Kap.10 Schalter)
sop	Steueroperator
SPEI-Matrix	Speichermatrix
TNS	Transport Nach dem Speicher (vom AC)
TVS	Transport Vom Speicher (zum AC)
USB	Uniforme Systembeschreibung
VLSI	Very Large Scale Integration
Z	Zweigeweiche

Einleitung

Anliegen und Besonderheiten des Buches

Seid fruchtbar und mehret euch und füllet die Erde und machet sie euch untertan.

1.Mose 1, 28

Denn das wissenschaftliche Denken verlangt nun einmal nach Kausalität, insofern ist wissenschaftliches Denken gleichbedeutend mit kausalem Denken, und das letzte Ziel einer jeden Wissenschaft besteht in der vollständigen Durchführung der kausalen Betrachtungsweise. 1

MAX PLANCK[1]

Die Gesichter und Hände der Mitmenschen, ein Stück Himmel und vielleicht noch eine Zimmerpflanze oder ein Blumenstrauß - das ist so ziemlich das einzige, was ein Großstädter an Natürlichem in seiner Umwelt zu sehen bekommt. Aus der natürlichen Umwelt ist eine künstliche geworden. Das bedeutet nicht, dass die heutigen Menschen die Welt besser beherrschen und die Zukunft besser gestalten als die Menschen vor 1000 oder 10000 Jahren, denn sie können die Wirkungen ihrer eigenen Ideen und Errungenschaften nicht voraussehen. So wie man die Auswirkungen der Dampfmaschine auf die menschliche Gesellschaft nicht voraussehen konnte, so können wir Heutigen die Auswirkungen des Computers nicht voraussehen.

Die Hilflosigkeit vor der eigenen technischen Machtfülle kennzeichnet die gegenwärtige Situation der Menschheit. Sie ist die Fortsetzung der Hilflosigkeit des primitiven Menschen der Natur gegenüber, und in ihrer Hilflosigkeit brauchen die Menschen von heute genauso wie ihre Vorfahren einen Hoffnungsträger, einen Glauben, einen Gott, eine Religion. Das gilt sicher nicht für jeden einzelnen Zeitgenossen, wohl aber für die Menschheit insgesamt, die sich bedroht und ohnmächtig fühlt. Denn die Beherrschung der Welt liegt immer jenseits des Horizontes, dem die Menschheit seit Adam zustrebt gemäß dem eingangs zitierten biblischen Befehl. Es ist jedoch nicht der Befehl einer höheren Instanz, sondern ein innerer Befehl, es ist das Produkt der genetischen und der Motor der kulturellen Evolution.

Im Spannungsfeld zwischen Technik und Psyche läuft eine atemberaubende Entwicklung ab, in der die Informatik ständig an Bedeutung gewinnt. Die Welt wird

---

1 Zitat aus dem Vortrag “Kausalgesetz und Willensfreiheit”, abgedruckt in [Planck 90]

immer künstlicher, und das betrifft in zunehmendem Maße auch die sprachliche Kommunikation. Immer mehr Informationen, die auf den Menschen einstürmen, werden von Maschinen "hergestellt", und zwar werden sie nicht nur maschinell codiert, sondern auch artikuliert, d.h. gedruckt oder gesprochen, sodass der Eindruck entstehen kann, nicht nur die Zeichen, sondern auch ihre Bedeutung, die der Mensch in die Zeichen hineinlegt, seien vom Computer produziert.

Der Computer, das Produkt menschlicher Erfindungsgabe, verändert die Welt und verändert unser Weltbild, ohne dass wir wissen, wohin die Reise geht und ohne dass wir die Entwicklung nach eigenem Gutdünken, nach eigenem Wissen und Gewissen steuern könnten. Die Folge ist ein zunehmender Druck in Richtung einer organisatorischen und sittlichen Erneuerung der Gesellschaft und speziell in Richtung einer notwendigen Anpassung an das eigene Kind, den Computer, und umgekehrt der Computertechnik an den Menschen, womit den Informatikern eine besondere Verantwortung auferlegt ist.

Die Informatik befindet sich ihrer Jugend gemäß gegenwärtig noch in der expansiven Phase des Erkundens, Erfindens und Sammelns. Die Entwicklung schreitet divergent in zahllosen Richtungen voran, wobei es nicht ausbleiben kann, dass die rechte Hand oft nicht weiß, was die linke tut. Der Evolutionsdruck in Richtung nachträglicher Konvergenz der divergierenden Zweige kommt erst in Ansätzen zur Wirkung, und der Übergang von der differenzierenden zur generalisierenden Phase, den jede Wissenschaft im Großen ebenso wie jede Abstraktion im Kleinen durchläuft, ist Aufgabe der Zukunft. Es ist also noch ein weiter Weg zurückzulegen, ehe die Informatik die Geschlossenheit beispielsweise der Physik erreicht haben und zu einer fundamentalen Wissenschaft geworden sein wird. Im Gegensatz zu manchen meiner Fachkollegen glaube ich, dass am Ende des Weges eine "*exakte Natur-Wissenschaft* Informatik" stehen wird, wobei unter "Natur" alles zu verstehen ist, was uns umgibt und was wir wahrnehmen bzw. direkt oder indirekt beobachten können. Angesichts der immer durchgreifenderen Beeinflussung der Natur durch den Menschen erweitert sich der Gegenstand der "Naturwissenschaften" zunehmend vom Natürlichen zum Wahrnehmbaren oder Beobachtbaren, das Künstliche eingeschlossen. Man denke nur an die "Naturwissenschaft Chemie", die sich mit Stoffen beschäftigt, die in der unberührten Natur nicht vorkommen. Die *Naturwissenschaft* als Wissenschaft vom Naturgegebenben hat sich fast unbemerkt zur *Wissenschaft des Beobachtbaren* erweitert.

Den Weg der Informatik zu einer exakten "Naturwissenschaft" (in dem genannten erweiterten Sinne) ebnen zu helfen, ist ein Anliegen des Buches. Es bemüht sich um eine einheitliche Sicht auf die verschiedenen Teilgebiete der Informatik und will das "Verständnis im Überblick" mit dem "Verständnis des Details" verbinden. Die Einheitlichkeit der Sicht spiegelt sich in der Einheitlichkeit des begrifflichen Apparates wider, mit dessen Hilfe sich sowohl die Hardware (die gerätetechnische Basis) als auch die Software (der programmtechnische Überbau) informationeller Systeme, darüber hinaus aber auch die Strukturen von Fertigungssystemen und der Ablauf von

Fertigungsprozessen, beispielsweise der Automobilherstellung, in einheitlicher Weise beschreiben lassen.

Angesichts der thematischen Breite des Buches verbietet sich das Eingehen auf tiefere Details. Einige speziellere rechentechnische Probleme sind aus dem Gedankengang vom Bit zur künstlichen Intelligenz, der sich durch die Teile 2 und 3 zieht, herausgelöst und in Teil 4 zusammengefasst. Das weite Feld der Anwendung der Rechentechnik wird nur hier und da in Form von Beispielen berührt. Ausführliche Darstellungen der verschiedenen Gebiete der Informatik findet der Leser in der angegebenen Literatur. Die Literaturauswahl ist bewusst klein gehalten. Sie beschränkt sich im Wesentlichen auf deutschsprachige Monographien mit umfangreichen Literaturangaben sowie auf einige moderne Lehrbücher und Lexika.

Die Begriffe, in denen ich meine Gedanken zur Informatik in diesem Buch niedergelegt habe, und speziell die in Kap.8 eingeführte Methode der *uniformen Systembeschreibung* (USB) sind das Ergebnis einer kritischen Betrachtung der Phänomene der Informationsverarbeitung mit den Augen eines Naturwissenschaftlers, dessen Denken ursprünglich nicht durch die Informatik, sondern durch die Physik geprägt worden ist. Angestrebt wird ein System von Begriffen, das zunächst auf Sinnfälligkeit ausgerichtet ist, das aber eine Schärfung der Begriffe bis zu mathematischer Strenge erlaubt, die allerdings in dem Buch nicht erreicht wird. Das Ziel des Buches besteht also nicht darin, eine einheitliche *formale Theorie* der Informatik vorzulegen, sondern eher darin, die gedanklichen und begrifflichen Grundlagen für eine solche Theorie zu erarbeiten.

Der in Teil 1 eingeführte Begriffsapparat wird in den nachfolgenden Teilen auf die *Prozessorinformatik* angewendet, d.h. auf die Lehre von der traditionellen Informationsverarbeitung, die sich des klassischen Prozessors bedient. Die Anwendung auf die *technische Neuroinformatik*, d.h. auf die Lehre von der Informationsverarbeitung durch künstliche neuronale Netze, wird lediglich angedeutet. Das moderne Gebiet der *Quanteninformatik* wird nicht behandelt[2].

Die Bemühungen um einen einheitlichen Begriffsapparat führen hier und da zur Einführung unkonventioneller Begriffe, zur Verwendung unkonventioneller Bezeichnungen für gängige Begriffe oder zur Verwendung gängiger Wörter in unüblichem Sinne. Letzteres trifft z.B. für den Intelligenzbegriff zu (s.u.). Das wohl auffallendste Beispiel unkonventioneller Begriffe ist das Begriffspaar **Realem - Idem** (Betonung jeweils auf der letzten Silbe, wie bei Morphem). Grob gesagt bezeichnet ein Realem-Idem-Paar ein reales Objekt und seine Widerspiegelung im Bewusstsein.

Der Grund für die Einführung dieses Begriffspaares ist der Wunsch, den wohl schwierigsten Begriff, mit dem die Informatiker zu tun haben, den Begriff der

2 Eine kompakte Darstellung der physikalischen Grundlagen des Quantencomputers und des gegenwärtigen Entwicklungsstandes findet der Leser in [Briegel 99].

Bedeutung, wie er im täglichen Umgang verwendet wird, von vornherein aus allen technischen und logischen Gedankengängen auszuklammern und ihm eine Sonderstellung zuzuweisen. Das dient der Klarheit der Darstellung. Denn der Sinn des Wortes "Bedeutung" ist in seinem vollen umgangssprachlichen Gehalt kaum zu erfassen. Das Buch akzentuiert die Schwierigkeit mit der Feststellung, dass Bedeutung (im umgangssprachlichen Sinne des Wortes) an *Bewusstsein* gekoppelt ist. Anders wird man dem Wort "Bedeutung" nicht gerecht. Genau genommen ist "Bedeutung" nämlich etwas zutiefst Individuelles. Ein reales Objekt, sei es ein Gegenstand oder ein gesprochenes oder gedrucktes Wort, hat für jeden, der es wahrnimmt bzw. interpretiert, *seine eigene*, d.h. seine personenspezifische *Bedeutung*. Diese personenspezifische und niemals vollständig mitteilbare Bedeutung wird vom Begriff *Idem* erfasst.

Darüber, was Bedeutung und was Bewusstsein "wirklich" sind, fällt in dem Buch kein Wort. Es handelt sich um unmittelbare Erfahrungen, denen jeder Menschen "sprachlos" gegenübersteht, die keine sprachliche Erklärung zulassen. Auf sie trifft der wittgensteinsche Imperativ zu: "*Wovon man nicht sprechen kann, darüber muss man schweigen*." An ihn halten wir uns. Dennoch muss der Begriff der *Bedeutung als persönlicher, in Worte nicht fassbarer Erfahrung* benannt werden, denn ohne ihn ist es nicht möglich über Sprache und Informatik nachzudenken und sich mitzuteilen.

Als Autor wusste ich mir keinen besseren Rat als die Erfindung eines neuen Wortes für diesen Begriff und eines zweiten Wortes für sein Pendant in der Realität, falls ein solches existiert. Nur so lassen sich Missverständnisse vermeiden, die allzu leicht dadurch entstehen, dass man meint, jeder verstünde unter einem bestimmten Wort genau dass Gleiche wie man selber. Die Gefahr, falsch verstanden zu werden ist besonders groß im Falle von Wörtern, die nicht einheitlich verwendet werden oder die bedeutungsmäßig überladen sind wie z.B die Wörter *Information* und *Semantik*.

Die Einführung neuer Wörter führt zuweilen zu einem unkonventionellen Sprachgebrauch, der für den Fachmann ohne Frage eine Unannehmlichkeit darstellt und wohlwollende Nachsicht und die Bereitschaft voraussetzt, ungewohnte Wörter in gewohnte zu übersetzen. Das Glossar am Ende des Buches soll das Übersetzen erleichtern.

Die Einführung des Begriffspaares *Realem* - *Idem* erfolgt gleich zu Beginn des Buches. Das Idem wird unter Verwendung des Wortes *Bewusstsein* eingeführt, also eines Begriffs, der außerhalb der konventionellen Informatik liegt und der für eine rein technische Darstellung nicht erforderlich ist. Denn es gibt keinen vernünftigen Grund, dem Computer Bewusstsein zuzubilligen. Dieser Sachverhalt hat zusammen mit der genannten Kopplung zwischen Bedeutung und Bewusstsein folgende Kon-
2 sequenz: *Computer verstehen und verarbeiten* ***keine "Bedeutung"***. Diesen Schluss nenne ich das ***Prinzip der Bedeutungsfreiheit*** *der Informationsverabeitung durch den Computer*. Das Wort "Prinzip" soll die "prinzipielle" Wichtigkeit des Schlusses unterstreichen. Das Prinzip der Bedeutungsfreiheit hat in der klassischen Logik sein Pendant im sog. *Extensionalitätsprinzip*.

Die Herangehensweise des Buches an die Probleme der Informatik zeichnet sich durch eine weitere konzeptionelle Besonderheit aus. Sie besteht darin, dass die Stellung jeder Frage und die Suche nach Antworten stets vom materiellen *Träger* der informationellen Prozesse ausgeht. Das gilt insbesondere für die Frage*: Zu welchen Leistungen ist ein Information verarbeitendes System fähig?* Im Falle der Informationsverarbeitung durch den Computer, kurz der *Computer-IV*, ist der Träger das als Computer bezeichnete *technische Gerät*. Im Falle der Informationsverabeitung durch den Menschen, kurz der *Human-IV*, ist der Träger das *Gehirn*. Die genannte konzeptionelle Besonderheit nenne ich das **Trägerprinzip**. Nach diesem Prinzip sind die Probleme der Informatik vom Träger her und nicht von der Mathematik, von der Logik oder von irgend einem anderen abstrakten Denksystem her anzugehen. Da das Trägerprinzip grundlegend für die Herangehensweise des Buches ist, sollte der Leser schon jetzt klar erkennen, worin das Prinzip genau besteht und welche Konsequenzen es hat. In einem ähnlichen, aber auf die Computer-IV beschränkten Sinne wird zuweilen vom *Prozessorprinzip* gesprochen. 3

Es würde naheliegen, unter dem Träger technischer informationeller Prozesse den Computer als Einheit von Hardware (gerätetechnischer Basis) und Software (programmtechnischem Überbau) zu verstehen. Die obige Frage lautet dann: *Zu welchen Leistungen kann eine gegebene Software eine gegebene Hardware befähigen?* In konkreten Fällen kann diese Frage von großer Wichtigkeit sein. Es ist aber nicht diejenige Frage, der unser Interesse gilt. Unsere Frage lautet: *Zu welchen Leistungen kann eine gegebene Hardware "**im Prinzip**" befähigt werden?* Hier bedeutet "im Prinzip" soviel wie "durch beliebige Software" oder genauer "durch die Gesamtheit aller denkbaren Softwaresysteme", was im Grunde gleichbedeutend ist mit "ohne Berücksichtigung der Software".

Die Frage ist scheinbar sinnlos, denn ohne Software ist die Hardware zu nichts in der Lage, sie ist "tot". Dass die Frage trotzdem nicht sinnlos ist, zeigt folgender Vergleich. Wir ersetzen die Hardware durch ein Auto und die Software, die das Verhalten der Hardware *steuert*, durch einen Fahrer, der das Verhalten des Autos *steuert*. Obwohl ein Auto ohne Fahrer "tot" ist, hat die Frage Sinn, was ein Auto "im Prinzip", d.h. ohne Berücksichtigung der Fähigkeiten des Fahrers, zu leisten vermag. Man kann z.B. nach der Höchstgeschwindigkeit oder nach dem Kraftstoffverbrauch fragen. Genauso kann hinsichtlich der Rechnerhardware fragen, welche Funktionen sie "im Prinzip" berechnen kann.

Die Forderung des Trägerprinzips stellt keine Notwendigkeit dar. Das Nachdenken der theoretischen Informatiker geht in der Regel nicht vom Trägerprinzip, sondern vom "*Denkprinzip*" aus, d.h. davon, *wie und was* der Mensch denkt. Das führt zwangsläufig zu der Frage, ob ein Computer "dasselbe und ebenso *denkt*" wie der Mensch, und zu der speziellen Frage, ob er dasselbe (dieselben Funktionen) *berechnen* kann wie der Mensch. Das Trägerprinzip stellt diese Frage *nicht*, sondern fragt, welche Hardware wie und was "denken" bzw. berechnen kann. Wir werden die Frage hinsichtlich einer Hardware stellen, die wir in Kap.13 entwickeln (nacher-

finden) werden. Es handelt sich um den sog. *Prozessorrechner*. Um Aussagen über seine "prinzipielle" Leistungsfähigkeit machen zu können, benötigen wir eine Methode, nach der sich sämtliche denkbaren Steuervorschriften für die Hardware (sog. Maschinenprogramme) erzeugen ("komponieren") lassen. Eine solche Methode wird in Kap.8 entwickelt und USB-Methode genannt (USB von Uniforme System-Beschreibung). Auf diesem Wege werden wir zu der bekannten Aussage gelangen, dass der Computer "nur" sog. *rekursive Funktionen* berechnen kann.

In enger Beziehung zum Trägerprinzip steht das **Realisierbarkeitsprinzip** der Computer-IV. Es schränkt den Bereich, über den nachgedacht wird, auf das *effektiv Machbare*, das *praktisch Realisierbare* ein. Das betrifft nicht nur den Träger, z.B. den Computer, sondern auch das, was der Träger "macht" (denkt, berechnet) bzw. machen soll. Gemäß diesem Prinzip werden nur solche informationellen Systeme betrachtet, die mit *endlichem* Aufwand gebaut werden können, und nur solche Verarbeitungsprozesse, die in *endlicher* Zeit "terminieren", d.h. ihr Ende erreichen.

Die drei genannten Prinzipien,

- das Prinzip der Bedeutungsfreiheit,
- das Trägerprinzip,
- das Realisierbarkeitsprinzip,

bestimmen die Herangehensweise des Buches an die Probleme nicht nur der Computertechnik, sondern der Informatik überhaupt.

Die Prinzipien ergeben sich aus dem Begriff der Informatik, wie er in Kap.3.1 definiert wird und im Untertitel des Buches verwendet ist. In Kap.3.1 wird die **Informatik** als *Lehre vom aktiven sprachlichen Modellieren* definiert. Ein Modell heißt *sprachlich*, wenn es das Original in codierter Form wiedergibt ("beschreibt"). Es heißt aktiv, wenn der Träger (z.B. der Computer) als zum Modell gehörig betrachtet wird. Dies ist ein Beispiel für die Verwendung eines gängigen Wortes in einer unüblichen Bedeutung. Die so definierte *Wissenschaft Informatik* besitzt viele Bezüge zur *Kognitionswissenschaft*, der Lehre von den menschlichen Methoden der Organisation und Verarbeitung von Wissen. Die Verwandtschaft beider Wissenschaften tritt besonders deutlich in den Kapiteln 2, 5, 7, im gesamten Teil 3 sowie in Kap.21 zutage. Der Gegenstand der Kognitionswissenschaft überdeckt sich weitgehend mit dem der Human-IV (s.o).

Ein anderes Beispiel für die Verwendung eines gängigen Wortes in einer unüblichen Bedeutung ist das Wort "Intelligenz". In diesem Buch wird mit **Intelligenz** die *Fähigkeit zum sprachlichen Modellieren* bezeichnet, und das Computermodell dieser menschlichen Fähigkeit, m.a.W. die simulierte natürliche Intelligenz, wird **künstliche Intelligenz** genannt.

Als Originale, also als Objekte des sprachlichen Modellierens, werden i.d.R. nur Ausschnitte der Realität, des "Wirklichen" betrachtet. Das stellt insofern eine Beschränkung dar, als der Mensch auch das "Unwirkliche" denken kann. Er kann beispielsweise bewusst oder unbewusst etwas Sinnloses denken und auch artikulieren. Ferner kann er sein eigenes Denken modellieren. Sein Denken kann zu endlosen

Schlussfolgerungsketten oder zu Antinomien führen. All diese Möglichkeiten sind nicht der eigentliche Gegenstand des Buches und werden nur am Rande erwähnt.

Diese Beschränkung bedeutet einen Verzicht auf viele tiefgründige und faszinierende Fragen, auf die man stößt, wenn man über das *Denken* nachdenkt. Doch kann man auf die Behandlung dieser Fragen verzichten, wenn man verstehen will, wie ein Computer funktioniert und was er kann. Wer sich für die hier ausgegrenzten Probleme interessiert, findet eine ausführliche und gleichzeitig kurzweilige Darstellung mit vielen Beispielen in [Hofstadter 85].

Auf ein anderes Buch möchte ich besonders hinweisen, weil es gewissermaßen ein Pendant zu diesem Buch darstellt. Es handelt sich um das Buch "Nichtphysikalische Grundlagen der Informationstechnik" von Siegfried Wendt [Wendt 89]. Dort wird - ebenso wie in diesem Buch - versucht, ein einheitliches Gebäude der Informatik zu "konstruieren". Ausgangspunkt ist jedoch nicht das Trägerprinzip; das Fundament der Konstruktion ist nicht das Stofflich-physikalische, nicht das *Naturnotwendige*, sondern das *Logische*, das *Denknotwendige*. Im Titel des vorliegenden Buches kennzeichnet das Wort "kausal" die Herangehensweise im Sinne des Trägerprinzips. Das bedeutet, dass die Informatik nicht als Gebäude logischer, sondern kausaler Beziehungen entwickelt wird und dass die elementaren Bausteine des Gebäudes nicht Zeichen oder Code, sondern codierende Zustände des Trägers sind. "*Kausale* Informatik" ist als Pendant zu einer "nichtkausalen", d.h. zu einer rein logischen Informatik zu verstehen, keinesfalls als Gegenentwurf zu einer "akausalen" Informatik , was keinen Sinn ergäbe. In dem Wort "kausal" findet folgender Sachverhalt seinen kurzen und treffenden Ausdruck. Sprachliches Modellieren beruht letzten Endes auf Abbildungen kausaler Ereignisketten im Original in kausale Ereignisketten im Modellträger (Computer oder Gehirn) sowie in Abbildungen zwischen logischen Schlussketten über das Original und kausalen Ereignisketten im Modellträger.

Die kausale Herangehensweise liegt dem Gedankengang zugrunde, der das gesamte Buch durchzieht. Sein Ziel ist die kausale Beschreibung des sprachlichen Modellierens. Am Anfang und am Ende des Buches steht jeweils ein Zitat von MAX PLANCK; das eine ist diesem Abschnitt, das andere dem Kapitel 22 als Motto vorangestellt. Im Sinne dieser Zitate artikuliert der Titel des Buches das *notwendige*, aber *unerreichbare* Ziel der "Wissenschaft Informatik". Das Adjektiv "kausal" im Buchtitel ist im Sinne des Planck-Zitats als Forderung nach "*der vollständigen Durchführung der kausalen Betrachtungsweise*" vom physikalischen Modellträger bis zum sprachlichen Modell zu verstehen. Freilich können wir die Forderung nur hinsichtlich der maschinellen Informationsverarbeitung und der künstlichen Intelligenz, nicht aber hinsichtlich der natürlichen Intelligenz erfüllen.

Die Beschränkung auf das *Machbare* bedeutet nicht, dass ich ein reiner Pragmatiker bin. Vielmehr sind es gerade die tieferen menschlichen, gesellschaftlichen und philosophischen Probleme, in welche die Informatik involviert ist, die mich veranlasst haben, meinen Weg zur Informatik und zur künstlichen Intelligenz allgemein-

verständlich und dennoch detailliert darzustellen. Doch bilden diese Probleme nicht den Inhalt des Buches. Auf sie gehe ich sehr kurz im Schlusswort ein.

Ich habe das Buch im Vertrauen auf die Vernunft des Menschen geschrieben, die ihn befähigt, seiner Hilflosigkeit vor der eigenen technischen Machtfülle Herr zu werden.

Hinweise zum Lesen des Buches

Das Lesen, sowohl das Überfliegen als auch das Arbeiten mit dem Buch, wird durch Zusammenfassungen vor den Kapiteln, durch ein Glossar am Ende des Buches und durch unterschiedlichen Druck unterstützt. Neu eingeführte Begriffe und nichtnummerierte Abschnittsüberschriften sind fett gedruckt. Zitate, wichtige Textstellen und gedanklich zu betonende Wörter sind kursiv gedruckt. Kursiv gedruckte Textstellen von besonderer Bedeutung enthalten fettgedruckte Wörter. Programme sind in Schreibmaschinenschrift gedruckt. Sogenannte objektsprachliche Wörter (Wörter, *über* die etwas ausgesagt wird) werden durch Anführungsstriche oder durch Kursivdruck gekennzeichnet.

Das Buch enthält viele Querverweise. Das Verweisen erfolgt nicht über Seitenzahlen, sondern über spezielle *Verweiszahlen*. Textstellen, *auf* die verwiesen wird, sind kapitelweise durchnummeriert. An derjenigen Textstelle, *von* der aus verwiesen wird, ist die Verweiszahl in eckigen Klammern angegeben. Falls auf ein anderes Kapitel verwiesen wird, ist auch die Kapitelnummer angegeben, z.B. [8.12]. Die Textstelle, *auf* die verwiesen wird, findet der Leser über die betreffende Verweiszahl auf dem Rand neben der betreffenden Stelle.

Ich habe mich um eine Darstellung bemüht, die ein *durchgängiges* Lesen ohne Vor- und Zurückblättern ermöglicht, sodass die Verweise i.Allg. außer Acht gelassen werden können. Die Verweise sollen denjenigen helfen, die ein Kapitel ohne Kenntnis der vorangehenden Kapitel lesen möchten, aber auch denjenigen die das Buch nicht nur durchlesen, sondern als in sich geschlossenes Gedankengebäude erfassen wollen. Wenn es jedoch nur darum geht, sich an Definitionen zu erinnern, wird es oft einfacher sein, nicht die Verweise, sondern das Glossar zu benutzen.

Um sich ein Bild vom Inhalt des Buches zu machen, kann es ausreichen, die Zusammenfassungen vor den einzelnen Kapiteln zu lesen. Diejenigen Leser, welche sich für die mathematischen Grundlagen der Informatik weniger interessieren, können die Kapitel, die im Inhaltsverzeichnis mit einem Stern gekennzeichnet sind, ohne wesentlichen Verlust überspringen. Das Entsprechende gilt hinsichtlich der Kapitel 19 und 20 für Leser, die auf Ergänzungen zur Rechnerarchitektur, zu Betriebssystemen und zu Programmiersprachen verzichten wollen. Bei nichtdurchgängiger Lektüre des Buches kann das Glossar gute Dienste leisten.

Teil 1

Grundbegriffe und Grundideen

Zusammenfassung der Kapitel 1 bis 3

Ein relativ abgeschlossener Bewusstseinsinhalt, der im Denken als selbständige Einheit, als Objekt des Denkens auftritt, heißt *Idem*. Ein Ausschnitt der realen Welt, der sich im Bewusstsein eines Menschen widerspiegelt, dem also ein Idem entspricht, heißt *Realem*. Zeichenkörper, z.B. einzelne Zeichen, Zeichenketten oder Muster, die Ideme auslösen, also Bewusstseinsinhalte hervorrufen, heißen *Zeichenrealeme* im Gegensatz zu den *Urrealemen*, den Gegenständen selbst, die durch Zeichenrealeme beschrieben (sprachlich modelliert) werden. Das Zuordnen eines Zeichenkörpers zu einem Idem heißt *Codieren*. Das Zeichenrealem "codiert" das Idem. Darum wird es auch *Code* genannt. Die Zusammenfassung eines Realems mit dem zugeordneten Idem heißt *Information*, die Zusammenfassung eines Zeichenrealems mit dem zugeordneten Idem heißt *Zeicheninformation*. Ein Zeichenrealem ist ein Zeichenkörper, dem ein Artikulierer (Sender) eine Bedeutung (ein Idem) zugeordnet *hat* und dem ein Interpretierer (Empfänger) eine Bedeutung zuordnen *kann*.

Sprache ist Mittel der Codierung und dient der sprachlichen, d.h. codierten Modellierung der Welt. Natürliche Sprachen sind Mittel der Informationsübertragung zwischen Menschen zum Zwecke ihrer Kooperation. Informationsübertragung setzt voraus, dass der Empfänger versteht, was der Sender meint, m.a.W. dass das Idem des Interpretierers mit dem Idem des Artikulierers in ausreichendem Maße zusammenfällt. *Sprechen* bzw. *Denken* ist extern codiertes (mitteilendes) bzw. extern nicht codiertes (nicht mitteilendes, sondern nur "gedachtes") sprachliches Modellieren. *Externes Codieren* und *Artikulieren* sind Synonyme. Die Fähigkeit zum sprachlichen Modellieren heißt *Intelligenz*.

Information ist das Elixier der intellektuellen und kulturellen Evolution. Unter *intellektueller* Evolution wird die Entwicklung der individuellen Denkfähigkeit eines Menschen verstanden, unter *kultureller* Evolution die Entwicklung der sprachlichen Modellierung der Welt durch eine *Kulturgemeinschaft* (über Generationen kooperierende Gemeinschaft von Menschen) und die darauf aufbauende technische, wissenschaftliche, künstlerische und weltanschauliche Entwicklung. Von Generation zu Generation wird Information (Erfahrung) weitergegeben und akkumuliert, sodass eine Entwicklung resultiert. Individuelle wie kulturelle Evolution sind wegen der Notwendigkeit der Informationsweitergabe an Codierung gebunden.

Ein *Symbol* ist eine Zeicheninformation, deren Zeichenkörper relativ kompakt, evtl. ein elementares Zeichen ist und deren Bedeutung für alle Mitglieder einer Kulturgemeinschaft durch Vereinbarung oder Gewohnheit einheitlich festgelegt ist (Additionszeichen, Kruzifix).

Die Informationsverarbeitung durch den Menschen kann auf zwei Ebenen untersucht werden, die als *symbolische* bzw. *subsymbolische* Betrachtungsebene bezeichnet werden, dabei ist der Wortteil "symbol" als Zeicheninformation im allgemeinen Sinne zu verstehen, also nicht unbedingt im speziellen Sinne von "Symbol". Auf der *symbolischen Ebene* wird jeder Code, jede Zeichenkette als interpretierbar angenom-

men; jeder Zeichenkörper ist ein Zeichenrealem. Für die *subsymbolische Ebene* gilt das nicht. Zustände des materiellen *Trägers* der Informationsverarbeitung werden nicht als *codierende* Zustände aufgefasst. "Informationsverarbeitung" ist reine Zeichenverarbeitung und erfolgt ausschließlich nach den Gesetzen der Physiologie des Gehirns. Das Interpretieren (Zuordnen von Bedeutung) ist eine von der Zeichenverarbeitung losgelöste, nachträgliche, bewusste Tätigkeit des Menschen. Auf der subsymbolischen bzw. symbolischen Ebene werden die Prozesse der Informationsverarbeitung aus der Sicht der Neurophysiologie bzw. der Psychologie untersucht. Informationsverarbeitung durch den Menschen lässt sich auf symbolischer wie auf subsymbolischer Ebene technisch simulieren. Als Träger der Simulationsprozesse (der technischen Informationsverarbeitung) dient der Computer.

Informatik ist die Wissenschaft und die Technik der aktiven (den Träger einschließenden) sprachlichen Modellierung. Es wird zwischen *biologischer* und *technischer Informatik* unterschieden, je nachdem, ob der Träger der informationellen Prozesse ein biologisches oder ein technisches System ist. Die biologische Informatik umfasst zwei Bereiche, die Lehre von der sprachlichen Modellierung der Welt mit Hilfe der "Sprache des Nervensystems" und die Lehre von der sprachlichen Modellierung des eigenen Organismus mit Hilfe der "Sprache des genetischen Systems. Dieses Buch behandelt die Probleme der Informatik auf der symbolischen Ebene.

1 Codierung, Evolution und Information

Im Anfang war das Wort.
Joh.1,1

Das Motto dieses Kapitels ist sicher einer der tiefsinnigsten Sätze der Weltliteratur. Das Bibelzitat hat die Form eines Aussagesatzes, es ist eine Feststellung, freilich eine sehr unterschiedlich und sehr weit auslegbare Feststellung. Faust versucht, das griechische "logos" nicht mit "Wort", sondern mit "Kraft" oder "Tat" zu übersetzen. Die folgenden Darlegungen werden uns zu einer anderen, in der Form sehr ähnlichen, inhaltlich aber viel vordergründigeren und eindeutigeren Feststellung führen, zu der Aussage: *Im Anfang war das Symbol*, oder etwas nüchterner: *Im Anfang war der Code.*

Um zu erklären, was mit dem Wort "Code" gemeint ist, führen wir zunächst den Begriff des Zeichenkörpers ein. *Ein **Zeichenkörper** ist entweder ein elementares Zeichen, z.B. ein Buchstabe, oder ein zusammengesetztes Zeichen (sog. Kompositzeichen), z.B. eine Kette oder ein Muster elementarer Zeichen*. Ein Zeichenkörper wird oft kurz *Zeichen* genannt. Nun vereinbaren wir: *Ein **Code** ist ein realer (akustischer oder visueller) oder ein gedachter Zeichenkörper, der einen bestimmten Bedeutungsinhalt verschlüsselt oder "codiert"*. Die Zuordnung heißt *Codierung*, doch wird dieses Wort in einer allgemeineren Bedeutung verwendet. *Als **Codierung** wird jede Zuordnung von Zeichenkörpern zu Bedeutungsinhalten oder von Zeichenkörpern zu Zeichenkörpern bezeichnet.* Im letzteren Falle sprechen wir auch von **Umcodierung**. Wenn einem Zeichenkörper ein Bedeutungsinhalt zugeordnet ist, muss dieser bei Umcodierung erhalten bleiben.

Wenn man in dem Bibelzitat "Wort" durch "Code" ersetzt und unter "Anfang" den Beginn der menschlichen Gesellschaft versteht, ergibt sich die fast selbstverständliche Feststellung, dass die Entwicklung der menschlichen Kultur mit der Verwendung von Zeichen als Bedeutungsträgern beginnt. Tatsächlich gilt eine viel allgemeinere Aussage, nämlich: *Evolution ist an Codierung gebunden*. Dabei ist das Wort Evolution in einem erweiterten, aber dennoch eingeschränkten Sinne zu verstehen; es umfasst die genetische, die intellektuelle und die kulturelle Evolution, nicht aber die kosmische Evolution insgesamt. Unter **intellektueller Evolution** soll die Entwicklung der individuellen Denkfähigkeit, speziell des begrifflichen Gebäudes verstanden werden, in welchem ein Mensch denkt, die Entwicklung seines Welt- und Selbstverständnisses. Unter **kultureller Evolution** soll die Entwicklung des Welt- und Selbstverständnisses einer Kulturgemeinschaft und der von ihr hervorgebrachten Artefakte verstanden werden (siehe in Bild 1.1 den Punkt 3 unter "Welt 1"). Genetische, intellektuelle und kulturelle Evolution können unter der Bezeichnung **codierende Evolution** zusammengefasst werden. Diese ist eine Komponente der

kosmischen Evolution. Im Weiteren ist unter Evolution stets codierende Evolution zu verstehen.

2 Evolution - man denke zunächst an die genetische - setzt voraus, dass die Subjekte, welche die Evolution tragen, sich in leicht veränderter, aber existenzfähiger Form reproduzieren können. Wenn die Reproduktion autonom erfolgt, also nicht unter der Leitung einer übergeordneten Instanz, muss eine Reproduktionsanleitung von Glied zu Glied der Evolutionskette weitergegeben werden. Jedes Glied muss seinen Nachfolger "bilden", ihm seine Form geben, d.h. ihn "informieren" im ursprünglichen Sinne des lateinischen Wortes `informare`. Es ist also gar nicht so fernliegend, dasjenige, was durch die Evolutionskette weitergereicht wird und den Nachfolger informiert, als *Information* zu bezeichnen[1]. Die weitergereichte Information muss mindestens drei Bedingungen erfüllen: 1. Sie muss die Reproduktion des Vorgängers ermöglichen; 2. sie muss verändert werden können, wobei die Veränderung so eng begrenzte Wirkungen haben muss, dass die Reproduktion (der Nachfolger) existenzfähig ist; 3. sie muss so kompakt sein, dass ihre Übergabe effektiv möglich ist.

3 Wenn der so charakterisierte Begriff "Information" genannt wird, dann hat das zunächst wenig mit dem Informationsbegriff zu tun, wie er im Zusammenhang mit der zwischenmenschlichen Kommunikation verwendet wird. Dort lässt er sich folgendermaßen bestimmen: *Eine **Information** ist ein Zeichenkörper zusammen mit der Bedeutung, die ihm der Artikulierer (Sender) oder ein Interpretierer (Empfänger) zugeordnet hat.* In mathematischer Schreibweise ist eine Information als Paar zu notieren:

Information = (Zeichenkörper, Bedeutung).

Man beachte, dass diese Definition nicht die Übereinstimmung der artikulierten mit der interpretierten Bedeutung verlangt, sodass Fehlinterpretationen möglich sind. Ferner beachte man, dass Artikulieren und Interpretieren Bewusstsein voraussetzen. Danach ist Information an Bewusstsein gekoppelt.

Wir sind zu zwei scheinbar sehr unterschiedlichen Informationsbegriffen gelangt, einem seiner Natur nach "*evolutiven*" und einem "*kommunikativen*". Analysiert man den kommunikativen Informationsbegriff hinsichtlich der drei Bedingungen, die von evolutiver Information erfüllt werden müssen, stellt man Folgendes fest. Kompaktheit ist eine wesentliche Forderung der sprachlichen Kommunikation und wird durch effektive *Codierung* erreicht, d.h. Bedeutungsinhalte - eventuell sogar sehr komplexe - werden durch kompakte Zeichenkörper (Wörter, Namen, Sätze) codiert (benannt). Das *Interpretieren*, also das Zuordnen einer Bedeutung zu einem Zeichenkörper, entspricht dem Reproduzieren gemäß Vorschrift. Und schließlich kann durch Verändern des Zeichenkörpers (der Zeichenkette) oder durch Fehlinterpretation die Bedeutung verändert werden.

---

1 Der Etymologie des Informationsbegriffs ist R. Capurro in [Capurro 78] nachgegangen.

Die Verwendung des Wortes Information für das, was durch eine Evolutionskette läuft, scheint also gerechtfertigt zu sein. Dabei darf aber ein wesentlicher Umstand nicht übersehen werden. Die Definition des kommunikativen Informationsbegriffs basiert darauf, dass die verwendeten Zeichenkörper Träger von Bedeutungsinhalten sind, die für die Kommunikationspartner durch Konvention (Vereinbarung, Gewöhnung) einheitlich festgelegt sind. Verwendet man diesen Informationsbegriff im Zusammenhang mit der genetischen Evolution, ergibt sich eine Ungereimtheit. Die Bedeutung des genetischen Codes ist keine Sache der Konvention und ist nicht an Bewusstsein gekoppelt.

Offensichtlich gehört der Begriff der *genetischen Information* einer begrifflichen Ebene an, die unterhalb derjenigen liegt, auf der kommunikative Information ausgetauscht wird; dabei bedeutet "unterhalb" soviel wie "näher an der organischen Basis, am stofflichen Träger". Es handelt sich um zwei Betrachtungsebenen, die als *symbolische* bzw. *subsymbolische* Ebene bezeichnet werden. Bevor wir sie charakterisieren, holen wir die Bestimmung des Symbolbegriffs nach, obwohl wir ihn wegen seiner Vieldeutigkeit nach Möglichkeit vermeiden werden.

Ein *Symbol* ist eine Information (präziser eine Zeicheninformation; siehe Kap.2 [2.3]) deren Zeichenkörper relativ kompakt, evtl. ein elementares Zeichen ist und deren Bedeutung für alle Mitglieder der Kulturgemeinschaft, in der das Symbol verwendet wird, durch Vereinbarung oder Gewohnheit einheitlich festgelegt ist, man denke beispielsweise an das mathematische Symbol der Addition oder an das religiöse Symbol des Kreuzes. Das Kreuz macht deutlich, wie weit der Bedeutungsumfang, wie "tief die Symbolik" eines Symbols sein kann.

In den Bezeichnungen der beiden Betrachtungsebenen als symbolische bzw. subsymbolische Ebene ist der Wortteil "symbol" als Zeicheninformation im allgemeinen Sinne zu verstehen, nicht unbedingt im speziellen Sinne von "Symbol". Wenn von Informationsverarbeitung auf subsymbolischer Ebene gesprochen wird, verbindet sich damit die Vorstellung, dass von Prozessen die Rede ist, die stets als physikalisch-chemische Prozesse verstanden werden können, welcher Art und wie komplex die Informationsverarbeitung aus "symbolischer" und damit aus psychologischer Sicht auch sei. In diesem Buch wird unter symbolischer bzw. subsymbolischer Betrachtungsebene Folgendes verstanden. 4

*Auf der **symbolischen Ebene** werden Zeichenkörper stets in Verbindung mit ihrer Bedeutung betrachtet; es werden also Informationen und ihre Verarbeitung betrachtet.* Die semantische Belegung der Zeichenkörper wird grundsätzlich als bereits erfolgt angenommen. Der Computer "weiß" von ihr nichts. *Auf der **subsymbolischen Ebene** werden primär keine Informationen betrachtet, sondern nur stoffliche Träger informationeller Prozesse.* Die Zuordnung von Bedeutungen zu bestimmten *Zuständen* im Träger wird nicht vorausgesetzt. "Informationsverarbeitung" ist reine Zeichenverarbeitung ("Zeichen" im Sinne von "Zustand ohne Bedeutung") und erfolgt ausschließlich nach den Gesetzen der Physiologie des Gehirns. Das Interpretieren (Zuordnen von Bedeutung) ist eine von der Zeichenverarbeitung losgelöste, sekun-

däre, evtl. bewusste Tätigkeit des Menschen. Es kann *nach*, aber auch *während* der Zeichenverarbeitung erfolgen, z.B. durch Gewöhnung oder Lernen[2]. Wenn eine Zuordnung getroffen wird, sodass die Zustände zu Codes werden, bedeutet das den "Aufstieg" auf die symbolische Ebene.

Im Zusammenhang mit dem genetischen Code wird zuweilen von Symbolen gesprochen. Das bedeutet ein stillschweigendes Überwechseln von der subsymbolischen Ebene der chemischen Verbindungen bzw. der Genotypen in die symbolische Ebene der Phänotypen, der in den "Symbolen codierten" Eigenschaften von Individuen. Die Zusammenfassung eines Genotyps mit seinem Phänotyp stellt eine genetische "Information" (im Sinne unserer Definition) dar, eine Erb-Information.

Informationsverarbeitung durch den Menschen ist im Grunde immer sprachliches Modellieren und kann auf beiden Betrachtungsebenen auf dem Computer simuliert werden. Die Entwicklung der Informatik und der *künstlichen Intelligenz* verlief über Jahrzehnte auf der symbolischen Ebene. Erst in jüngerer Zeit besinnt man sich auf die tieferliegende, subsymbolische Ebene. Das hat zur Aufspaltung der *technischen* Informatik, d.h. der Lehre von der Informationsverarbeitung durch technische Systeme, in zwei Teilgebiete geführt, die "*traditionelle Informatik*" und die "*technische Neuroinformatik*", deren Gegenstand künstliche neuronale Netze sind. Letztere finden in diesem Buch nur am Rande Erwähnung.

Die fundamentalste Frage, die man hinsichtlich der Information stellen kann, ist die nach ihrem Ursprung. Dahinter verbirgt sich die Frage nach dem Ursprung der Evolution. Einen exakten naturwissenschaftlichen Lösungsansatz dieser Frage in Bezug auf die *genetische* Evolution und damit auf den Ursprung des Lebens hat MANFRED EIGEN vorgeschlagen, wobei er explizit von *Entstehung von Information* spricht [Eigen 76].

Bezüglich der *intellektuellen* und *kulturellen* Evolution bewegt sich die Diskussion im Bereich der Philosophie. Man sucht noch nach den geeigneten Begriffen. Einen bedeutungsvollen Schritt auf diesem Wege hat KARL POPPER mit der Einführung seiner drei Welten getan [Popper 82] (siehe Bild 1.1[3]). Welt 1 umfasst die physischen Gegebenheiten, Gegenstände und Zustände (anorganische und organische, natürliche und künstliche), Welt 2 die Bewusstseinszustände (subjektives Wissen, Erfahrungen) und Welt 3 das objektivierte und materialisierte Wissen (kulturelles Erbe, theoretische Systeme). Der Bezug der drei Welten zu den Begriffen Evolution und Information ist offensichtlich. Welt 2 bzw. Welt 3 beinhaltet die in der intellektuellen bzw. kulturellen Evolution weitergereichte Information. Der Sprung von der symbolischen in die subsymbolische Ebene bei der Betrachtung der

---

2 In [Fleissner 98] wird anhand eines Computerspiels "Der blinde Springer" ein sehr durchsichtiger Mechanismus simuliert, nach welchem sich zwei Partner während ihrer Kooperation an bestimmte Zeichen (Symbole) gewöhnen, mit deren Hilfe sie sich nach einer Lernphase (Gewöhnungsphase) fehlerfrei verständigen können.

3 Entnommen aus [Eccles 88]

Welt 1 Physische Objekte und Zustände	**Welt 2** Bewusstseins-zustände	**Welt 3** Wissen im objektiven Sinn
1. Anorganische Materie und Energie des Kosmos 2. Biologie: Struktur und Wirkung aller lebenden Wesen, menschliches Gehirn 3. Artefakte: materielle Sub-strate mensch-licher Kreativität: Werkzeuge Maschinen, Bücher, Kunstwerke, Musik	Subjektive Erkenntnisse, Erfahrungen von: Wahrnehmungen, Denken, Emotionen zielgerichteten Strebungen, Erinnerungen Träumen schöpferischer Phantasie	1. Aufzeichnungen intellektueller Arbeiten: philosophische, theologische, wissenschaftl., geschichtliche literarische, künstlerische, technologische 2. Theoretische Systeme Wissenschaftliche Probleme, kritische Argumente

Bild 1.1 Die drei Welten Karl Poppers.

Informationsverarbeitung durch das menschliche Gehirn bedeutet den Übergang von der Welt 2 in die Welt 1 als Träger der intellektuellen Information.

Abschließend soll ein kurzer Vergleich unseres derzeitigen Erkenntnisstandes hinsichtlich der Informationsverarbeitung durch das genetische System einerseits und durch das Zentralnervensystems (ZNS) andrerseits angestellt werden. Der Kürze halber werden wir von DNS-Code bzw. ZNS-Code sprechen, womit die Realisierung der Codierung durch die DNS (Desoxyribonucleinsäure) bzw. das ZNS gemeint ist. Hinsichtlich der Einsichten in die Realisierung des Codes ist die genetische Forschung erheblich weiter fortgeschritten als die Gehirnforschung. Die Struktur der DNS ist praktisch vollständig als Folge der Zeichen des genetischen Alphabets erkannt.

Dieser Erfolg wird in den Medien zuweilen durch spektakuläre Meldungen wie z.B. “Der genetische Code ist geknackt” ziemlich irreführend dargestellt. Üblicherweise wird unter dem “Knacken” eines Codes, z.B. eines Geheimcodes, dessen Entschlüsselung verstanden, was bedeutet, dass die verschlüsselte Nachricht *verstan-*

den worden ist. Davon kann jedoch hinsichtlich des genetischen Codes nicht die Rede sein. Die Aufgabe der Genforscher ähnelt derjenigen eines Archäologen, der einen Text entschlüsseln will, der auf den Scherben einer zerbrochenen Tontafel in einer ihm unbekannten Sprache und in unbekannten Zeichen aufgeschrieben ist. Die einzelnen Zeichen hat der Archäologe bereits entziffert und die Serben hat er wie die Teile eines Puzzles weitgehend zusammengesetzt; aber die ungleich schwierigere Arbeit steht noch bevor, nämlich das Entschlüsseln, d.h. das "Übersetzen" des Textes in eine moderne Sprache. Für das Entschlüsseln werden die Genforscher voraussichtlich noch Jahrzehnte benötigen. Entschlüsseln bedeutet in diesem Falle das Verstehen der Funktion des genetischen Codes, d.h. die Einsicht in seine Wirkung auf die Merkmale des durch die DNS "beschriebenen" Individuums.

Über die Realisierung des ZNS-Codes gibt es zwar viele Hypothesen, doch sind die neurophysiologischen Befunde noch sehr mager. Die experimentelle Technik reicht gegenwärtig nicht aus, um ohne körperliche Eingriffe ausreichend scharf in das Gehirn hineinsehen zu können. Wir wissen nicht, wie ein Bewusstseinsinhalt, wie Denken und wie Gefühle intern "codiert" sind. Wir wissen nicht einmal, ob es in Analogie zum "DNS-Alphabet" überhaupt so etwas wie ein "ZNS-Alphabet" gibt. Angesichts der hohen Parallelität der Tätigkeit des Gehirns wird der Begriff des Alphabets wahrscheinlich wenig sinnvoll sein. Ferner ist anzunehmen, dass das Gehirn im Gegensatz zum genetischen System mit *dynamischer* Codierung arbeitet, worauf in Kap.9.1 eingegangen wird.

2 Gedanke und Sprache

Über die selbstverständlichsten Phänomene wissen wir wenig oder nichts.

MOSHÉ FELDENKRAIS

Dieser Satz von FELDENKRAIS trifft insbesondere für die Sprache zu. Was Sprache "wirklich" ist, wie sie "wirkt", wird uns noch lange beschäftigen. Nach den vorangehenden Überlegungen ist Sprache ein Ergebnis der Evolution. Dass die Evolution Sprache hervorbringt, wird verständlich, wenn man bedenkt, dass die Überlebenschancen einer Population, speziell einer menschlichen Gemeinschaft oder Gesellschaft, im Kampf mit der Natur und im Konkurrenzkampf mit anderen Populationen umso günstiger sind, je besser die Individuen der Population miteinander kooperieren. Kooperation aber setzt Kommunikation voraus. Im frühen Stadium der Entwicklung erfolgt sie mittels einfacher Signale, später mittels Sprache. *Natürliche Sprachen sind Mittel der Kommunikation zwischen Menschen zum Zwecke ihrer Kooperation.*

Folglich muss Sprache der Beschreibung der Umwelt dienen, in der die Kooperation stattfindet, die Beschreibung der Ziele und Handlungen der Kooperationspartner eingeschlossen. Mit Hilfe der Sprache muss sich die Welt (die Umwelt des Sprechenden) *modellieren* lassen. Insofern stellt Sprache ein Mittel der Modellierung dar, der *sprachlichen Modellierung*. *Ein* ***sprachliches Modell*** *ist eine durch Idealisierung vereinfachte Beschreibung eines Originals*. Eine Aussage über die Welt, ein *Aussagesatz,* ist demzufolge die primäre sprachliche Einheit. Das zeigt sich z.B. darin, dass Kinder ihre Muttersprache in Form ganzer Sätze lernen. Tatsächlich enthält jede abgeschlossene Äußerung die Bedeutung eines ganzen Satzes. Das können auch einzelne Wörter sein, wie z.B. die Entgegnung "Doch!" oder der Ausruf "Au!". 1

Für das sprechende Subjekt ist Sprechen das Artikulieren (das sprachliche Wiedergeben) von gedachten Sachverhalten. Objektiv betrachtet, aus der Sicht des Biologen, ist dieser Wiedergabeprozess sehr kompliziert. Subjektiv, introspektiv ist er dagegen so einfach, so selbstverständlich, dass man meinen könnte, Denken und Sprechen sei ein und dasselbe. Dass dies ein Irrtum ist, erkennt ein Mensch, welcher mehrere Sprachen perfekt beherrscht, spätestens dann, wenn er sich fragt oder gefragt wird, in welcher Sprache er denkt. Dass hier ein Missverständnis vorliegen muss, wird anhand der Frage deutlich, in welcher Sprache ein Dolmetscher denkt, der simultan übersetzt. Offensichtlich ist das Sprechen ein dem Denken nachgeschalteter Prozess, für den verschiedene sprachzpezifische Reflexsysteme "eingeschaltet" werden können. Das Entsprechende gilt auch für den entgegengesetzten Prozess, das Hören und Verstehen sprachlicher Äußerungen anderer.

Die Selbstverständlichkeit des Sprechens ist einer der Gründe für unser mangelndes Verständnis dessen, was wir mit *Denken* und *Sprechen* bezeichnen, und für die Schwierigkeiten, die mit der Bestimmung der Begriffe Gedanke und Sprache verbunden sind, sowie für die Missverständnisse, die solche Begriffe wie Bedeutung oder *Semantik* eines sprachlichen Ausdrucks hervorrufen können. Um von vornherein Irrtümer und Missverständnisse nach Möglichkeit auszuschließen, wollen wir uns von dem üblichen Sprachgebrauch lösen. Zu diesem Zweck führen wir zwei neue Wörter für scheinbar geläufige Inhalte ein, die Wörter *Realem* und *Idem* (mit der Betonung jeweils auf der letzten Silbe).

Ein Bewusstseinsinhalt oder Bewusstseinsausschnitt, der relativ abgeschlossen ist und der im Denken als selbständige Einheit fungiert, heißt ***Idem****. Ein Ausschnitt der realen Welt, der sich im Bewusstsein eines Menschen widerspiegelt, dem also ein Idem entspricht, heißt das dem Idem zugeordnete* ***Realem***.

Die Wörter *Idem* und *Realem* sind nach dem Vorbild solcher Wörter wie *Morphem* oder *Lexem* gebildet. Die Begriffsbestimmung setzt voraus, dass eine reale Außenwelt tatsächlich existiert und dass die Vorstellung eines realen Objektes ihre Entsprechung in der Außenwelt besitzt. Dass sich beides nicht beweisen lässt, ist Ursache für uralte philosophische Diskussionen.

Wir gehen davon aus, dass Gedanken und Sprachen, wie eingangs dargelegt, primär der Widerspiegelung einer unabhängig von uns existierenden Wirklichkeit dienen. In diesem Sinne verbindet unsere Definition ganz unbekümmert etwas Reelles, das Realem, mit etwas Ideellem, dem Idem, allerdings in unsymmetrischer Weise. Ein Realem setzt die Existenz eines zugeordneten Idems voraus. Das Umgekehrte gilt nicht. Die Phantasie und die Abstraktionsfähigkeit des Menschen ermöglichen die Bildung von Idemen, die keine Entsprechung in der realen Welt besitzen. Die Frage, ob es Dinge gibt, die sich nicht im Bewusstsein widerspiegeln können, für die es also keine Ideme geben kann, sind vom Menschen nicht zu beantworten, denn was nicht in seinem Bewusstsein existiert, das existiert für ihn überhaupt nicht. Dieser Schverhalt lässt sich auch so ausdrücken: *Die Welt, in der ein Mensch lebt, wird durch sein Bewusstsein* ***definiert***.

Eine besondere Klasse von Realemen sind die Zeichenrealeme. Ein Zeichenrealem ist die materielle Realisierung eines Codes. Hier ist eine Präzisierung des Begriffs “Code” (und ebenso der Begriffe “Name” und “Satz”), wie er im vorangehenden Kapitel [1.1] verwendet wurde, notwendig. Genau genommen muss nämlich zwischen abstraktem (gedachtem) und konkretem (materialisiertem) Code bzw. Namen oder Satz unterschieden werden.

Ein Code braucht einen Träger, um übertragen und verarbeitet werden zu können, er muss materialisiert sein, z.B. als Schwärzungsmuster, als Lochungsmuster, als Magnetisierungsmuster oder auch als Schwingungsmuster der Luftdichte (Schallwellen) u.ä.m. All diese Muster sind Realeme und gleichzeitig Zeichen. Darum
2 sprechen wir von **Zeichenrealemen**. Im Unterschied dazu nennen wir das ursprüngliche, also das bezeichnete Realem **Urrealem**. Analog ist zwischen dem ursprüngli-

chen Bewusstseinsinhalt, dem **Uridem** und seiner gedachten Bezeichnung, dem **Zeichenidem** zu unterscheiden. Den Uridemen bzw. Zeichenidemen entspricht in gewissem Sinne das erste bzw. zweite Signalsystem nach PAWLOV.

Mit den eingeführten vier Begriffen sind vier Objektklassen definiert, deren Beziehungen untereinander die Linguistik untersucht. In Bild 2.1 sind sie in ihrem Zusammenhang dargestellt. In Klammern sind einige linguistische Begriffe hinzugefügt, die unseren Begriffen etwa entsprechen, jedoch in der Literatur nicht einheitlich verwendet werden.

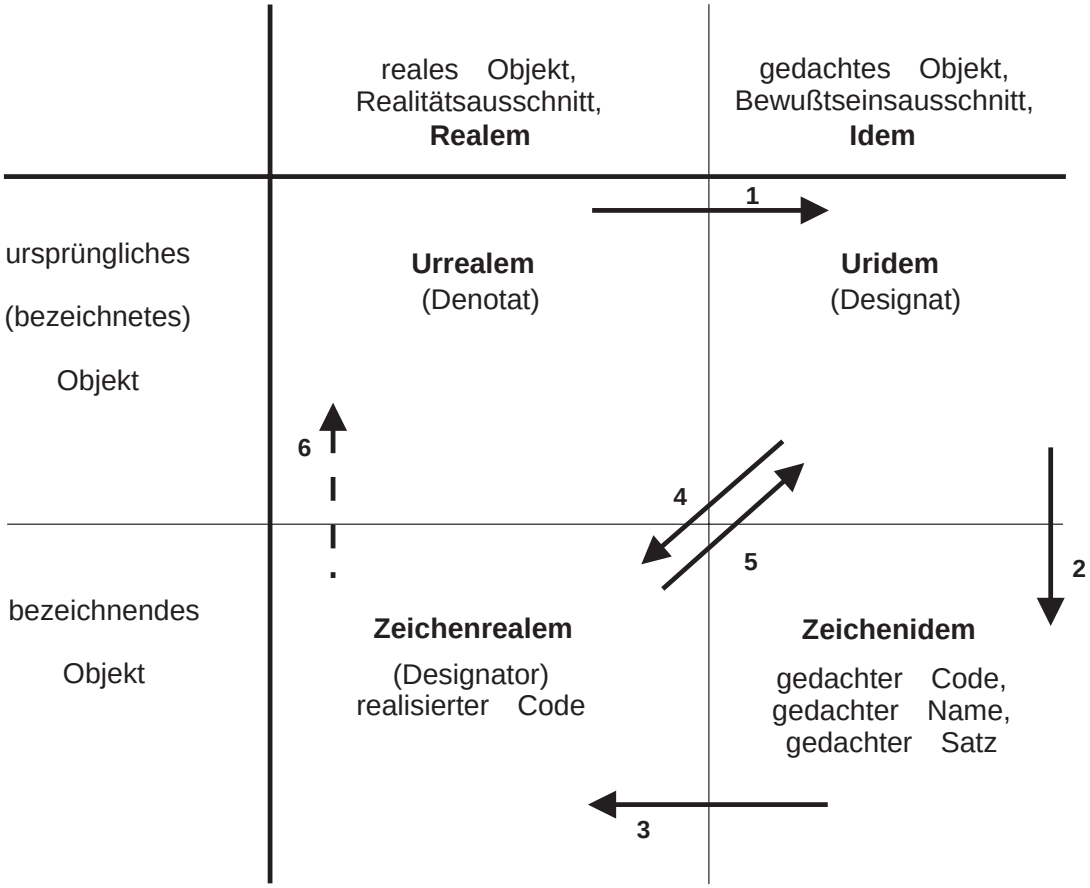

Bild 2.1 Abbildung des sprachlichen Modellierens.
1- Interpretieren; 2 - internes Codieren, Benennen; 3 - externes Codieren, Ausprägen, materielles Instanzieren; 4 - Artikulieren; 5 - Interpretieren; 6- Realisieren eines sprachlichen Modells.

Die Pfeile geben die Übergänge zwischen den Objektklassen an. Sie sind mit den Bezeichnungen für die jeweiligen Übergangsprozesse benannt. *Artikulieren* ist das *Codieren* gedachter Bedeutungsinhalte mit Hilfe einer vereinbarten Sprache. Es erfolgt in zwei Schritten, dem *internen* und dem *externen Codieren*. (Man beachte, dass unser *Code*begriff die Begriffe *Zeichenrealem* und *Zeichenidem* umfasst.) Mit *Interpretieren* bezeichnen wir das Erkennen oder Ausdeuten von Urrealemen (nichtsymbolischen Objekten der Außenwelt) oder von Zeichenrealemen (symbolischen Objekten der Außenwelt).

Das Wort *Instanzieren*[1] (hier als Synonym zu *Ausprägen* verwendet) hat sich erst in jüngerer Zeit in der Informatik eingebürgert. Es kommt nicht vom Wort ”Instanz" im Sinne von Dienststelle, sondern vom englischen Wort “instance” im Sinne von “einzelner Fall” oder “Beispiel” und bezeichnet ganz allgemein den Übergang von einem Sammelbegriff oder von einer Klasse zu einem Exemplar der Klasse (vgl. Bild 5.5). Speziell bezeichnet materielles Instanzieren das Produzieren eines materiellen Exemplars, z.B. das hier gedruckte “A” als ein Exemplar des ersten Buchstabens des Alphabets. Mit *materiellem Instanzieren* kann man auch das *Prägen* eines Datenträgers, z.B. das Ein*stanzen* von Löchern in einen Lochstreifen oder eine Lochkarte assoziieren , was heutzutage allerdings praktisch ausgestorben ist.

Die Wege über die Pfeilfolgen 1-2-3 und 1-4 entsprechen beide dem externsprachlichen Modellieren. Die Pfeile zeigen vom Original zum Abbild (bzw. zur Widerspiegelung). Nur der gestrichelte Pfeil 6 zeigt in entgegengesetzter Richtung, wenn er als Abbildung der *sprachlichen* Modellierung aufgefasst wird. Er kann aber auch als Abbildung der *nichtsprachlichen* (*analogen*) Modellierung aufgefasst werden, von der in Kap.4 die Rede sein wird. Dann zeigt er von einem externsprachlichen Realem zu einem realen Objekt (z.B. von einer Differenzialgleichung zu einem entsprechend programmierten Analogrechner; siehe Kap.4.2 [4.5]).

Die eingeführten Begriffe sollen anhand eines Beispiels illustriert werden. Urrealem (primäres Objekt) möge mein alter Bekannter namens *Hans* sein. Wenn ich auf der Straße eine Person sehe und als *Hans* erkenne, so heißt dies, dass *Hans* in mein Bewusstsein getreten ist, m.a.W. dass mein Gehirn das gesehene Objekt als *Hans* interpretiert hat, dass also das Idem *Hans* ausgelöst wird. Angenommen, wir haben uns lange nicht gesehen, sodass ich mich erst auf seinen Namen besinnen muss. Erst wenn dieser in mein Bewusstsein getreten ist, kann ich meinen Bekannten anreden, d.h. den gedachten Namen aussprechen (materiell instanzieren).

Die Selbstverständlichkeit des Sprechens, z.B. des Anredens mit Namen, hat ihre Ursache darin, dass einem das Interpretieren und gedankliche Benennen nicht zum Bewusstsein kommt, sondern reflektorisch überbrückt ist. Tatsächlich haben sich diese Stufen des bewussten Artikulierens erst im Laufe der genetischen Evolution herausgebildet. Eine motorische Reaktion z.B. des Kehlkopfes auf eine Empfindung, z.B. eine visuelle oder akustische, setzt noch kein bewusstes Denken in Vorstellungen, Begriffen und Symbolen voraus. Auch Tiere können auf Eindrücke mit Lauten reagieren. Die Lautbildung erfolgt also ursprünglich auf subsymbolischer Ebene [1.4].

Mit Hilfe der eingeführten Begriffe können wir die in Kap.1 [1.3] gegebene
3 Definition der *Information* präzisieren: *Die Zusammenfassung eines Idems mit dem zugeordneten Realem heißt* ***Information****. Im Falle eines Zeichenrealem heißt sie*

1 In der Literatur wird i.Allg. “Instantiieren” geschrieben. Im Sinne der neuen Rechtschreibung ist “Instanzieren” vorzuziehen.

artikulierte Information oder ***Zeicheninformation****. Andernfalls heißt sie nichtartikulierte oder* ***uneigentliche Information****.* Wenn nichts anderes gesagt wird, ist im Weiteren unter Information stets Zeicheninformation zu verstehen. Dem Begriff der Zeicheninformation entspricht etwa der linguistische Begriff des *Denotators* (siehe z.B. [Conrad 75]. Der in Kap.1 eingeführte Symbolbegriff kann nun auch als *vereinbarte Zeicheninformation* definiert werden.

Die Präzisierung des Informationsbegriffs hat mehrere Konsequenzen. Zunächst einmal bedeutet sie, dass nicht nur ein Idem, sondern auch eine Information an einen individuellen Träger, an den Sender oder Empfänger gebunden ist (insofern impliziert der Informationsbegriff den Sender bzw. Empfänger). Diese Vorstellung ist nicht üblich, bedeutet aber eine begriffliche Klärung. Danach schließt nämlich das Übertragen von Information nicht nur das Transportiern, sondern auch das Artikulieren und Interpretieren ein. Information wird also nur soweit übertragen, als Senderidem und Empfängeridem zusammenfallen. Andernfalls kommt es zu Missverständnissen. Oft wird das Wort *Informationsübertragung* irreführenderweise im Sinne von *Zeichenrealemübetragung* verwendet.

Eine andere Konsequenz ergibt sich aus der Möglichkeit der mehrfachen Instanzierung von Zeichenidemen sowie der Vervielfältigung von Zeichenrealemen und damit von Informationen. Es gibt also keinen Erhaltungssatz der Information, wie er z.B. für die Energie gilt. Hier wird deutlich, dass in diesem Buch der Informationsbegriff nicht in der üblichen Weise verwendet wird. Wenn beispielsweise jemand ein und dieselbe Auskunft zweimal erhält, handelt es sich nach obiger Definition um verschiedenen Informationen. Zwar sind die Ideme die gleichen, doch die Realeme sind unterschiedliche, z.B. zwei verschiedene bedruckte Blätter.

Aber nicht nur das Instanzieren, sondern auch das Benennen (Codieren) kann wiederholt werden. Es kann auch reflexiv (zirkulär) erfolgen; ein Name kann sich selbst benennen. Auf die Konsequenzen und begrifflichen Schwierigkeiten, die sich daraus ergeben, gehen wir in Kap.6 ein.

Abschließend sei auf ein fundamentales Problem des sprachlichen Modellierens hingewiesen. Wie ist es möglich, die unendliche Vielfalt der Welt mit den endlichen Mitteln der Sprache darzustellen? Man ist daran gewöhnt, dass man alles, was man sieht, beschreiben kann, und kommt gar nicht auf den Gedanken, sich darüber zu wundern, dass das überhaupt möglich ist, wie man sich eben auch nicht darüber wundert, dass man das, was man denkt, auch sagen kann, wovon eingangs die Rede war. In Kap.5 werden wir näher untersuchen, wie die Evolution das angedeutete Problem gelöst hat. Dabei handelt es sich nicht um die genetische, sondern um die intellektuelle und die kulturelle Evolution. *Sprache ist Mittel und Gegenstand sowohl der intellektuellen Evolution des Individuums als auch der kulturellen Evolution der Menschheit*.

Nachden wir den Begriff der Information definiert haben, sind wir für das nächste Kapitel gerüstet. Wir werden versuchen, den Begriff der Informatik zu definieren, indem wir den Gegenstand bestimmen, mit dem sich die Informatik als Wissenschaft

beschäftigt. Wir werden uns nicht damit begnügen, eine Reihe von Teilgebieten aufzuzählen, in deren Bezeichnung das Wort “Information” vorkommt, sondern wir werden nach der Rolle suchen, die der zu bestimmende Gegenstand im menschlichen Leben spielt.

3 Informatik - Lehre vom aktiven sprachlichen Modellieren

3.1 Modellklassen. Begriffsbestimmung der Informatik

In der Literatur findet man die unterschiedlichsten Definitionen der Informatik. Meistens handelt es dich um Definitionen durch Aufzählung von Teilgebieten, z.B.: *Die Informatik ist die Wissenschaft von der Verarbeitung, Speicherung und Übertragung von Informationen*, oder: *Die Informatik ist die Wissenschaft von der Erstellung und Nutzung der Hardware und Software informationeller Systeme*, oder mehr aus der Sicht der letztendlichen technischen Zielstellung: *Die Informatik ist die Wissenschaft von der Entwicklung und Nutzung künstlicher Intelligenz.* Zuweilen wird auch einfach gesagt: *Die Informatik ist die Wissenschaft von der Informationsverarbeitung.* Das ist sicher nicht falsch, aber wenig aussagekräftig. Nach diesem Vorbild könnte man beispielsweise die Chemie als Wissenschaft von der Chemikalienverarbeitung definieren. Im Informatik-Duden [Duden 89] ist die Informatik etwas präziser als "*Wissenschaft von der systematischen Verarbeitung von Informationen, besonders der automatischen Verarbeitung mit Hilfe von Digitalrechnern (Computern)*" definiert, ähnlich in [Werner 95]: "*Die Informatik (computer science) ist eine Wissenschafts- und Technologiedisziplin, die sich mit Methoden und Verfahren der automatisierten Informationsverarbeitung befasst*".

Vielleicht sollte man sich mit einer Definition analog derjenigen begnügen, die DAVID HILBERT für die Mathematik vorgeschlagen hat, und sagen: *Informatik ist das, worüber sich kompetente Personen auf Informatikerkongressen unterhalten*. Hinter diesem Vorschlag steht die Einsicht, dass eine Wissenschaft ihr lebendiger Inhalt ist, der sich kaum durch eine tote Definition erfassen lässt. Das ist sicher richtig. Dennoch soll versucht werden, eine Begriffsbestimmung zu finden, die keine Aufzählung verschiedener Gegenstände der Informatik darstellt und die nicht auf Begriffen wie Information oder Intelligenz beruht.

Gesucht sind nicht die Attribute und Erscheinungsformen der Informationsverarbeitung, sondern ihr wesentlicher Kern, die Wurzel, aus der alles wächst, sich alles "von selber" ergibt, auch der Informationsbegriff. Diesen hatten wir aber bereits eingeführt [1.3] und zwar ausgehend vom *sprachlichen Modellieren*. Danach liegt es nahe, die Informatik als Lehre vom sprachlichen Modellieren zu definieren. Diese Bezeichnung kann aber auch die Physik für sich beanspruchen, im Grunde sogar jede Wissenschaft, die sprachliche - z.B. mathematische - Modelle der Welt entwickelt. Die Definition ist offensichtlich zu allgemein. Die notwendige Präzisierung ergibt sich aus Bild 3.1. Es zeigt eine Klassifikation von Modellen nach verschiedenen Merkmalen.

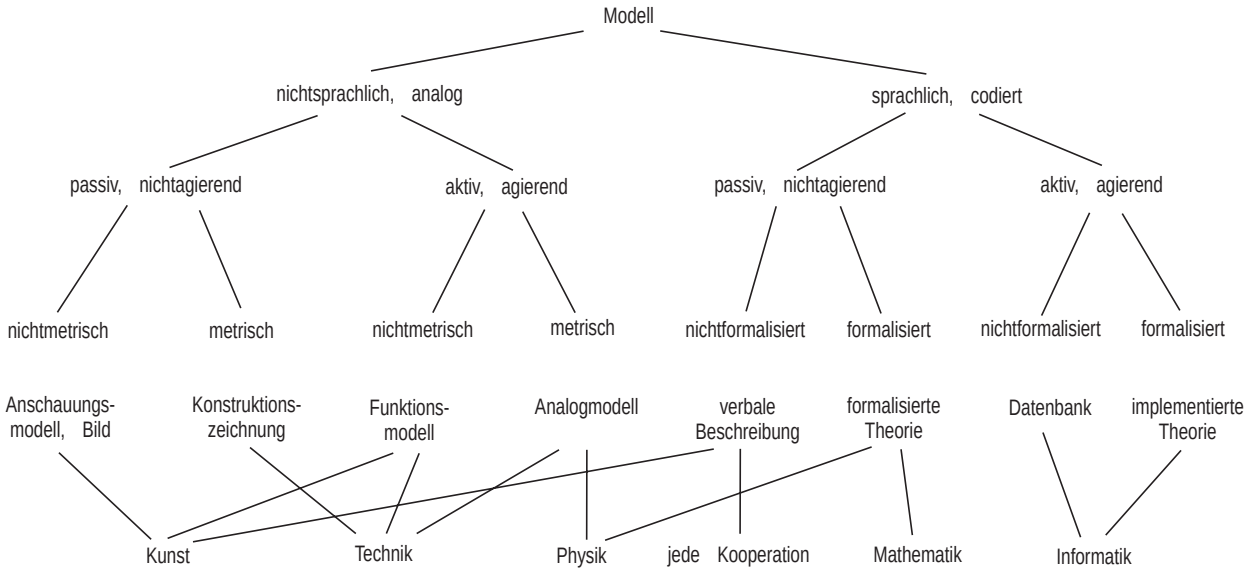

Bild 3.1 Modellklassen

Die dargestellte Systematik unterteilt die Menge möglicher Modelle nach drei Kriterien in je zwei Teilmengen und zwar:

1. Nichtsprachliche oder **analoge** und **sprachliche** oder **codierte Modelle**. *Analoge* Modelle verwenden keine Codierung. Die zu modellierenden Eigenschaften des Originals haben ihr anschauliches Analogon im Modell. Ein sprachliches Modell ist eine Menge von *Aussagen* über das Original. In erweitertem Sinne nennen wir jedes Zeichenrealem, dem ein Urrealem entspricht, sprachliches Modell des Urrealems (des Originals).

2. Aktive oder agierende und **passive** oder **nichtagierende Modelle.** *Aktive analoge* Modelle veranschaulichen Prozesse, indem sie selber agieren (z.B. eine Modelleisenbahn); sie sind dynamisch.

Aktive oder *agierende* sprachliche Modelle sind solche, welche die Modellaussagen selber artikulieren bzw. interpretieren. *Ein aktives sprachliches Modell schließt also das Trägersystem - das kann ein Gerät (Computer) oder ein Mensch sein - in sich ein*. Für passive sprachliche Modelle gilt das nicht. Beispielsweise ist ein Kursbuch ein passives, ein Auskunftsautomat ein aktives sprachliches Modell der Zugverbindungen.

3. Exakte und **nichtexakte Modelle.** *Exakte analoge* Modelle gestatten es, Eigenschaften des Originals durch Zählen oder Messen am Modell zu bestimmen. Aus diesem Grunde sind die Bezeichnungen **metrisches Modell** bzw., falls die Bedingung der Meßbarkeit nicht gegeben ist, **nichtmetrisches Modell** gerechtfertigt.

Vorgreifend auf Kap. 5.4 vereinbaren wir: **Exaktes sprachliches Modellieren** ist das Artikulieren *objektiver* Aussagen über ein Original, d.h. solcher Aussagen, die von allen Beteiligten einheitlich, also *subjektunabhängig* interpretiert werden, wobei

die Objektivierung durch Anbindung der *externen* Semantik an eine *formale* Semantik mittels Formalisierung erreicht wird. Aus diesem Grunde sind die Bezeichnungen **formalisiertes** bzw. - im Falle nichtexakter Artikulation - **nichtformalisiertes Modell** gerechtfertigt.

Bei der soeben getroffendnen Vereinbarung wurde davon ausgegangen, dass die Wortverbindungen “externe Semantik” und “formale Semantik” vom Leser intuitiv richtig verstanden werden. Die Definitionen dieser Begriffe erfolgt in Kap.5.4. Auch die Erklärung des in Bild 3.1 auftretenden Begriffs der “formalisierten Theorie” muss auf Kap. 5.4 [5.11] (dort “*formale* Theorie genannt”) verschoben werden, weil vorher der *Kalkülbegriff* [5.9] eingeführt werden muss. Es wird sich herausstellen, dass für die Klassifikation sprachlicher Modelle das Wort “kalkülisiert” noch treffender ist als das Wort “formalisiert”.

Die behandelten drei Unterscheidungen von je zwei Fällen liefern insgesamt 8 Modellklassen. In Bild 3.1 sind Beispiele für die verschiedenen Klassen angeführt, die gleichzeitig darauf hinweisen, dass die Grenzen zwischen den Klassen nicht scharf sind. Wie sind z.B. Kunstwerke einzuordnen, die Symbole enthalten, wie beispielsweise Chagalls Bilder den Eselskopf oder Wagners Opern die Klangmotive der Helden. Das sind interessante Fragen, die hier nicht zur Debatte stehen.

Schließlich ist in Bild 3.1 durch Verbindungsgerade (z.B. zwischen “Kunst” und “Bild”) angegeben, welche Bereiche menschlicher Tätigkeit sich vorzugsweise welcher Modellklassen bedienen. Genau genommen wären weit mehr Verbindungsgeraden einzuzeichnen, beispielsweise von “verbale Beschreibung” zu sämtlichen Bereichen. Man könnte versuchen, daraus Begriffsbestimmungen z.B. der Physik, der Mathematik und auch der Informatik abzuleiten. Für letztere ergibt sich: *Die **Informatik** ist die Lehre vom aktiven sprachlichen Modellieren*[1], eine ungewöhnliche, aber, wie sich zeigen wird, tragfähige Begriffsbestimmung. Die Tragfähigkeit muss sich im lebendigen Sprachgebrauch erweisen und ihre Konsequenzen müssen diskutiert werden. Gegen sie lassen sich verschiedene Einwände erheben. Auf zwei Einwände soll eingegangen werden.

Erster Einwand. Nach der gegebenen Definition ist ein Mensch mit seinem inneren Modell der Welt ein aktives sprachliches Modell und als solches Gegenstand der Informatik. Die Informatik ist aber eine technische Wissenschaft und der Mensch nicht ihr Gegenstand. Außerdem ist es eine extrem reduktionistische Herangehensweise, im Menschen ein agierendes sprachliches Modell zu sehen. Tatsächlich ist die Auffassung, die Informatik sei eine rein technische Wissenschaft, eine Gewohnheit aus der Vergangenheit, und die Themen heutiger Informatikerkongresse zeigen, dass diese Gewohnheit überlebt ist. Mit dem Vorwurf des Reduktionismus werden wir uns im Nachwort auseinandersetzen. Er kann nur gegen kategorische Erklärungen

1 Diese Definition ist erstmalig in [Jungclaussen 88a] veröffentlicht worden.

erhoben werden. Solche ergeben sich jedoch nicht zwangsläufig aus obiger Definition.

Zweiter Einwand. Es ist üblich, zwei Klassen informationeller Systeme zu unterscheiden: *Digitalrechner* und *Analogrechner*. Letztere verwenden keine Codierung und keine Symbole. Sie verarbeiten keine Information im Sinne unserer Definition. Ein "programmierter", d.h. für die Modellierung eines Originalbereiches *konditionierter* (vorbereiteter, konkret: "verdrahteter") Analogrechner ist also kein sprachliches, sondern ein analoges, metrisches Modell. Er ist also weder Mittel noch Gegenstand der Informatik im oben definierten Sinne. Das ist ungewöhnlich, aber konsequent und vermeidet viele Missverständnisse und inhaltlose Diskussionen. Hier sind wir auf einen Sachverhalt gestoßen, der für ein tieferes Verständnis dessen, was Informatik ist, hervorragende Bedeutung hat. Darum soll auf ihn in Kap.4 näher eingegangen werden.

3.2 Fundamente der Informatik

Im vorangehenden Kapitel wurde gefordert, dass sich die Tragfähigkeit der dort gegebenen Definition der Informatik als "Lehre vom aktiven sprachlichen Modellieren" im lebendigen Sprachgebrauch zu erweisen habe. Es soll versucht werden, den Nachweis dadurch zu erbringen, dass aus der gegebenen *intensionalen*, also logisch-inhaltlichen Definition eine *extensionale* Definition abgeleitet wird, also eine Aufzählung der Teilbereiche der Informatik oder, im Sinne von Hilbert, eine Aufzählung dessen, worüber sich Informatiker auf ihren Kongressen unterhalten. Zu diesem Zwecke stellen wir folgende Fragen: Was muss man verstehen, um das aktive sprachliche Modellieren zu verstehen und was muss man tun, um es zu realisieren, kurz: *Was sind die Fundamente der Informatik*? [2]

Bild 3.2 gibt Antwort auf die gestellten Fragen. In ihr sind die wichtigsten Wissenschaftsdisziplinen, die für das Verständnis und die Verwirklichung des aktiven sprachlichen Modellierens die Voraussetzungen liefern, zusammengestellt. Der eigentliche Tabelleninhalt (die beiden rechten Spalten ohne die Kopfzeile) enthält 10 Felder. In jedem Feld ist eine Disziplin (ein Wissenschaftszweig) angegeben, welche die Grundlagen für die Untersuchung bzw. Realisierung des in der 2. Spalte der jeweiligen Zeile bezeichneten Gegenstandes liefert. Dabei ist zwischen natürlichem (biologischem, organismischem) sprachlichem Modellieren (linke Spalte) und künstlichem (technischem) sprachlichem Modellieren (rechte Spalte) unterschieden. Die Teilbereiche sind in 2 Gruppen zusammengefasst. Die Teilbereiche der ersten (oberen) Gruppe beschäftigen sich mit dem Trägersystem. Sie untersuchen die Funktionsprinzipien des *systeminternen* (primär *subsymbolischen*) Modellierens.

2 Mit geringen Änderungen [Jungclaussen 88] entnommen.

Die Teilbereiche der zweiten Gruppe untersuchen die Möglichkeiten und Mittel der *externen* Modellierung, genauer der externen, symbolischen Darstellung interner Modelle.

Ein Beispiel soll das Lesen der Tabelle demonstrieren. Die Wissenschaftsdisziplin, die sich mit den Artikulierungsmitteln beschäftigt, ist im Falle des künstlichen sprachlichen Modellierens die *Wissenschaft von den Programmiersprachen*, im Falle des natürlichen sprachlichen Modellierens die *Sprachwissenschaft*, speziell die *Semiotik*.

	Gegenstand	natürliche Modelle	künstliche Modelle
interne Modelle	Trägersystem	Neurophysiologie	Gerätetechnik der Computer-IV
	Entwicklung bzw. Herstellung des Trägersystems	Wissenschaft von der Evolution des Zentralnervensystems	Hardwaretechnologie
externe Modelle	Artikulierungsmittel	Sprachwissenschaft, Semiotik	Wissenschaft von den Programmiersprachen
	Artikulierung	Schreib- und Redekunst (Rhetorik)	Programmierungstechnik, Softwaretechnologie
	Metamodellierung	Wissenschaft von der natürlichen Intelligenz	Wissenschaft von der künstlichen Intelligenz

Bild 3.2 Fundamente des sprachlichen Modellierens

Die letzte Zeile der Tabelle enthält zwei Begriffe, die erklärt werden müssen. Unter ***Metamodellierung*** ist hier die *externe sprachliche Modellierung der internen sprachlichen Modellierung* zu verstehen. (Nach dem üblichen Sprachgebrauch wäre unter sprachlichem Metamodell (metasprachlichem Modell) das externsprachliche Modell eines *externsprachlichen* Modells zu verstehen). Das Wort “Intelligenz” werden wir hier und im Weiteren in einem Sinne verwenden, der sich ganz natürlich und zwanglos aus den bisherigen Gedankengängen ergibt: ***Intelligenz*** *ist die Fähigkeit ihres Trägers zur Erstellung und Nutzung interner, sprachlicher Modelle. Beschränkt sich die Fähigkeit auf die Nutzung, sprechen wir von reproduktiver, andernfalls von produktiver Intelligenz.* Demnach ist *künstliche Intelligenz* die Fähigkeit *technischer* informationeller Systeme zum sprachlichen Modellieren.

1

Diese Begriffsbestimmung mag insofern mangelhaft erscheinen, als sie Intelligenz auf das Modellieren beschränkt und das Erfinden als typische Fähigkeit intelligenter Wesen nicht berücksichtigt. Tatsächlich ist aber auch Erfinden sprachliches Modellieren, wenn auch einer noch nicht existierenden Realität, oder besser und allgemeiner: *Sprachliches Modellieren ist Mittel des Erfindens und Problemlösens*. In Kap.7 werden wir untersuchen, inwiefern umgekehrt Erfinden Mittel des sprachlichen Modellierens ist.

Gehirnprozesse, die dem gedanklichen Modellieren der Welt oder dem Erfinden gedachter Welten zugrunde liegen, bezeichnen wir als Denken. Dazu gehört auch das
2 bildliche (zwei- oder dreidimensionale) Modellieren, das *Sich-Vorstellen* der Welt. ***Denken*** *ist sprachliches Modellieren ohne externes Codieren (Artikulieren).*

Sieht man sich die rechte Spalte von Bild 3.2 genauer an, erkennt man eine Liste von Teilgebieten, die eine extensionale Definition der Informatik darstellt, wie sie häufig zu finden ist und die einer Definition im HILBERTschen Sinne sehr nahe kommt. Diesen Umstand werten wir als Nachweis der Tragfähigkeit unserer intensionalen Definition der Informatik als *Lehre vom aktiven sprachlichen Modellieren.* Man beachte, dass in der rechten Spalte von Bild 3.2 die Fundamente nur eines Teils der Informatik aufgelistet sind und zwar desjenigen Teils, welcher das sprachliche Modellieren durch technische Geräte (Computer) betrifft. Diesen Teil nennen wir **technische Informatik**.[3] Demgegenüber sind in der vorletzten Spalte die Fundamente desjenigen Teils der Informatik aufgelistet, welcher das sprachliche Modellieren durch den Menschen betrifft. Man könnte ihn **Humaninformatik** nennen. Dieser Teil ist ein Unterbereich der **biologischen Informatik**. Sie beschäftigt sich mit der Informationsverarbeitung durch Lebewesen, den Menschen eingeschlossen[4]. Die biologische Informatik umfasst zwei Bereiche, die Lehre von der sprachlichen Modellierung der Welt mit Hilfe der “Sprache des Nervensystems” und die Lehre von der sprachlichen Modellierung des eigenen Organismus mit Hilfe der “Sprache des genetischen Systems”. Die biologische Informatik ist nicht der eigentliche Gegenstand des Buches. Dennoch muss auf die Informationsverarbeitung durch den Menschen eingegangen werden, um verstehen zu können, wie eine sprachliche Modellierung der Welt durch den Computer und wie die Kommunikation zwischen Mensch und Maschine möglich ist.

Wir verlassen jetzt den Bereich der Informatik, um die Grenze zwischen der digitalen Rechentechnik und der analogen Rechentechnik, die nicht zur Informatik (im Sinne dieses Buches) gehört, herauszuarbeiten. In Kap.5 werden wir unseren Gedankengang zum sprachlichen Modellieren fortsetzen.

---

3 Es sei darauf hingewiesen, dass die Wortverbindung “technischen Informatik” in der Literatur zuweilen in anderen, spezielleren Bedeutungen verwendet wird.

4 Mit *Bioinformatik* wird die Anwendung der technischen Informatik in der Biologie bezeichnet, in Analogie zur Wirtschaftsinformatik, der Anwendung der Technischen Informatik im Bereich der Wirtschaft.

4 Analoges und digitales Modellieren

Zusammenfassung

Die kulturelle Evolution hat dazu geführt, dass wir uns Raum, Zeit und Kausalität als Kontinuum vorstellen. Der Versuch, die räumlichen, zeitlichen und kausalen Gegebenheiten der Welt sprachlich zu modellieren, stößt auf eine grundsätzliche Schwierigkeit, auf den Widerspruch zwischen der kontinuierlichen Natur des Gedachten, des Vorgestellten, und der nichtkontinuierlichen Natur des sprachlichen Modellierens, und zwar sowohl des *extern nichtcodierten* Modellierens, des Denkens, als auch des *extern codierten* Modellierens, des Sprechens und Schreibens.

Um die kontinuierliche Welt quantitativ und kontinuierlich modellieren zu können, sind die reellen Zahlen und die Infinitesimalrechnung erfunden worden. Dennoch bleibt das quantitative Modellieren in Zahlen ein diskontinuierliches Modellieren. Die kontinuierliche Folge der reellen Zahlen oder der Punkte einer Linie sind sprachlich nicht realisierbar, sondern nur mit Hilfe abstrakter Begriffe beschreibbar, z.B. mit Hilfe des Begriffs der reellen Zahl, des Grenzwertes oder des Differenzialquotienten.

Das diskontinuierliche sprachliche Modellieren, das Denken und Sprechen, basiert auf der kontinuierlichen physischen Realität (im Sinne der klassischen Physik) als dem stofflichen Träger des Modellierens. Beim Übergang aus dem kontinuierlichen (*analogen*) in den diskontinuierlichen (*digitalen*) Bereich muss eine Analog-digital-Konversion der Merkmalswerte stattfinden, derer sich das Modellieren bedient. Sie kann von einem Gerät (Konverter) oder vom Menschen ausgeführt werden, indem er z.B. einen Wert misst oder berechnet. Ein agierendes Modell heißt analog oder nichtsprachlich, wenn die Konversion nicht Teil des Modells ist, sondern *nach* der Bestimmung der Merkmalswerte vorgenommen wird. Ein Modell heißt digital oder sprachlich, wenn die Konversion *vor* der Bestimmung der Merkmalswerte vorgenommen wird, also Teil des Modells ist.

Ein analoges Modell wird durch dieselben Gleichungen (z.B. Differenzialgleichungen) beschrieben wie das Original. Das Original selbst stellt eine analoge Lösung der Gleichungen dar; ein analoges Modell "*berechnet*" die Lösung, d.h. es generiert die *Lösungsfunktion*. Dafür hat die Analogrechentechnik elektronische Geräte entwickelt. Die Merkmalswerte (die Werte der Lösungsfunktion) sind Werte elektrischer Größen. Sie werden vom Modell in analoger Form geliefert und können analog, z.B. als Zeigerausschlag oder als Strecke auf dem Bildschirm ausgegeben werden. Die Ausgabe kann auch digital als Dezimalzahl erfolgen, wobei der Digitalwert ein Näherungswert ist (falls die betreffende Gleichung nicht ausnahmsweise eine rationalzahlige Lösung besitzt). Für die Digitalausgabe ist ein Konverter erforderlich, der die Grenze des analogen Bereichs überschreitet.

4.1 Messen und reelle Zahlen

In Kap.2 war an Sprachen die Forderung gestellt worden, dass sie die Möglichkeit bieten müssen, die Unendlichkeit der Welt sprachlich wiederzugeben. Dieser Satz demonstriert mit der Verwendung des Wortes *Unendlichkeit* die Methode, wie Sprachen die gestellte Forderung erfüllen können. Es bilden sich im Laufe der Zeit geeignete *Begriffe* heraus, also *benannte Ideme,* wie beispielsweise der Begriff der Unendlichkeit. Das ist ein Beispiel für ein Idem ohne entsprechendes anschauliches Urrealem; das Unendliche ist nicht vorstellbar. Dennoch ist der Begriff nützlich und sogar objektivierbar bis hin zum mathematischen Begriff des Unendlichen, genauer des unendlich Großen. Er ist ein erstaunliches Resultat der kulturellen Evolution und löst den fundamentalen *Widerspruch zwischen der Unendlichkeit des Gedachten und der Endlichkeit des Denkens und der Sprache.* Es ist offensichtlich, dass eine Sprache, genauer die Anzahl ihrer Zeichenrealeme und der ihnen entsprechenden Ideme, endlich sein muss, um ihre Artikulierbarkeit und Interpretierbarkeit zu gewährleisten.

Daneben hat sich der Begriff des "*unendlich Kleinen*" gebildet. Man gelangt zu ihm, wenn man ein Stück Realität, z.B. einen Stein, eine Länge oder ein Zeitintervall gedanklich "unendlich" verkleinert. Intuitiv ist man davon überzeugt, dass unendliches Verkleinern tatsächlich möglich ist, dass die Punkte des Raumes unendlich dicht nebeneinander liegen, dass der Raum ein *Kontinuum* bildet und dass dasselbe auch für die Zeit gilt. Diese Vorstellung ist uns infolge ständiger Erfahrung beim Beobachten und "Begreifen" der Welt in Fleisch und Blut übergegangen.

Der Wunsch (der evolutionäre Zwang), das Kontinuum mit Worten zu erfassen, führt zu einem zweiten fundamentalen Widerspruch des sprachlichen Modellierens,
1 dem *Widerspruch zwischen der kontinuierlichen Natur des Gedachten und der nichtkontinuierlichen Natur des Denkens.* Letztere ergibt sich wiederum aus der Endlichkeit der Menge der Zeichenrealeme und der ihnen entprechenden Ideme. Auch die Begriffe des unendlich Kleinen und des Kontinuums sind objektivierbar und mathematisierbar. Darauf soll näher eingegangen werden, weil hier der Schlüssel zur Unterscheidung zwischen *analogem* und *digitalem* Modellieren liegt.

Wir beginnen mit der gedanklichen und sprachlichen Modellierung des *linearen* (*eindimensionalen*) räumlichen Kontinuums, der *Linie.* Global erfolgt sie eben durch das Wort "Linie". Eine Linie lässt sich als unendliche Menge aneinandergereihter Punkte gedanklich und sprachlich (begrifflich) modellieren. Die exakte (quantitative, metrische) Modellierung erfolgt durch Zuordnung von Zahlen zu den Punkten der Linie. Dazu werden zwei Punkte markiert und mit 0 und 1 benannt. Das Linienstück zwischen den beiden Punkten ist die Längeneinheit. Nun wird jedem Punkt der Linie diejenige Zahl zugeordnet, die seinen Abstand vom Nullpunkt gemessen in Längeneinheiten angibt, normalerweise in Form einer gebrochenen Dezimalzahl. Wenn die Linie eine Gerade ist, spricht man von einer *Zahlengeraden*.

Die Zuordnung einer Zahl zu einer Länge nach der eben beschriebenen Vorschrift wird *Digitalisierung* oder A*nalog-digital-Konvertierung* genannt. Das Wort *digital* kommt vom englischen digit = einstellige Zahl, Ziffer. Das Wort *analog* in diesem Zusammenhang ist das Ergebnis eines Bedeutungswandels, auf den wir noch zu sprechen kommen. Die genannte Konvertierung ist der Übergang vom analogen zum digitalen, in diesem Fall vom geometrisch-kontinuierlichen zum sprachlich-diskreten Modellieren. Das ist leicht gesagt aber schwer getan. Es stellt sich nämlich heraus, dass den meisten Punkten der Zahlengeraden gar keine in Ziffern genau angebbare Zahl entspricht. Das manifestiert sich in der Ungenauigkeit, mit der sich die Länge eines Abschnitts der Zahlengeraden (einer Strecke) messen bzw. berechnen lässt.

Die Ungenauigkeit hat unterschiedliche Ursachen. Jedes Messen ist, genau genommen, ein *Klassifizieren.* Alle Längen, die innerhalb eines bestimmten Wertebereichs liegen oder innerhalb der *Messgenauigkeit* einander gleich sind, werden zu einer *Klasse* zusammengefasst, und die Klasse wird mit einer Zahl *benannt*, dem gemessenen Wert. Die Größe des Wertebereichs stellt die minimale Längeneinheit dar, also die Genauigkeit, mit der gemessen wird. Die Prozedur des Messens (Klassifizierens) ist ein Zählen; es wird gezählt, wie oft die Längeneinheit in den zu messenden Abschnitt hineinpasst. Das gilt für alle Messungen, die auf Längenmessungen zurückgeführt werden, z.B. für Druck- oder Temperaturmessungen, die auf die Messung der Länge einer Flüssigkeitssäule (z.B.Quecksilbersäule) zurückgeführt werden, aber auch für alle Messungen mit Hilfe von Zeigerintrumenten. In jedem Fall handelt es sich um ein “Zerstückeln”, ein *Diskretisieren des Kontinuums* und Zuordnung von Zahlen zu den Stücken (nicht zu Punkten), also um ein *Digitalisieren.* 2

Die Begriffe Messen und Digitalisieren sind demnach Synonyme. Diese Feststellung ist ungewohnt, jedoch selbstverständlich, wenn man sich vergegenwärtigt, dass Messen das *Kontinuumproblem* lösen, d.h. den oben genannten fundamentalen Widerspruch überbrücken muss.

Üblicherweise werden die Wörter “Messen” und “Digitalisieren” nicht als Synonyme aufgefasst, sondern es wird nur dann von Digitalisieren gesprochen, wenn dies durch ein Gerät, einen sog. *Umsetzer* oder *Konverter* erfolgt, nicht aber, wenn es durch den Menschen erfolgt, wie z.B. beim Ablesen eines analog anzeigenden Messinstruments. Tatsächlich spielt in diesem Fall der Mensch die Rolle eines Analog-digital-Konverters.

Technische Analog-digital-Konverter arbeiten nach dem Schwellwertprinzip. Sie enthalten ein oder mehrere *Schwellenoperatoren.* Bild 4.1 zeigt die Funktionsweise eines solchen Operators. Auf den Eingang wird eine re-

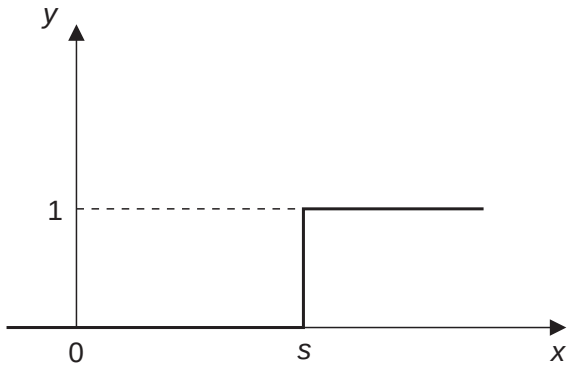

Bild 4.1 Funktionsweise des Schwellenoperators

elle Größe *x* gegeben. Das ist in der Regel eine Spannung, die von einem geeigneten Messfühler geliefert wird. Die Ausgabegröße *y* kann zwei Werte annehmen, die mit 0 und 1 bezeichnet sind. Der Wert zeigt an, ob *x* eine bestimmte *Schwelle s* überschreitet (dann ist $y = 1$) oder nicht ($y = 0$).

Mit Hilfe von zwei Schwellenoperatoren, deren Schwellen die Werte *s* und $s + \Delta s$ haben, lässt sich feststellen, ob der Wert von *x* in den Bereich (in den *Spalt*) zwischen *s* und $s + \Delta s$ fällt. Eine Messung von *x* mit der Genauigkeit Δs lässt sich nun dadurch bewerkstelligen, dass *s*, beginnend mit 0, schrittweise um Δs erhöht wird, und dabei gezählt wird, wieviele Schritte getan werden müssen, bis *x* die untere Schwelle überschreitet, die obere jedoch nicht, also in den momentanen Spalt fällt. Dies ist das Grundprinzip jedes Messens.

Die Umkehrung der Längenmessung, also das Abtragen einer zahlenmäßig gegebenen Länge auf einer Geraden als Strecke, ist eine Überführung einer zahlenmäßigen, also sprachlichen in eine kontinuierliche Darstellung, eine sog. *Digital-analog-Konvertierung*.

3 Die Unmöglichkeit des absolut genauen Messens, m.a.W. der prinzipielle Näherungscharakter des Digitalisierens, ist also nicht einfach die Folge technischer Unzulänglichkeiten, sondern letzten Endes eine prinzipielle Gegebenheit, die ihre Ursache in der diskreten Natur des Denkens und der Sprache hat. Wenn man dagegen im analogen Bereich bleibt, wenn man z.B. *analog rechnet* (siehe weiter unten), ohne zu digitalisieren, dann freilich ist die "Rechenungenauigkeit" allein die Folge technischer Unzulänglichkeiten.

Ähnliches wie für das Messen gilt auch für das Berechnen. *Die absolut genaue Berechnung einer durch Konstruktion festgelegten Länge ist - zumindest in der Mehrzahl der Fälle - nicht möglich.* Diese Unmöglichkeit überrascht wohl niemanden, obwohl sie nicht weniger merkwürdig ist als die prinzipielle Unmöglichkeit des absolut genauen Messens. Man hat sich an sie gewöhnt.

Konstruiert man z.B. über der Zahlengeraden ein gleichschenklig-rechtwinkliges Dreieck, dessen Katheten die Länge 1 besitzen und dessen Hypotenuse auf die Zahlengerade und dessen linke Ecke in den Nullpunkt fällt, so markiert die rechte Ecke einen Punkt, der nicht exakt digitalisierbar, also nicht exakt mit Hilfe von Ziffern angebbar ist. Indirekt ist er allerdings dennoch sprachlich modellierbar, indem man eine Vorschrift für seine Berechnung angibt oder ihm einfach irgendeinen Namen zuweist, z.B. $\sqrt{2}$. Versucht man aber, die Wurzel exakt zu ziehen, stellt man fest, dass dies nicht möglich ist, weil die Prozedur der *numerischen (zahlenmäßigen) Berechnung* niemals abbricht, sondern sich ins Unendliche fortsetzt. Das exakte Ergebnis ist eine Unendliche Folge von Ziffern. Derartige Zahlen heißen *Irrationalzahlen*.

Eine andere Irrationalzahl ergibt sich, wenn man einen Kreis mit dem Durchmesser 1 auf der Zahlengeraden abrollt, beginnend im Nullpunkt. Nach einer vollen Umdrehung des Kreises markiert dieser auf der Zahlengeraden einen Punkt, dem sich wiederum keine Zahl in Form einer endlichen Ziffernfolge zuordnen lässt. Um ihn

dennoch sprachlich exakt angeben zu können, hat man ihm den Namen π zugewiesen. Irrationalzahlen sind im Gegensatz zu den sog. *Rationalzahlen* nicht als Quotient zweier ganzer Zahlen darstellbar. Beide Zahlenklassen werden zur Klasse (oder Menge) der *reellen Zahlen* zusammengefasst.

Man kann nun die scheinbar dumme Frage stellen, wie viele reelle Zahlen es gibt. Natürlich gibt es unendlich viele, denn die Zahlengerade besitzt unendlich viele Punkte, sie ist ein Kontinuum. Nun gibt es aber bereits unendlich viele rationale Zahlen, und man könnte glauben, dass sie ausreichen, um alle Punkte der Zahlengeraden darzustellen (auf sie abzubilden). Das ist jedoch, wie wir gesehen haben, nicht der Fall. Zwischen den Punkten, denen rationale Zahlen entsprechen, liegen unendlich viele weitere Punkte. Tatsächlich ist die Anzahl der reellen Zahlen in "höherem Grade unendlich" als die der rationalen Zahlen. Die Mathematiker drücken sich exakter aus und sagen: *Die Menge der rationalen Zahlen ist* ***abzählbar unendlich****, die der reellen Zahlen ist* ***überabzählbar unendlich***.

Diese Ausdrucksweise hat folgende Ursache. Man kann zeigen, dass sich die rationalen Zahlen in einer lückenlosen und unendlich fortsetzbaren Reihe ordnen und also auch abzählen lassen. Für die reellen Zahlen ist das nicht möglich. Nach welcher Regel man auch versucht, sie zu ordnen, es bleiben immer unendlich viele Lücken erhalten. Hier zeigt sich der Widerspruch zwischen der diskreten Natur des Denkens und der kontinuierlichen Natur des Gedachten in *zahlentheoretischer* Form.

4.2 Analoges Rechnen

Wir wollen nun auch die Zeit in unsere Überlegungen einbeziehen. Das zeitliche Kontinuum ist ebenso wie die Linie eindimensional und lässt sich analog als Zahlengerade darstellen. Zusätzlich ist dabei deren positive Richtung vorgegeben, nämlich als zeitlicher Fortschritt aus der Vergangenheit in die Zukunft.

Die Vorstellung der kontinuierlichen Zeit führt zur Vorstellung der Kontinuität von *Prozessen*, d.h. der Kontinuität der zeitlichen Änderung von Merkmalen realer Objekte, z.B. ihrer Lage (Kontinuität der Bewegung). Es erhebt sich nun die Frage, wie sich die Kontinuität von Prozessen und speziell die Kontinuität der Bewegung sprachlich modellieren lässt. Diese Frage beschäftigte die Menschen seit Jahrtausenden, nachweislich seit dem Streit zwischen zwei philosophischen Schulen des griechischen Altertums, den Herakliten und den Eleaten. Gelöst wurde das Problem erst in der Neuzeit, und zwar durch ISAAC NEWTON, als er versuchte, die Planetenbewegung zu *erklären.* Es ist aufschlussreich, den Weg dahin zu verfolgen, denn er ist charakteristisch dafür, wie die Physiker die Natur sprachlich, und zwar mathematisch modellieren.

Am Anfang jeder Modellbildung steht die Beobachtung und deren Protokoll. Im Falle der newtonschen Himmelsmechanik waren das die Messergebnisse von TYCHO

DE BRAHE. Er hat genau protokolliert, welcher Planet wann und wo zu sehen war. Das Beobachtungsprotokoll eines Planeten hat die Form einer zeitlichen Reihe

$$z_0, z_1, \ldots, z_{t-1}, z_t, \ldots,$$

wo z_t den Ort zum Zeitpunkt t_1, also den gemessenen Azimut und Erhebungswinkel, bezeichnet, z stellt also ein Wertepaar dar. Aus solchen Protokollen hat JOHANNES KEPLER seine berühmten Gesetze abgeleitet, aus denen sich die Bewegung jedes Planeten als stetige Funktion der Zeit $z(t)$ berechnen lässt. Die keplerschen Gesetze beschreiben die Planetenbewegung mit erstaunlicher Genauigkeit, aber sie erklären sie nicht. Es sind empirische Gesetze. Ihre Erklärung gelang Newton 70 Jahre später. Er konnte sie aus seinen Prinzipien der Mechanik und seinem Gravitationsprinzip ableiten. Dazu musste er die Differenzialrechnung erfinden.

Newton ging davon aus, dass der Zustand eines dynamischen Systems die kausale Folge des vorangehenden Zustandes ist. Das lässt sich formal durch

$$z_t = f(z_{t-1}) \quad \text{bzw.} \quad z_t = f(z_{t-1}, x_{t-1})$$

ausdrücken. Die erste Gleichung gilt für abgeschlossene oder *autonome*, die zweite für nicht abgeschlossene oder *nichtautonome* Systeme, wobei x die Einwirkung der Umwelt bezeichnet. Wenn f eine bekannte Funktion ist, lässt sich z für alle möglichen (diskreten) Zeitpunkte berechnen. Beispielsweise lässt sich die Bewegung des Sekundenzeigers einer Uhr, der in jeder Sekunde einen Sprung macht, durch die Gleichung $z(t) = z(t–1) + 6$ beschreiben, wobei z die Zeigerstellung in Grad ist und der Winkel im Uhrzeigersinn gemessen wird.

4 Newton gab sich mit einer solchen diskreten, genauer *kausaldiskreten* Beschreibung nicht zufrieden, denn sie gibt den Ursache-Wirkungs-Zusammenhang nur in Sprüngen, also unvollständig wieder. Er war überzeugt, dass nicht nur Raum und Zeit, sondern auch die Kausalität, der Ursache-Wirkungs-Zusammenhang kontinuierlicher Natur ist, dass also der jeweilige Zustand eines abgeschlossenen Systems - letztendlich des Weltalls - die Folge des "unendlich kurz" vorangehenden Zustandes ist. Er suchte nach einer *kausalkontinuierlichen* Beschreibung. Das stieß auf eine grundsätzliche Schwierigkeit. Wenn man das Zeitintervall Δt immer kleiner und schließlich zu *0* werden lässt, dann wird, wie die Erfahrung zeigt, auch die Zustandsänderung Δz zu *0* (die Natur macht keine Sprünge, "natura non facit saltus"), und die obige kausaldiskrete Beschreibung verliert ihren Sinn. Newton fand folgenden Ausweg.

Bekanntlich hat der Quotient 0/0 i.Allg. keinen bestimmten Wert, d.h. er kann jeden beliebigen Wert annehmen. Das gilt auch für $\Delta z/\Delta t$, wenn beide Intervalle gleich 0 sind. Wenn aber z eine *glatte* Funktion der Zeit ist, also eine Funktion ohne Sprünge und Ecken, dann strebt der *Differenzenquotient* $\Delta z/\Delta t$ mit verschwindendem Δt einem bestimmten *Grenzwert* zu, dem sog. *Differenzialquotienten*, der mit dz/dt bezeichnet wird. Beispielsweise sei $z = 3t$. Dann gilt stets $\Delta z/\Delta t = 3$, unabhängig von

der Größe des Zeitintervalls Δt, und auch im Grenzfall $\Delta t \to 0$ gilt $dz/dt = 3$. Geometrisch bedeutet dies, dass die Funktion $z(t)$ eine Gerade mit der *Steigung* 3 beschreibt. Physikalisch wird dadurch eine Bewegung mit der konstanten *Geschwindigkeit* 3 beschrieben, z.B. mit der Geschwindigkeit 3 m/s, falls z in Metern und t in Sekunden gemessen wurde. Wenn mit Größen gerechnet wird, deren Wert in dem beschriebenen Sinne "gegen den Grenzwert Null streben", spricht man von *Infinitesimalrechnung*.

Der Differenzialquotient dz/dt wird auch als *Ableitung der Funktion* $z(t)$ *nach der Zeit* bezeichnet. Seine Berechnung wird *Differenzieren* oder *Ableiten* genannt. Die Ableitung einer Funktion ist wieder eine Funktion von der gleichen Variablen.

Die Erfindung des Differenzierens ermöglicht eine kausalkontinuierliche Prozessbeschreibung, indem nicht der Modellparameter z als kausale Folge des vorangehenden Zustandes aufgefasst wird, sondern seine zeitliche Ableitung dz/dt als kausale Folge von z, formal notiert:

$$dz/dt = f(z) \quad \text{bzw.} \quad dz/dt = f(z,x) \tag{4.1}$$

Derartige Gleichungen heißen *Differenzialgleichungen*. Die erste Gleichung gilt wieder für autonome, die zweite für nichtautonome Systeme. Ebenso wie z ist auch x eine Funktion der Zeit. Wenn die Funktion f bekannt ist, lässt sich der zeitliche Verlauf des Zustandes $z(t)$ aus der Differanzialgleichung berechnen, oft allerdings nur näherungsweise. Für die kausalkontinuierliche Beschreibung der Planetenbewegung leitete Newton Differenzialgleichungen ab, deren Lösung auf die keplerschen Gesetze führt. Die Gleichungen sind allerdings komplizierter als (4.1) und enthalten nicht nur die erste Ableitung (s.u.). Damit begann eine neue Epoche der Physik. Die Differenzialgleichung wurde zum wichtigsten Arbeitsmittel der theoretischen Physiker.

Es sei noch einmal unterstrichen, dass die Prozessbeschreibung mittels Differenzialgleichungen eine Modellierung des Kontinuums mit sprachlichen, also nichtkontinuierlichen (diskreten) Mitteln darstellt. Erst bedeutend später erfand man eine systematische und allgemein verwendbare Methode, den kausalkontinuierlich gedachten Ablauf von Prozessen auch kausalkontinuierlich, also *nicht*sprachlich zu *modellieren*. Der Methode liegt folgende Idee zugrunde.

Wenn zwei verschiedene Prozesse durch formal identische Differenzialgleichungen beschrieben werden, so verlaufen sie offensichtlich "analog" (die Anführungsstriche zeigen an, dass das Wort nicht in der bisherigen, sondern in seiner umgangssprachlichen Bedeutung verwendet wurde) d.h. die Prozessparameter werden durch (im Wesentlichen) gleiche Zeitfunktionen beschrieben. Dabei können die analogen Prozesse ganz unterschiedlicher Natur sein. So lassen sich beispielsweise Luft-, Licht- und Wasserwellen bei geeigneter Idealisierung [5.12] durch ein und dieselbe Differenzialgleichung, die sog. *Wellengleichung*, beschreiben. Es ist also möglich, einen dieser Prozesse durch einen anderen zu modellieren. Auf dieser Tatsache fußt

die Idee des *Analogrechners*. Der erfindungsreiche Weg zur Verwirklichung dieser Idee soll in großen Zügen skizziert werden.

Wir gehen von dem obigen Beispiel der *geometrisch-analogen Konstruktion* der Größe $\sqrt{2}$ aus. Derartige analoge Konstruktionsmethoden lassen sich auch für andere numerische Rechenoperationen angeben. So kann die Summe $a+b$ durch Aneinanderlegen zweier Strecken der Länge a und b bestimmt werden. Das Produkt $a \times b$ lässt sich analog als Fläche eines Rechtecks mit den Seitenlängen a und b konstruieren. Um das Resultat zu digitalisieren, kann man den Flächeninhalt z.B. mit Hilfe eines Planimeters bestimmen. Man kann auch das Rechteck ausschneiden und sein Gewicht und das Gewicht der Flächeneinheit bestimmen. Ferner besteht die Möglichkeit, das Multiplizieren auf das Aneinanderlegen von Strecken zurückzuführen, indem man als Maßzahlen für die Längen nicht die Zahlenwerte der Faktoren, sondern deren Logarithmen verwendet. Nach diesem Prinzip arbeitet der Rechenschieber.

Wenn man nun zu jeder arithmetischen Operation eine geometrisch-konstruktive Variante realisiert, sozusagen ein "*analoges Analogon*", dann lässt sich jede numerische Rechnung, also jedes Rechnen mit Zahlen auch konstruktiv, also kontinuierlich (genauer: räumlich kontinuierlich) ausführen. Man spricht dann von "*analogem Rechnen*". Die Wortverbindung "analoges Analogon" ist kein Pleonasmus (wie z.B. "runder Kreis"), sie widerspiegelt vielmehr den Bedeutungswandel des Wortes "analog", von dem bereits die Rede war. Ursprünglich bezeichnete es die Entsprechung, die Ähnlichkeitsbeziehung, präziser die *Homomorphie* zwischen diskontinuierlich-zahlenmäßigem und kontinuierlich-konstruktivem "Rechnen", allgemeiner kann man auch sagen: zwischen exaktem (formalisiertem) sprachlichem und exaktem (metrischem) nichtsprachlichem Modellieren. Später bürgerte es sich ein, das kontinuierliche Rechnen selber als - im übertragenen Sinne - *analog* zu bezeichnen und auch die Operanden und Resultate des analogen Rechnens *analoge Größen* zu nennen. In der obigen Wortverbindung "analoges Analogon" ist das erste Wort im übertragenen, das zweite im ursprünglichen Sinne zu verstehen.

Vom logischen Standpunkt aus ist ein solcher Sprachgebrauch irritierend. Aber Sprachen entwickeln sich eben nicht nach logischen Gesichtspunkten, selbst die Sprache der Techniker und sogar der Mathematiker nicht unbedingt. Noch irritierender wird die Terminologie, wenn das zeitliche Kontinuum von dieser Sprachregelung ausgenommen wird, wie es häufig der Fall ist, wenn also hinsichtlich der Zeit von kontinuierlichen, sonst aber von analogen Größen gesprochen wird.

Der nächste Schritt in Richtung Analogrechner besteht im Übergang vom geometrischen zum physikalischen, speziell zum elektronischen analogen Rechnen. Um das physikalische Analogon einer arithmetischen Operation zu realisieren, muss man einen physikalischen, z.B. elektrischen Effekt finden, der einer Gesetzmäßigkeit gehorcht, die sich mathematisch durch die betreffende Operation ausdrückt. Wenn beispielsweise zwei Spannungsquellen in Reihe geschaltet werden, ergibt sich die Summe der Einzelspannungen. Entsprechend kann man Differenzen bilden. Tatsäch-

lich lassen sich sämtliche arithmetische Operationen analog-elektronisch realisieren und die Resultate "*analog berechnen*" (generieren), allerdings nur mit einer bestimmten Genauigkeit, die von den technischen Parametern der Schaltung abhängt und in der Regel mit Mühe 0,001% erreicht.

Der letzte Schritt zum Analogrechner besteht in der elektronischen Realisierung des *Integrierens*. Mit Integrieren wird die Umkehrung des Differenzierens bezeichnet, also das Bestimmen einer Funktion aus ihrer Ableitung. Das Ergebnis heißt *Integral*. Geometrisch kann das Integrieren als Flächenbestimmung aufgefasst werden. Sein geometrisch-konstruktives Analogon ist also das Planimetrieren. Das elektronische Analogon lässt sich mit Hilfe eines Kondensators realisieren. Die Geschwindigkeit, mit der sich die Spannung an einem Kondensator ändert, also die Ableitung der Spannung nach der Zeit, ist nämlich proportional dem elektrischen Strom, der durch den Kondensator fließt, m.a.W. du/dt ist proportional dem Ladungsfluss i, der den Kondensator auf- bzw. entlädt. Folglich ist die Spannung proportional dem Integral des Stromes, formal notiert:

$$u(t) = \text{const} \int i(t)\, \mathrm{d}t$$

Dabei ist angenommen, dass im Zeitpunkt t=0 auch u=0 gilt. Mit const ist eine Proportionalitätskonstante bezeichnet; ihr Kehrwert heißt *Kapazität* des Kondensators. Es sind elektronische Integratoren hoher Genauigkeit entwickelt worden, deren Kernstück ein Kondensator ist.

Ein ***Analogrechner*** *ist in seiner ursprünglichen Form ein Baukasten elektronischer Standardschaltungen zur analogen Berechnung derjenigen Operationen, die in Gleichungen (z.B. algebraischen Gleichungen oder Differenzialgleichungen) auftreten.* Für seine Verwendung zur *analogen Prozessmodellierung* ergibt sich aus den vorangehenden Überlegungen folgende Vorgehensweise. Zunächst versucht man, die zeitliche Änderung des zu modellierenden Prozessparameters mittels einer Differenzialgleichung zu beschreiben. Wenn das gelungen ist, verbindet man die Elemente des elektronischen Baukastens (des Analogrechners) so miteinander, wie die Differenzialgleichung es vorschreibt. Dann stellt der so "programmierte" (d.h. verschaltete) Analogrechner ein analoges Prozessmodell dar, d.h. der zeitliche Verlauf seiner Ausgangsspannung ist identisch mit dem des modellierten Prozessparameters.

Die *Programmierung* eines Analogrechners - diese Redeweise ist irritierend, hat sich aber eingebürgert - entspricht dem Pfeil 6 in Bild 2.1, also dem Übergang von einem externen sprachlichen zu einem externen nichtsprachlichen Objekt, speziell in unserem Falle von einem formalisierten sprachlichen zu einem metrischen nichtsprachlichen Modell. Die Programmierung soll an zwei Beispielen demonstriert werden, zunächst anhand der rechten Differenzialgleichungen (4.1)

$$\mathrm{d}z/\mathrm{d}t = f(z(t),x(t)),$$

wobei die Abhängigkeit der Variablen z und x von der Zeit explizit angegeben ist. Integration beider Seiten ergibt (von einer additiven Konstanten abgesehen)

$$z(t) = \int f(z(t),x(t))\,\mathrm{d}t.$$

Wir wollen annehmen, dass die Funktion $f(t) = z(t) + x\,(t)$ ist. Bild 4.2a zeigt die Analogrechnerschaltung (einen programmierten Analogrechner), der die Differenzialgleichung *löst,* d.h. der am Ausgang $z(t)$ liefert. Das Programmieren besteht darin, dass zwei analoge Operatoren, ein Summator und ein Integrator zu einem Ring (einer Rückkopplungsschleife) miteinander verbunden und Ein- und Ausgabeleitungen angekoppelt werden, so wie es die Differenzialgleichung vorschreibt. Dabei entspricht dem Gleichheitszeichen das Schließen der Rückkopplungsschleife vom Integrator zum Summator. Die Schließung hat zur Folge, dass der Ausgabewert des Summators, also $z+x$, stets gleich ist dem Eingabewert des Integrators, also $\mathrm{d}z/\mathrm{d}t$, wie die Differenzialgleichung es verlangt. Auf Probleme der Dimensionierung soll nicht eingegangen werden.

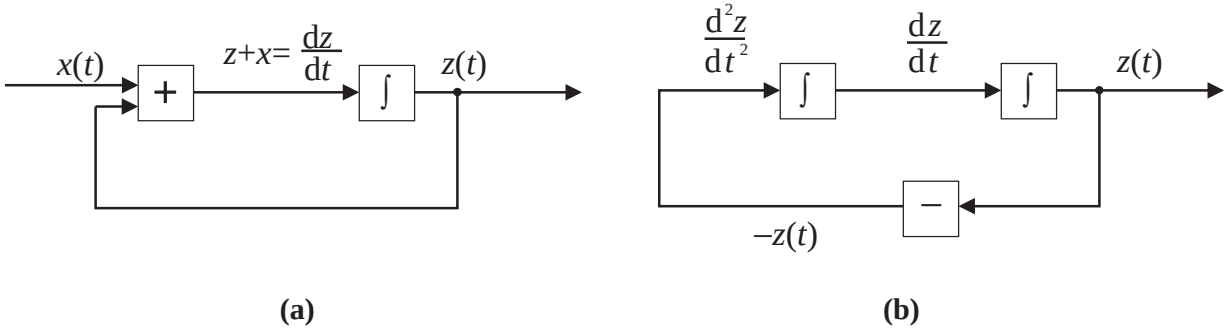

Bild 4.2 Analogrechnerschaltung zur Lösung zweier einfacher Differenzialgleichungen

Bild 4.2b zeigt ein zweites Beispiel. Diese Schaltung löst die denkbar einfachste *Differenzialgleichung zweiter Ordnung*. So wird eine Differenzialgleichung genannt, welche die *zweite Ableitung*, das ist die Ableitung der Ableitung, enthält (aber keine höhere Ableitung). Anstelle von $\mathrm{d}(\mathrm{d}z/\mathrm{d}t)/\mathrm{d}t$ (Ableitung der Ableitung) schreibt man $\mathrm{d}^2z/\mathrm{d}t^2$. Zur Lösung der Gleichung sind zwei Integratoren erforderlich. Die dargestellte Schaltung enthält einen weiteren Operator, der den negativen Wert des Eingabewertes liefert (Multiplikation mit –1). Die Schaltung löst die Gleichung

$$\mathrm{d}^2z/\mathrm{d}t^2 = -z. \qquad (4.2)$$

Das ist die Wellengleichung in ihrer einfachsten Form. Die Lösung lautet (von Konstanten abgesehen) $z = \sin(t)$, denn die zweite Ableitung der Sinusfunktion ist die Sinusfunktion mit negativem Vorzeichen. Das ist eine periodische Schwingung mit der Frequenz 1 pro Zeiteinheit (von der Dimensionierung sehen wir wieder ab).

Die Schaltung in Bild 4.2b besitzt keinen Eingang, sie stellt ein autonomes System dar, sie schwingt “von selbst”, sie *generiert* Sinusschwingungen. Sie kann also als

sogenannter *Sinusgenerator* eingesetzt werden. Die Leitungsführung auch dieser Schaltung läuft in sich selbst zurück, sie ist zu einem "Kreis" oder "Zirkel" kurzgeschlossen, der Input ist gleich dem Output. Darin spiegelt sich die für die newtonschen Bewegungsgleichungen charakteristische Relativität von Ursache und Wirkung wider. In ihnen kann man die Kraft als Ursache und die Beschleunigung als Wirkung auffassen, mit demselben Recht aber auch die Beschleunigung als Ursache und die Kraft (Trägheitskraft) als Wirkung. In diesem Sinne sprechen wir von 6
Ursache-Wirkung-Zirkularität. In Kap.6.3 [6.4] kommen wir darauf zurück.

Bei diesen wenigen Bemerkungen zur Analogrechentechnik wollen wir es bewenden lassen. Viele wichtige Details sind nicht zur Sprache gekommen, u.a. die Eingabe von Anfangswerten, die Vorzeichenumkehr, die Dimensionierung und Normierung.[1]

Die Analogrechentechnik hat bei der Entwicklung der modernen *Regelungstechnik* eine bedeutende Rolle gespielt. Ein starker Impuls ging von NORBERT WIENER aus, von seiner Idee einheitlicher Prinzipien der Regelung und Informationsübertragung in Lebewesen und in Maschinen [Wiener 63]. Er stellte das Regelungsproblem, genauer das Problem der Selbststeuerung und Selbststabilisierung von Prozessen durch Rückkopplung, in einen neuen, fast universell anmutenden Zusammenhang. Die Popularität seiner Bücher und seiner Wortschöpfung "*Kybernetik*" löste einen regelrechten Boom aus.

Inzwischen sind die wienerschen Ideen von allerhand nachträglichem Beiwerk wieder befreit worden, um das andere Autoren sie "bereichert" hatten. Das Wertvolle ist geblieben, aber leider auch die Gewohnheit, das, was in einen Analogrechner eingegeben wird und was er ausgibt, als *Information* und den Analogrechner selbst als informationelles System zu bezeichnen. Das ist eine schlechte, weil irreführende Gewohnheit. Denn mit dem gleichen Recht könnte man beispielsweise auch Ein- und Ausgangsspannungen eines Transformators als Information bezeichnen, ja, sogar den Durchmesser eines Werkstücks, das auf einer Drehbank bearbeitet wird. Schließlich könnte jede kausale Folge als Information über die Ursache aufgefasst werden. Derartige Vorschläge gibt es. Aber sie führen zu inhaltlosen Diskussionen über den Begriff der Information, der bei einer derartigen Verallgemeinerung seinen eigentlichen Inhalt verliert, sodass er überflüssig wird. Seine eigenständige Bedeutung erhält der Informationsbegriff erst in Verbindung mit Zeichen, mit Codieren.

Wenn wir die Analogrechentechnik relativ ausführlich behandelt haben, obwohl sie nach unseren Begriffsbestimmungen nicht zur Informatik gehört, so nicht nur, um die Einordnung des Analogrechners in die Modellklassifikation gemäß Bild 3.1 zu begründen und um das Verständnis des Unterschiedes zwischen analogem und digitalem Modellieren zu vertiefen, sondern auch, weil in vielen Bereichen der Praxis der Analogrechner mit zum Instrumentarium des Informatikers gehört. Damit hat es folgende Bewandtnis.

---

1 Eine ausführliche Darstellung findet der Leser z. B. in [Schwarz 71]

Wenn einem Analogrechner ein Digital-analog -Konverter vorgeschaltet und ein Analog-digital-Konverter nachgeschaltet wird, ergibt sich ein Gerät, das für den, der es als *schwarzen Kasten* betrachtet und sich für sein Innenleben nicht interessiert, als informationelles System darstellt. Ein Analogrechner lässt sich in jedes echte, d.h. durchgängig digital funktionierende informationelle System einbauen. Eine solche Vorgehensweise kann von Vorteil sein, wenn eine Berechnung mit vorgegebener Rechengenauigkeit auf digitalem Wege aufwendiger ist, als auf analogem. Das ist häufig bei der Lösung von Differenzialgleichungen der Fall.

7 Weit öfter ist in der Praxis das umgekehrte Vorgehen anzutreffen. Wenn einem Digitalrechner ein Analog-digital-Konverter vorgeschaltet und ein Digital-analog-Konverter nachgeschaltet wird, ergibt sich ein Gerät, das als schwarzer Kasten betrachtet einen Analogrechner darstellt. Man kann es z.B. in einen Regelkreis einbauen, wo es die Rolle des früher verwendeten echten Analogrechners übernimmt. Auf diese Weise lässt sich eine bedeutend flexiblere und auch präzisere Regelung erreichen. Das hat zu einer weiteren sprunghaften Entwicklung der Regelungstechnik geführt.

Abschließend sei eine Bemerkung gemacht, die für jeden, der mit Computern zu tun hat, eine Selbstverständlichkeit ist. Das Modellieren mit dem Digitalrechner muss aus der Sicht des Nutzers kein zahlenmäßiges Modellieren sein, kein numerisches Rechnen, was das Wort “Digitalrechner” eigentlich erwarten ließe. Es braucht überhaupt kein mathematisches Modellieren zu sein, sondern es kann ein Modellieren in irgendeiner Sprache, vielleicht in einer geeigneten Fachsprache sein, möglicherweise sogar in der natürlichen Umgangssprache. Ganz allgemein kann es ein Modellieren mit Hilfe irgendwelcher Symbole sein. Aus dieser Sicht wäre die Bezeichnung *Symbolrechner* treffender. Wenn dennoch von “Digitalrechner” gesprochen wird, so verbindet sich damit ein verdeckter Bedeutungswandel des Wortes “digital” von “zahlenmäßig” zu “zeichenmäßig” und schließlich zu “sprachlich”. Freilich ist dieser Bedeutungswandel nicht so tiefgehend wie der des Wortes “analog” von “entsprechend” über “kontinuierlich” zu “nichtsprachlich”.

Um schließlich noch einmal auf die Überschrift dieses Kapitels “Analoges und digitales Modellieren” zurückzukommen, so hätte sie, bei Verwendung der in Kap.3.1 benutzten Bezeichnungen für die verschiedenen Modellklassen auch “Nichtsprachliches und sprachliches Modellieren” lauten können.

5 Syntax und Semantik

Sprache ist das Vermögen, von endlichen Begriffen einen unendlichen Gebrauch zu machen
Wilhelm von Humboldt

Zusammenfassung

Sprachliches Modellieren beginnt mit der Unterscheidung von Gegenständen auf Grund unterschiedlicher Merkmale und mit der Herausbildung entsprechender *Denkobjekte* (Ideme). Ein *sprachliches Modell* ist eine Menge von Aussagen über das Original. *Aussagen* ordnen Objekten Merkmale zu. Die Ausdrucksstärke und Flexibilität der menschlichen Sprache, die erforderlich ist, um die Welt modellieren zu können, wird durch Begriffsbildung und syntaktische Regeln erreicht. Begriffe sind benannte Denkobjekte. Neue Begriffe können aus alten durch komponierende, idealisierende, klassifizierende oder generalisierende Abstraktion gebildet werden (siehe Bild 5.4).

Syntaktische Regeln, kurz *Syntaxregeln*, dienen der Reduzierung des Codierungsaufwandes. Um nicht für jedes Idem (jeden Bewusstseinsinhalt, jedes Denkobjekt) ein besonderes Zeichen (“Hieroglyph”) für seine Artikulierung benutzen zu müssen, werden aus einer relativ kleinen Anzahl elementarer Zeichenrealeme (Buchstaben, Morpheme, Wörter) nach bestimmten Regeln Kompositzeichen gebildet, sodass ein Empfänger (Interpretierer) jedem nach den Syntaxregeln komponierten Zeichenrealem (Wort, Satz) mehr oder weniger genau dasjenige Idem zuordnen kann, welches der Sender (Artikulierer) ihm zugeordnet hat.

Wenn davon ausgegangen wird, dass alle Beteiligten (z.B. alle Mitlieder einer Sprachgemeinschaft oder einer Diskussionsrunde) ein Zeichenrealem ausreichend einheitlich interpretieren, sodass eine Verständigung möglich ist, wird das zugeordnete Idem die *Semantik* des Zeichenrealems genannt. Das Wort *Bedeutung* wird umgangssprachlich sowohl im Sinne von Idem als auch im Sinne von Semantik verwendet. Wir folgen dem umgangssprachlichen Gebrauch, wenn aus dem Kontext hervorgeht, in welchem Sinne das Wort benutzt wird. Exakte Bedeutungsgleichheit kann durch Formalisierung erreicht werden. Dazu muss der Diskursbereich, d.i. der Originalbereich, der sprachlich modelliert wird, über den man sich unterhält, *kalkülisiert* werden, d.h. es muss ein “passender” Kalkül ge- oder erfunden werden, sodass sich der Originalbereich als Interpretation des Kalküls auffassen lässt. Eine solche Interpretation heißt *extern* (bezüglich des Kalküls).

Ein Kalkül ist durch eine formale (z.B. mathematische) Sprache und durch Regeln festgelegt, nach denen sich Ausdrücke der Sprache (d.h. Ausdrücke, die nach den Syntyxregeln der Sprache gebildet sind) in andere Ausdrücke überführen lassen. Die Bedeutung, die ein Zeichenrealem im Rahmen eines Kalküls besitzt (d.h. ohne externe Interpretation), heißt *formale Semantik*. Die Schwierigkeit des Kalkülisierens

liegt - abgesehen vom Finden oder Erfinden eines geeigneten Kalküls - in der Anbindung der *externen Semantik* (der Bedeutung von Zeichenrealemen hinsichtlich des Originalbereiches) an die formale Semantik.

Die Schwierigkeit des Modellierens durch den Computer (des Simulierens) liegt in der Anbindung der externen Semantik (*Nutzersemantik*, also der Semantik, "in der" ein Computernutzer den Originalbereich gedanklich modelliert) an die *interne Semantik* bezüglich des modellierenden Computers (*Computersemantik*). Die *trägerinterne* oder kurz *interne Semantik* eines Zeichenrealems ist die physische Wirkung, die das Zeichenrealem im Interpretierer (im Träger des Interpretationsprozesses) auslöst. Die trägerinterne Semantik bezüglich eines Menschen ist die Gesamtheit der neurophysiologischen Prozesse, die das betreffende Zeichenrealem im Gehirn auslöst. Die Computersemantik ist die Gesamtheit der elektronischen Prozesse, die das Zeichenrealem im Computer auslöst.

Die Informations- und Codierungstheorie vergleicht den Aufwand von Codierungen mittels Zeichenketten (also Zeichenkettenlängen) in verschiedenen Codierungssystemen (in verschiedenen Sprachen) ohne Berücksichtigung der Semantik. Es wird nicht die Codierung von Idemen durch Zeichenrealeme, sondern die *Um*codierung einer gegebenen Codierung in eine andere, speziell in die binäre Codierung mittels Bitketten untersucht. Mit Hilfe der Theorie lässt sich der Codierungsaufwand durch Umcodierung minimieren.

Das Verhältnis der Länge einer durch Umcodierung erhaltenen Bitkette zur Länge der ursprünglichen Zeichenkette (zur Länge der "*Nachricht*") heißt **Informationsgehalt** der Nachricht pro Zeichen. Bei unbeschränkter Verlängerung der Nachricht geht der analytische Ausdruck für den Informationsgehalt in die Formel für die Entropie eines thermodynamischen Systems über (von einem konstanten Faktor abgesehen). Dabei entsprechen die Wahrscheinlichkeiten in der Entropieformel den Auftrittswahrscheinlichkeiten der Alphabetzeichen in der (unendlich langen) Nachricht, d.h. in der Sprache, in der die Nachricht artikuliert ist. Der physikalische Entropiebegriff wird bei der Betrachtung der sprachlichen Modellierung auf subsymbolischer Ebene bedeutungsvoll.

5.1 Sprache und Evolution

Nach dem Einschub über die Analogrechentechnik setzen wir den unterbrochenen Gedankengang zur *sprachlichen* Modellierung fort. Kapitel 2 begann mit der Feststellung, dass Sprache der Kooperation dient und die Überlebenschancen einer Population erhöht. Diese steigen, je besser die Kommunikation innerhalb der Population funktioniert, d.h. je *schneller, umfassender und unmissverständlicher* sich die Individuen der Population untereinander verständigen können. Es ist also zu erwarten, dass die Herausbildung natürlicher Sprachen einem Selektionsdruck in Richtung dieser drei Eigenschaften unterliegt. Außerdem ist eine ständige Spracherweiterung

zu erwarten. Denn mit zunehmendem Wissen über die Welt erweitert sich die Sprache, mit deren Hilfe die Welt modelliert wird.

Die Entwicklung und Erweiterung natürlicher Sprachen ist ein Spezialfall der allgemeineren Gesetzmäßigkeit, dass sich mit forschreitender Evolution auch der Code (die Sprache) weiterentwickelt, mit dessen Hilfe der Nachfolger über das bisherige Evolutionsergebnis informiert wird. Das gilt für alle Arten der Evolution. Der genetische Code wird von den niederen Tieren bis zu den Primaten umfangreicher (wenn auch in erstaunlich geringem Maße); der Sprachumfang eines Menschen nimmt mit seiner geistigen Entwicklung zu; und die Sprachen der Völker wachsen mit ihrer Kultur, mit dem Wachsen der Welt 3, um mit Popper zu sprechen.

Nach dem Gesagten muss eine natürliche Sprache widersprüchlichen Forderungen genügen. Einerseits muss sie beliebig erweiterbar sein, und andrerseits muss ihr Umfang endlich bleiben, und die Artikulationen müssen räumlich und zeitlich endlich und sogar möglichst kompakt und schnell interpretierbar sein. Aber trotz der Endlichkeit der Sprache muss die Welt mit ihrer unendlichen Erscheinungsvielfalt ausdrückbar sein.

Die "Aufgabe", Sprachen zu entwickeln, die den genannten Forderungen gerecht werden, hat die Evolution elegant "gelöst". Das Ergebnis lässt sich mit drei Schlagwörtern charakterisieren:

- Begriffsbildung,
- Grammatik,
- Idemobjektivierung.

Zur Begriffsbildung. Die Erweiterung einer Sprache beinhaltet im wesentlichen die Erweiterung ihrer Lexik durch Bildung neuer *Begriffe*, d.h. neuer *benannter Ideme*. Auf diese Weise wird der modellierbare Originalbereich erweitert und gleichzeitig die Ausdrucksstärke der Sprache erhöht, wodurch Artikulierungen kürzer und die sprachlichen Modelle kompakter werden. Viele Denkinhalte (Ideme), sogar sehr komplexe, lassen sich durch Wörter (durch kurze Zeichenrealeme) benennen (codieren). Wörter wie "Ich", "Tier" und "All" mögen das illustrieren. Begriffsbildung macht das sprachliche Modellieren der "unendlichen Welt" möglich und dient der Erhöhung der Kompaktheit sprachlicher Ausdrücke und damit der Beschleunigung der Kommunikation, denn je kürzer eine Mitteilung ist, umso weniger Zeit erfordert in der Regel ihre Artikulierung und Interpretierung.

Begriffsbildung dient außerdem, wie das Codieren überhaupt, der Anpassung an die physiologischen Gegebenheiten, insbesondere der Anpassung des Umfanges einer Mitteilung an die begrenzte Kapazität des Kurzzeitgedächtnisses, das beim Artikulieren und Interpretieren benötigt wird.

Zur Grammatik. Unter der *Grammatik* einer Sprache, wie sie in der Schule gelehrt wird, versteht man die *Regeln, nach denen Wörter gebildet und geformt, z.B. dekliniert oder konjugiert werden (**Morphologie**), sowie die Regeln, nach denen Sätze gebildet werden (**Syntax**)*. Diese Regeln gewährleisten die Flexibilität, die notwendig ist, um mit einem begrenzten Wortschatz praktisch alle denkbaren Ideme

artikulieren zu können. Außerdem unterstützen sie das Interpretieren insofern, als die Bedeutung einer Aussage aus ihrem Aufbau *hergeleitet* werden kann. Das kommt dem Nutzer einer Sprache jedoch nur dann zum Bewusstsein, wenn er die Sprache nicht perfekt beherrscht. In der technischen Informationsverarbeitung (Computer-IV) spielt das Ableiten der Bedeutung von Befehlen und Programmen aus ihrer syntaktischen Struktur eine besondere Rolle.

Zur Idemobjektivierung. Die dritte der eingangs genannten Forderungen betrifft die Unmissverständlichkeit sprachlicher Ausdrücke. In Kap.2 hatten wir festgestellt, dass Missverständnisse dann auftreten können, wenn sich die Ideme, die der Sender und der Empfänger eines Zeichenrealems diesem zuordnen, voneinander unterscheiden. Völlig identisch sind sie sicher nie, doch muss eine für die Kooperation notwendige Übereinstimmung gewährleistet sein. Diese wird durch *Idemobjektivierung* erreicht. *Unter **Idemobjektivierung** verstehen wir die Vereinheitlichung der Bedeutungen, die verschiedene Menschen in ein und dieselbe sprachliche Äußerung hineinlegen, m.a.W. die Annäherung aller subjektiven Ideme an ein "objektives" Idem.* Idemobjektivierung findet ständig im Prozess der Kommunikation statt. Sowohl die Denkinhalte (Ideme) der Kooperationspartner als auch die Artikulationen der Denkinhalte passen sich aneinander an. Idemobjektivierung ist Voraussetzung und Bestandteil der kulturellen Evolution. Das gilt zunächst für die gesprochene und in der Folge auch für die geschriebene Sprache.

Zwischen nicht miteinander kommunizierenden Kulturkreisen kann keine sprachliche Anpassung stattfinden. Dennoch ähneln sich die Prinzipien der Bildung von Begriffen, Wörtern und Sätzen auch in nicht verwandten Sprachen stärker, als man angesichts der objektiv gegebenen Möglichkeiten annehmen könnte. Theoretisch wäre es beispielsweise denkbar, jedem Idem sein eigenes elementares Realem (Alphabetzeichen) zuzuordnen. Die chinesische Begriffsschrift geht ziemlich weit in dieser Richtung. Die konsequente Durchführung dieser Methode würde aber am begrenzten menschlichen Gedächtnis scheitern. Die relative Ähnlichkeit der Sprachen der Welt wird verständlich, wenn man bedenkt, dass sich alle Sprachen unter den gleichen biologischen Nebenbedingungen entwickelt haben, insofern als alle Menschen über praktisch die gleichen Hör- und Sprechwerkzeuge und über die gleichen bzw. sehr ähnlichen neuronalen sprachverarbeitenden Strukturen verfügen. Offenbar hat der Selektionsdruck die Sprachen, begonnen mit den elementaren Lautzeichen, den *Phonemen*, optimal an die Anatomie der Menschen und an die jeweiligen stabilen Umweltbedingungen angepasst.

Unsere Überlegungen führen zu dem Schluss, dass Sprachen Resultate sowohl der genetischen als auch der kulturellen Evolution sind. Wenn erworbene Eigenschaften vererbt werden können, besteht sogar die Möglichkeit einer Rückwirkung der intellektuellen und kulturellen auf die genetische Evolution. Das Ergebnis könnte beispielsweise die Entwicklung eines angeborenen Interpretationsapparates sein. Dieser scheint tatsächlich zu existieren, denn anders lässt sich die Schnelligkeit kaum

erklären, mit der ein Kind seine Muttersprache erlernt. NOAM CHOMSKY hat diese Frage ausführlich diskutiert [Chomsky 70].

5.2 Hierarchische Komponierung

Bei der "Lösung des Sprachproblems" hat die Natur sich eines ihrer "Lieblings-*tricks*" bedient, der Methode des *hierarchischen Komponierens*. (Der Leser gestatte die teleonomische Deutung der Evolution als zielstrebigen Prozess.) ***Hierarchisches Komponieren*** *besteht ganz allgemein darin, dass aus Bausteinen neue Objekte aufgebaut - wir sagen komponiert - werden, sog.* ***Komposite****. Diese können ihrerseits als Bausteine verwendet werden, sodass eine Hierarchie von Kompositen entsteht.* Nach diesem Prinzip ist die tote wie die lebende Materie aufgebaut, nach ihm organisieren sich menschliche Gesellschaften, und nach ihm haben sich auch die natürlichen Sprachen entwickelt. Bild 5.1 illustriert die hierarchische Komponierung am Beispiel eines Satzes der deutschen Schriftsprache. Dabei gilt das Alphabetprinzip: *Die Kompositzeichen einer Sprache werden aus Elementarzeichen, den* ***Alphabetzeichen****, hierarchisch komponiert. Die Anzahl der Alphabetzeichen und der Kompositzeichen ist endlich.* Das Komponieren erfolgt nach Regeln der jeweiligen Sprache.

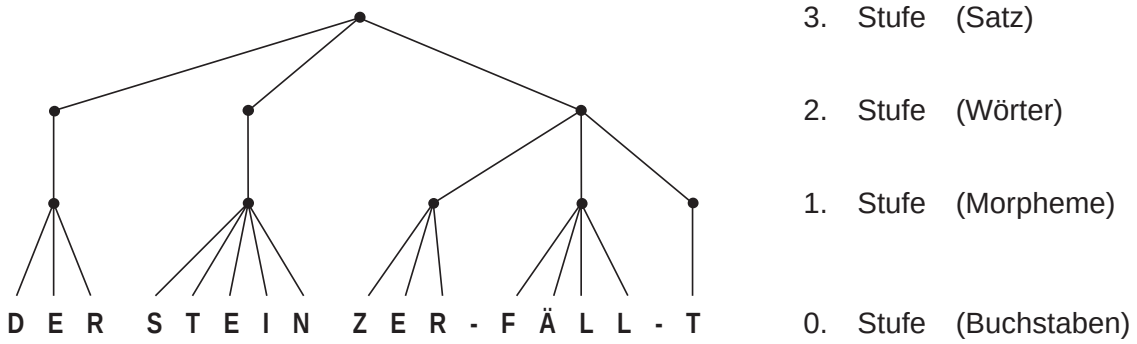

Bild 5.1 Hierarchische Zeichenkomponierung eines Satzes

Es liegt nahe, im hierarchischen Komponierungsprinzip, nach dem die Welt aufgebaut ist, ein ökonomisches Vorgehen der Natur zu sehen, in Analogie zum Vorgehen eines Ingenieurs, der seine Konstruktion aus Normteilen zusammensetzt, um die Herstellung und Wartung des Produktes kostengünstiger zu gestalten. In Wirklichkeit ist die scheinbar beabsichtigte Ökonomie ein Resultat der Evolution. Diese kann bei ihren Versuchen immer nur in sehr kleinen (z.B. kombinatorischen oder komponierenden) Schritten über das bereits Erprobte hinausgehen; auch hier gilt "natura non facit saltus". Für die genetische Evolution ist das sinnvoll, denn große Mutationen führen in der Regel zur Lebensunfähigkeit. Aus diesem Grunde konnte die Natur z.B. das Rad nicht "erfinden". Aus dem bereits Vorhandenen ließ es sich nicht komponieren.

Auch die Schritte der intellektuellen und in der Folge auch der kulturellen Evolution sind begrenzt. Phantasie und Denkkraft lassen nur kleine Abweichungen vom Gewohnten und damit nur kleine *Fort*schritte zu. Das gilt für alle Bereiche menschlicher intellektueller Tätigkeit, für die Technik ebenso wie für Kunst und Wissenschaft. Durch die Möglichkeit des gedanklichen Probierens wird die Evolution jedoch enorm beschleunigt und auch die Gedanken*sprünge* können größer werden. Der Mensch war in der Lage, das Rad zu erfinden, und die Entwicklung der künstlichen Sprachen macht unvergleichlich schnellere Fortschritte als die der natürlichen Sprachen. Nichtsdestoweniger ist es erstaunlich, wie langsam und mühsam sich die Programmiersprachen den Bedürfnissen der Nutzer anpassen. Das wird in Teil 3 deutlich werden.

Die hierarchische Struktur sprachlicher Ausdrücke ist der Syntax zuzurechnen, die anschließend behandelt wird. Wir haben die Besprechung des hierarchischen Komponierens vorgezogen, weil es ein Grundprinzip der Informatik ist, nach dem sowohl die Hardware als auch die Software informationeller Systeme aufgebaut ist.

5.3. Syntaxregeln

Das Wort *Syntax* wird in zwei Bedeutungen verwendet:

1. *Die **Syntax** eines Kompositzeichens ist dessen formale Komponierungsstruktur, d.h. sein Aufbau aus elementareren Zeichen (Bausteinzeichen).*
2. *Die **Syntax** einer Sprache ist die Gesamtheit aller Regeln (sog. Syntaxregeln), nach denen die Kompositzeichen der Sprache (z.B. Wörter, Sätze, Kommandos, Programme) formal, ohne Berücksichtigung der Bedeutung komponiert werden dürfen.*

Die aktuelle Bedeutung des Wortes “Syntax” geht aus dem jeweiligen Kontext hervor.

Die wichtigste Syntaxregel der deutschen Sprache legt fest, wie ein Aussagesatz normalerweise aufgebaut ist. Sie lautet:

<Aussagesatz> → <Subjekt> <Prädikat> [<Objekt>] (5.1)

3 z.B. “Hunde jagen Katzen.” Die Form, in der die Regel angegeben ist, lehnt sich an die sog. *Backus-Naur-Form* an. Der Pfeil (früher wurde an seiner Stelle ::= geschrieben) kann gelesen werden als “hat folgende Struktur”. Die Regel sagt aus, dass ein Aussagesatz mit dem Subjekt beginnt, ihm folgt das Prädikat und diesem das Objekt. Subjekt, Prädikat und Objekt werden unter dem Begriff *Satzglied* zusammengefasst. Die Bezeichnungen der Satzglieder sind in spitze Klammern eingeschlossen. Die eckigen Klammern zeigen an, dass das Objekt fehlen kann.

Für Programmiersprachen werden ähnliche Regeln vereinbart, nach denen u.a. Kommandos und Befehle komponiert werden, z.B. das Kommando “17+239=”. Es handelt sich um *Vereinbarungen*. Für das Additionskommando könnte ebenso gut

vereinbart werden, dass das Operationszeichen nicht zwischen den Operanden, sondern ihnen voran- oder nachzustellen ist. Die Rechnerhardware muss entsprechend aufgebaut bzw. die Software entsprechend programmiert sein. Befehle von *Maschinenprogrammen* für *Drei-Adress-Maschinen* (das sind Rechner, deren *Maschinenbefehle* drei Adressen enthalten) sind i.Allg. nach der Regel

<Befehl> → <Operationscode> <Operandenadresse1> <Operandenadresse2>

<Resultatadresse> (5.2)

aufgebaut (Code und Adressen in einer Zeile). Sogenannte *Assemblersprachen* gestatten die Verwendung von Namen anstelle von Adressen. Gemäß der angegebenen Regel würde z.B. der Befehl “ADD x y z” der Variablen z den Wert der Summe x+y zuweisen. Computer können i.Allg. nur syntaktisch richtige, der Mensch kann auch “verdrehte” Sätze verstehen.

Die Syntaxregel für Aussagesätze stellt eine Aussage über Aussagen dar. In diesem Sinne sprechen wir von *Metaaussagen*. Allgemein heißt eine Sprache, die der Beschreibung einer Sprache dient, *Metasprache*, die beschriebene Sprache heißt *Objektsprache*. Begriffe einer Metasprache wie z.B. “Aussagesatz”, “Satzglied” oder “Subjekt” sind *metasprachliche* oder metalinguistische Begriffe. Sie spielen in einer Syntaxregel, wo sie durch spitze Klammern gekennzeichnet werden, die Rolle von Platzhaltern für alle möglichen Wörter oder Wortgruppen der Objektsprache. Darum heißen sie auch **metasprachliche Variable**.

Satzglieder können oft weiter dekomponiert werden. Das Subjekt kann z.B. aus einem Artikel, einer Substantivform und Attributen bestehen gemäß der Syntaxregel

<Subjekt> → [<Artikel>] {<Attribut>} <Substantivform>. (5.3)

Die geschweiften Klammern bedeuten, dass die Anzahl der Attribute beliebig ist. Für Substantivformen gilt die Syntaxregel

<Substantivform> → [<Präfix>] <Stamm> [<Suffix>] [<Endung>]. (5.4)

Es ergibt sich eine hierarchische syntaktische Struktur.

Syntaxregeln können eine Verkürzung (Verdichtung) sprachlicher Ausdrücke bewirken. Das lässt sich recht eindrucksvoll am Beispiel der Zahlendarstellung veranschaulichen. So ist die arabische Darstellung (Dezimalsystem) i.Allg. kürzer als die römische, und diese ist kürzer als die Darstellung mittels Strichen oder Kerben (Unärsystem). Für die Darstellung einer zweistelligen Dezimalzahl braucht man im römischen System bis zu 8 Zeichen (LXXXVIII) und im unären System bis zu 99 Zeichen. Das hat zwei Gründe. Zum einen steht eine unterschiedliche Anzahl elementarer Zeichen (Ziffern) zur Verfügung, nämlich 10 bzw. 7 bzw. ein einziges. Zum anderen spielt der Platz einer Ziffer in einer Zahl eine unterschiedliche Rolle. Im Unärsystem spielt sie gar keine und im römischen System eine sehr eingeschränkte Rolle. Hier ist die Reihenfolge im wesentlichen vorgegeben. Beispielsweise darf

auf die Ziffer V nur die Ziffer I oder das Ende der Zahl folgen. Dagegen gibt der Platz (die "Stelle") einer Ziffer in einer Dezimalzahl die Zehnerpotenz der "Stelle" an. Tauscht man zwei nicht identische Ziffern einer Dezimalzahl aus, ergibt sich eine andere Zahl. Dies ist ein Beispiel dafür, wie aus der Syntax eines Kompositzeichens auf dessen Bedeutung geschlossen werden kann.

Ähnlich ist es bei der Satzsyntax. Wenn Subjekt und Prädikat ihre Plätze wechseln, wird aus einem Aussagesatz ein Fragesatz. Um das richtige Interpretieren zu sichern oder zu erleichtern, wird am Satzende die Stimme gehoben bzw. ein Fragezeichen gesetzt. Dadurch wird auch ein "verdrehter" Fragesatz wie z.B. "Hunde jagen Katzen?" richtig interpretierbar.

Syntaxregeln der natürlichen Sprachen sind keine strengen Regeln; man darf eventuell gegen sie verstoßen. Das gilt auch für die Syntax des Aussagesatzes. Beispielsweise lässt die deutsche Sprache auch die umgekehrte Interpretation des Satzes "Hunde jagen Katzen" zu im Sinne "Hunde werden von Katzen gejagt". Denn Subjekt und Objekt dürfen ihre Plätze wechseln, und aus den Endungen der Substantiva "Hunde" und "Katzen" geht nicht hervor, welches im Nominativ und welches im Akkusativ steht. Durch die Anwendung des Passivs wird die Bedeutung eindeutig.

Nichteindeutigkeiten werden in der Regel durch den Kontext, durch die Stimmführung oder durch Satzzeichen aufgehoben. Doch erschweren sie demjenigen das Verständnis sehr, der versucht, einen fremdsprachigen Text mit Hilfe von Lexikon
4 und Grammatik zu übersetzen. Er muss mit einer *Syntaxanalyse* beginnen und jeden Satz zerlegen, d.h. er muss mit Hilfe der Syntaxregeln schrittweise seinen *Syntaxbaum* konstruieren. Bild 5.2 zeigt den Syntaxbaum des Satzes von Bild 5.1. Dabei sind die Knoten des dortigen Graphen (die Punkte) mit den entsprechenden metasprachlichen Variablen benannt.

Rechner können den Kontext i.d.R. nicht berücksichtigen. Darum müssen Syntaxregeln streng befolgt werden. Die Zahlendarstellung hatte bereits gezeigt, dass die Bedeutung eines Kompositzeichens mit dessen syntaktischem Aufbau gekoppelt sein kann. Das gilt auch für Sätze. Aufbau und Bedeutung eines Satzes stehen miteinander in Beziehung und zwar über die abstrakte (metasprachliche) Bedeutung der Satzglieder (der metasprachlichen Variablen). Die abstrakte Bedeutung des Prädikats besteht beispielsweise darin, dass es dem Subjekt ein Merkmal (Eigenschafts- oder Tätigkeitsmerkmal) zuweist.

Die Beziehung zwischen Aufbau und Bedeutung eines Satzes ist Voraussetzung dafür, dass man ihn verstehen kann. Will man einen Satz aus einer Sprache, die man nicht beherrscht, übersetzen, muss man ihn zunächst syntaktisch analysieren. Darauf wird in Kap.16.4 eingegangen.

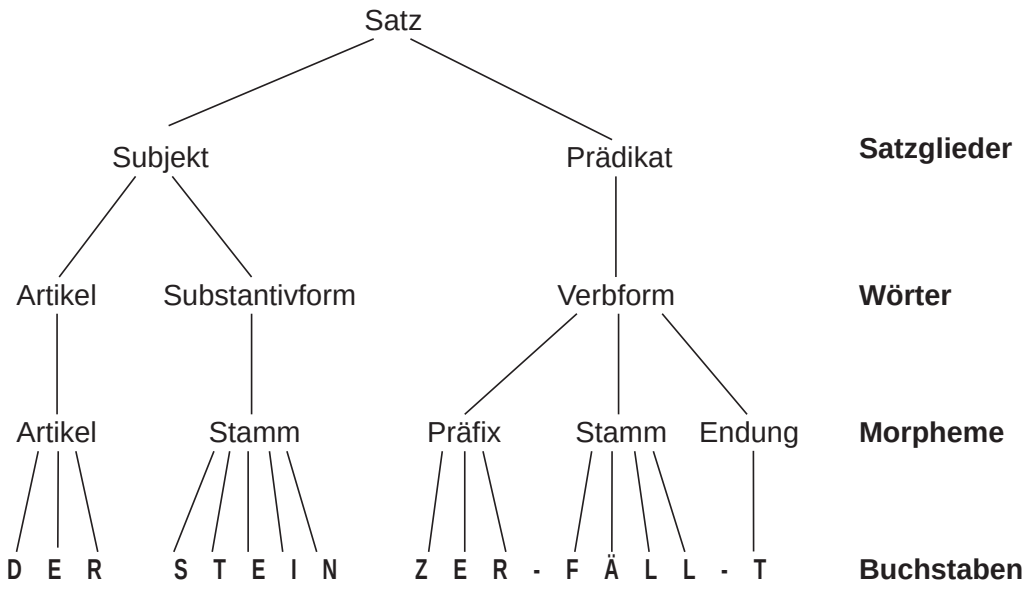

Bild 5.2 Syntaxbaum eines Aussagesatzes

5.4 Externe, interne und formale Semantik

Wir wenden uns nun der dritten der oben genannten Wirkungen des Selektionsdruckes zu, der *Idemobjektivierung*. In letzter Konsequenz hat sie zur Entwicklung der Mathematik geführt.

Vielen Lesern wäre es vielleicht natürlicher, nicht von Idemobjektivierung, sondern von der Herausbildung einer subjektunabhängigen *Semantik*, also von "Semantikobjektivierung" oder von "semantischer Objektivierung" zu sprechen. Wir haben den Semantikbegriff bisher bewusst vermieden, weil er vieldeutig ist und leicht zu Missverständnissen führt. Mit *Semantik* wird zum einen die Bedeutung sprachlicher Einheiten, zum anderen die Lehre von der Bedeutung sprachlicher Einheiten bezeichnet. Wir werden das Wort in der Regel im erstgenannten Sinne verwenden, jedoch mit folgender Präzisierung: *Die **Semantik** eines Kompositzeichens einer Sprache ist* 5
dessen Bedeutung unter der stillschweigenden Annahme, sie sei für alle Nutzer der Sprache die gleiche bzw. Unterschiede seien vernachlässigbar. Die Ausmerzung vorhandener und die Vermeidung möglicher Unterschiede ist das Ziel der Idemobjektivierung, die wir im Weiteren auch **semantische Objektivierung** nennen wer- 6
den. Sie ist die Voraussetzung der popperschen Welt 3..

In der technischen Informationsverarbeitung ist die Unmissverständlichkeit besonders wichtig, denn als Interpretierer sprachlicher Ausdrücke fungieren nicht nur Menschen, sondern auch Computer, und denen muss absolut eindeutig gesagt werden, was sie zu tun haben. Die Folge ist, dass in der Rechentechnik der Semantikbegriff i.Allg. nicht aus der Sicht des Nutzers, sondern aus der Sicht des Computers definiert wird. Wenn ein Informatiker, der sich mit der Entwicklung von Program-

miersprachen oder Compilern beschäftigt, allgemeinverständlich erklären soll, was
7 unter Semantik zu verstehen ist, wird er vielleicht sagen: *"Die **Semantik** von Befehlen oder Programmen ist deren **Wirkung** im Computer, es ist der durch sie ausgelöste Prozess bzw. dessen Resultat, es ist ihre **Interpretation** durch den Computer."*

Diese Begriffsbestimmung bindet die Semantik an den Computer, an das System, in dem der Interpretierungsprozess abläuft, das ihn "trägt". Darum sprechen wir von trägerinterner oder kurz von **interner Semantik**. Wenn man dagegen von der Semantik eines umgangssprachlichen Ausdrucks spricht, meint man nicht deren trägerinterne Semantik, also nicht irgendwelche neuronalen Prozesse im Gehirn. Nach unserer Charakterisierung natürlicher Sprachen als Mittel der Modellierung der Umwelt bedeutet ein Aussagesatz einen Sachverhalt, der in der Außenwelt gegeben ist. Darum sprechen wir von **externer Semantik**.

Es erhebt sich nun folgendes Problem. Wie kann der Mensch, der externsemantisch denkt und spricht, einem Computer, der internsemantisch "versteht" und
8 "denkt", Aufträge erteilen? Er muss nämlich beim Programmieren die *semantische Lücke* zwischen externer und interner Semantik gedanklich *überspringen*. Dies ist das Kernproblem des Programmierens, man kann sogar sagen das *Kernproblem der technischen Informatik*; wir nennen es das **technische Semantikproblem**. Wir werden uns ausführlich mit ihm beschäftigen müssen (siehe Kap.15.5). Hier soll nur auf den besonderen Fall eingegangen werden, dass die Lücke praktisch nicht existiert.

Wenn man beispielsweise in seinen Taschenrechner das *Kommando* "17×239=" eintastet, braucht man keinerlei "Semantiksprünge" zu machen, denn der Rechner "versteht" die Sprache der Arithmetik. Er kann arithmetische Ausdrücke interpretieren, weil die arithmetischen Operationen im Rechner hardwaremäßig "verdrahtet" sind. Es existieren die entsprechenden Operatoren, und eine sinnreiche Steuerung des Datenflusses sorgt dafür, dass die Operanden dem geforderten Operator zugeführt werden und das Resultat im Anzeigefeld erscheint.

Ergänzt man die arithmetischen durch einige *logische Operatoren*, so bedarf es weiter keiner Hardwareoperatoren für die Verarbeitung von Zeichenketten, um aus der "Rechentechnik" eine "Denktechnik" zu machen, um künstliche Intelligenz zu fabrizieren. Das erscheint auf den ersten Blick unglaublich, und man fragt sich, wie das möglich sein soll. Diese Frage wird im Weiteren Schritt für Schritt beantwortet.

Die scheinbare Nichtexistenz der semantischen Lücke bei der Nutzung eines Taschenrechners hat ihre Ursache in der impliziten Verwendung einer dritten Art von Semantik, die uns ganz besonders interessieren wird, denn es handelt sich um das Ergebnis uralter zielstrebiger Bemühungen der Menschen um einen exakten Sprachgebrauch zum Zwecke der Vermeidung von Missverständnissen, also um Idemobjektivierung. Um zu erreichen, dass alle Kommunikationspartner Mitteilungen identisch interpretieren, werden *formale Sprachen, Kalküle* und *formale Theorien* konstruiert.

9 *Eine **formale Sprache** ist eine Menge elementarer, semantikfreier Zeichen (Alphabet) zusammen mit einer Menge von Syntaxregeln*. Sämtliche Vereinbarungen

und Regeln sind exakt zu befolgen. *Ein **Kalkül** ist eine formale Sprache zusammen mit einer Menge von Transformations- oder Rechenregeln zum Umformen (Transformieren) von Ausdrücken der formalen Sprache*. Die Regeln legen fest, welche Ausdrücke in welche überführt (durch welche *substituiert*) werden dürfen. Beispielsweise sind 3+5=8 oder 3×5=15 Rechenregeln der Arithmetik, des *arithmetischen Kalküls*. Üblicherweise wird in Transformationsregeln der Pfeil anstelle des Gleichheitszeichens verwendet und z.B. "3+5→8" notiert. Ein anderes Beispiel ist die trigonometrische Transformationsregel sinx/cosx → tanx.

Die Formulierung derartiger Regeln beinhaltet eine Zuordnung bestimmter Bedeutungen zu den Zeichen der Sprache, beispielsweise die Festlegung, dass "+" das Operationszeichen der Addition ist. Dadurch werden Zeichen zu *Symbolen* und die zunächst völlig semantikfreie formale Sprache wird zu einer *Kalkülsprache*. Die 10
Bedeutungszuweisung heißt kalkülspezifische oder **formale Interpretation**. *Die zugeordnete Semantik heißt kalkülinterne oder **formale Semantik**. Eine formal (kalkülspezifisch) interpretierte formale Sprache heißt **Kalkülsprache***.

Um einen Kalkül für die sprachliche Modellierung eines Ausschnitts der Welt, eines "externen" (außerhalb des Kalküls existierenden) Originalbereichs verwenden zu können, müssen die Objekte des Originalbereichs sowie Aussagen über diesen Bereich Entsprechungen im Kalkül erhalten, m.a.W. der Kalkül muss durch den 11
Originalbereich *interpretiert* werden. Diese Interpretation nennen wir **externe Interpretation**. *Ein Kalkül, der durch einen Ausschnitt der Realität interpretiert ist, heißt **formale Theorie** des Realitätsausschnitts*. Diese Bezeichnung trifft z.B. auf physikalische Theorien zu. Durch externe Interpretation wird die *reine* zur *angewandten Mathematik*. Wenn nichts Gegenteiliges gesagt wird, ist im Weiteren unter Interpretation stets *externe* Interpretation zu verstehen. In der Mathematik wird unter Interpretation in der Regel die formale Interpretation verstanden.

Externe Interpretation kann auf unterschiedlichen Abstraktionsniveaus erfolgen. Beispielsweise kann ein Zeichen einen Gegenstand, eine Energie, eine Wahrscheinlichkeit, den Wahrscheinlichkeitswert einer Aussage oder noch abstraktere Inhalte bezeichnen. Aber selbst wenn ein konkreter Gegenstand der realen Welt bezeichnet wird, handelt es sich im Modell bereits um ein abstraktes Objekt. Newton musste, um seine Theorie der Planetenbewegung aufzubauen, den Begriff des Planeten - also auch den der Erde - auf den Begriff des Massenpunktes reduzieren und die Erde (ebenso wie die Planeten und die Sonne) durch einen einzigen metrischen Merkmalswert, ihre Masse beschreiben, abgesehen von den Zustandsmerkmalen Ort und Impuls. Dem stelle man den *Assoziationskomplex* gegenüber, den man mit dem Wort "Erde" in dem Satz "Die Erde war wüst und leer" verbindet. Man hat es hier mit einer immensen bedeutungsmäßigen Verkürzung, mit einer *Idemschärfung* zu tun, ohne welche die notwendige semantische Objektivierung nicht möglich gewesen wäre. Sie führt letzten Endes auf Zahlen, also auf Ideme, die für alle Menschen identisch sind und die wir darum *universelle Ideme* nennen. Sie sind die Voraussetzung für die mathematische Modellierung der Welt.

12 Die verwendete Methode, also das Verkürzen der Bedeutung um nichtrelevante (für das Modellierungsziel unwichtige) Inhalte, wird als *idealisierendes Abstrahieren* oder kurz *Idealisieren* bezeichnet. Idealisieren ist von fundamentaler Bedeutung, denn jeder Mensch, nicht nur der Mathematiker, idealisiert ständig beim Artikulieren von Idemen. ***Idealisierende Abstraktion*** *ist die Voraussetzung sprachlicher Modellierung und sprachlicher Kommunikation.*

Es ist eine ergänzende Bemerkung zum Kalkülbegriff erforderlich, denn die oben gegebene Definition ist insofern nicht ganz befriedigend, als sie nichts über den Umfang der Menge der Rechenregeln aussagt. Welche Regeln müssen beispielsweise aufgelistet werden, um die Arithmetik oder die Geometrie als Kalkül zu definieren. Das Problem wäre gelöst, wenn eine endliche Menge von Regeln angegeben werden könnte, aus denen sich alle anderen Regeln ableiten lassen. Diese Vorstellung führte zur sog. *Axiomatisierung* von Kalkülen und zu ihrer Definition mittels *Axiomensystemen. Ein* ***Axiomensystem*** *ist eine Menge voneinander unabhängiger und untereinander widerspruchsfreier Rechenregeln (allgemein von Sätzen), aus denen sich sämtliche Rechenregeln (sämtlich Sätze) eines Kalküls ableiten lassen.*

So lassen sich z.B. alle Aussagen der euklidischen Geometrie aus den von EUKLID formulierten Axiomen ableiten, und alle Aussagen über das Verhalten eines Systems von Massenpunkten, allgemein alle Aussagen der newtonschen Mechanik, lassen sich aus den newtonschen Axiomen ableiten. *Ein Kalkül mit Axiomensystem heißt* ***axiomatisierter Kalkül***.

Wir fassen die getroffenen Vereinbarungen noch einmal mit anderen Worten zusammen (vgl. Bild 5.3): *Wir sprechen von* ***externer*** *bzw.* ***interner*** *bzw.* ***formaler Semantik***[1] *eines sprachlichen Ausdrucks, wenn dieser bedeutungsmäßig an die Außenwelt bzw. an die Struktur und Funktion des Trägers bzw. an ein Kalkül*

Bezeichnung der Semantik eines sprachlichen Ausdrucks	Bedeutungsmäßige Anbindung des sprachlichen Ausdrucks
extern	an die Umwelt des modellierenden Menschen
formal	an ein Kalkül
intern (trägerintern)	an die Struktur und Funktion des Modellträgers

Bild 5.3 Drei Semantiken

1 Die Bezeichnung *formale Semantik* wird in der Literatur zuweilen in einem anderen, spezielleren Sinne verwendet.

angebunden ist. Abschließend sei eine im Grunde selbstverständliche Bemerkung angefügt. Grundlage des exakten, d.h. formalisierten oder genauer kalkülisierten 13
Modellierens ist die Mathematik, denn das Erfinden von und Hantieren mit Kalkülen ist Gegenstand der Mathematik.

5.5 Begriffsbildung

Begriffsbildung[2] und Grammatik tragen dazu bei, die Ausdrucksstärke einer Sprache zu erhöhen und so die Kommunikation zwischen den Kooperationspartnern schneller und flexibler zu gestalten. Anstelle von Ausdrucksstärke werden wir auch von *semantischer Dichte* sprechen. Dieser Begriff ist, wenn er sich auf natürliche Sprachen bezieht, ebenso wie der Idembegriff, introspektiv und insofern unscharf definiert. *Die **semantische Dichte** eines natürlichsprachigen Textes ist durch den* 14
Umfang des im Mittel pro Wort assoziierbaren Gedächtnisinhaltes (Menge der assoziierbaren Ideme) gegeben. Die semantische Dichte hängt also wesentlich vom Empfänger, von dessen Erfahrung und Wissen, von seiner Aufnahmefähigkeit und Phantasie und, wenn man die emotionale Komponente von Idemen einbezieht, auch von seinem Empfindungs- und Einfühlungsvermögen ab, was z.B. beim Interpretieren von Kunstwerken ganz entscheidend sein kann. Die semantische Dichte von Programmiersprachen bzw. Programmen lässt sich schärfer definieren, nämlich als *mittlere Anzahl elementarer Operationen (ALU-Operationen) pro Zeichen*.

Die *Grammatik* kann eine semantische Verdichtung durch Optimierung der Syntaxregeln mit dem Ziel einer Minimierung der Artikulierungen (der mittleren Länge der Zeichenrealeme) erreichen. Die *Begriffsbildung* kann eine semantische Verdichtung dadurch bewirken, dass Begriffe mit hoher semantischer Belegung gebildet werden.

*Ein **Begriff** ist ein objektiviertes, benanntes Idem.* Seine semantische Belegung ist umso höher, je mehr Gedächtnisinhalte der Interpretierer dem codierenden Zeichenrealem zuordnet, je mehr er mit ihm "assoziiert". Wörter mit hoher semantischer Belegung sind z.B. "Tierreich", "Erde", "Gott". In Kap.5.4 [12] hatten wir die Umkehrung, die Verkürzung der semantischen Belegung, die Idemschärfung durch *idealisierende Abstraktion* zum Zwecke der *semantischen Objektivierung* besprochen. Jetzt wenden wir uns den Methoden der semantischen Verdichtung durch solche Begriffsbildungen zu, die weder unscharf sind, noch der Idemschärfung dienen, sondern der sprachlichen Modellierbarkeit der unendlichen Welt mittels Abstraktion. ***Abstraktion** ist die einzige Möglichkeit, die unendliche Welt sprachlich zu modellieren.*

---

2 Zur Begriffsbildung siehe z.B. [Klix 80].

Bei den folgenden Methoden der Begriffsbildung handelt es sich um gedankliche Zusammenfassungen mehrerer, eventuell auch sehr vieler Objekte nach bestimmten Gesichtpunkten, wobei die Zusammenfassungen mit Namen *benannt* werden. Zwei Beispiele sollen das verdeutlichen. Durch die Sätze

Hose, Rock und Weste bilden einen Anzug

Hose, Rock und Weste sind Kleidungsstücke

werden drei Objekte nach unterschiedlichen Gesichtspunkten zusammengefasst und die Zusammenfassungen werden mit “Anzug” bzw. “Kleidungsstück” benannt. Der erste Satz bildet den Begriff “Anzug” dadurch, dass die Bestandteile eines Anzuges aufgezählt werden. Der Begriff “Anzug” wird durch Komponierung gebildet. Darum sprechen wir von **komponierender Abstraktion**. Im zweiten Satz werden die Begriffe Hose, Rock und Weste zum Begriff Kleidungsstück verallgemeinert oder, wie auch gesagt wird, unter dem Begriff Kleidungsstück *subsumiert*. In diesem Fall sprechen wir von *generalisierender Abstraktion* oder kurz von **Generalisierung**.

Es ist zu beachten, dass im zweiten Satz mit Hose, Rock und Weste keine konkreten Objekte, sondern Mengen (Klassen) von Objekten bezeichnet sind. Aber auch im ersten Satz sind keine kokreten Hosen, Röcke und Westen gemeint. Genauer müsste es heißen: Je ein Kleidungsstück von Typ Hose, Typ Rock und Typ Weste oder je ein Kleidungsstück der Klassen Hose, Rock und Weste bilden einen Anzug. Das Bilden von Begriffen ist eine gedankliche Tätigkeit und die Begriffe, mit denen gearbeitet wird, sind Ideme mit objektivierten Merkmalen, d.h. mit Merkmalen, die den Begriffen von allen Beteiligten einheitlich (mit der gleichen Bedeutung) zugeordnet werden, wie z.B. das Merkmal “ärmellos” dem Begriff “Weste”.

Um eine kompakte Beschreibung der verschiedenen Methoden der Begriffsbildung zu ermöglichen, führen wir die Begriffe Klasse und Typ ein. Eine **Klasse** ist die begriffliche Zusammenfassung von Objekten (den Elementen der Klasse), die in einem oder mehreren **Merkmalen** übereinstimmen, also dieselben *Merkmalsausprägungen* oder **Merkmalswerte** besitzen; mit anderen Worten: *eine **Klasse** ist eine Menge, die durch ein oder mehrere **Merkmalswerte** festgelegt ist, die alle Elemente der Menge besitzen*. Beispielsweise sind “rot” und “grün” Ausprägungen (Werte) des Merkmals “Farbe”. Umgangssprachlich wird i.Allg. nicht zwischen Merkmal und Merkmalswert unterschieden. Wenn Missverständnisse ausgeschlossen sind, wird auch in diesem Buch gelegentlich nicht zwischen Merkmal und Merkmalswert unterschieden. Von einem Element einer Klasse sagt man, dass es der betreffenden Klasse *angehört* oder dass es vom **Typ** der betreffenden Klasse ist. Beispielsweise gehört die Zahl π, aufgerundet gleich 3.1416, der Klasse der reellen Zahlen an, sie ist vom Typ der reellen Zahlen.

Mit Hilfe des Klassenbegriffs sind wir in der Lage, die Begriffsbildungsmethoden, die in der ersten Spalte von Bild 5.4 aufgeführt sind, kurz und klar zu beschreiben.

Klassifizierendes Abstrahieren *(kurz: Klassifizieren) ist das Zusammenfassen mehrerer Objekte, die in einem oder in mehreren Merkmalen übereinstimmen, zu*

einem neuen abstrakten Objekt, Klasse genannt. Das Klassifizieren von Daten wird in der Informatik auch **Typisieren** genannt. Vom Klassifizieren (wörtlich übersetzt: Klassen machen) ist das *Klassieren* zu unterscheiden. *Unter **Klassieren** verstehen wir das Einordnen eines Objektes in eine bereits existierende Klasse.*. Umgangssprachlich wird anstelle des Wortes *Klassieren* i.d.R. das Wort *Identifizieren* benutzt, beispielsweise, wenn man einen Baum als Buche "identifiziert" (klassiert). Klassifizieren und Klassieren beinhalten *Idealisieren* insofern, als beim Definieren einer Klasse bzw. beim Einordnen eines Objekts in eine Klasse alle nichtrelevanten Merkmale vernachlässigt werden, von ihnen wird "abstrahiert". In der Umgangssprache wird im allgemeinen nicht zwischen Klassifizieren und Klassieren unterschieden, sondern in beiden Fällen von Klassifizieren gesprochen. Wir schließen uns diesem Sprachgebrauch an, solange Missverständnisse ausgeschlossen sind.

***Komponierendes Abstrahieren** ist das Bilden einer Kompositklasse aus Komponentenklassen. Aus den Elementen der Komponentenklassen werden Elemente der Kompositklasse komponiert.* Im obigen ersten Beispielsatz sind Hose, Rock und Weste die Namen der Komponentenklassen und Anzug ist der Name der Kompositklasse. Komponierende Begriffsbildung liegt vor, wenn im Rahmen einer Komponierungshierarchie ein neues Komposit gebildet wird, das dann die Rolle eines selbständigen Denkobjekts spielt. Das komponierende Abstrahieren wird beim Komponieren von Operatorenhierarchien eine wichtige Rolle spielen. Auf höheren Komponierungebenen wird Komponieren zuweilen als *Aggregieren* bezeichnet. 16

***Generalisierendes Abstrahieren** ist das Zusammenfassen mehrerer Klassen, **Unterklassen** genannt, zu einer einzigen Klasse, **Oberklasse** genannt. Generalisieren erfolgt durch Außerachtlassung (Streichung) eines oder mehrerer Merkmale.* In dem obigen zweiten Beispielsatz sind Hose Rock und Weste die Namen der Unterklassen und Kleidungsstück ist der Name der Oberklasse. Durch Generalisierung werden z.B. die Klassen der rationalen und irrationale Zahlen zur Klasse der reellen Zahlen zusammengefasst durch Streichung des Merkmals "Darstellbarkeit als Bruch ganzer Zahlen", das für Rationalzahlen den Wert "darstellbar", für Irrationalzahlen den Wert "nichtdarstellbar" hat.

Man beachte, dass jede der drei genannten Arten des Abstrahierens *Merkmalsbildung* voraussetzt und *idealisierendes* Abstrahieren einschließt. Ohne Merkmale und Merkmalswerte und ohne Idealisieren, ohne Verzicht auf Beschreibungsdetails ist keine Klassenbildung möglich. Mit dem Erfassen von Merkmalen beginnt das sprachliche Modellieren der Welt durch ein Kind, sicherlich bereits vor seiner Geburt. Bei der Bildung neuer Denkobjekte und Begriffe finden Merkmalsbildung und Klassenbildung häufig gleichzeitig statt.

In Bild 5.4 sind die vier Arten des Abstrahierens zusammengestellt. Sie können als vier *begriffsbildende Operationen* aufgefasst werden. In der letzen Zeile ist das *Benennen* hinzugefügt, jedoch separat, denn das Benennen ist kein Abstrahieren, wohl aber eine Komponente der Begriffsbildung, denn Begriffe sind *benannte* objektivierte Ideme.

Begriffsbildende Operation		Aufwärtsrelation
aufwärts: Abstrahieren	abwärts: Konkretisieren	Übergang von — nach beim Abstrahieren
Idealisieren	Realisieren	der Realität näher — ferner
Klassifizieren, Typisieren	Instanzieren, Ausprägen	Exemplar — Klasse, Typ
Komponieren, Aggregieren	Dekomponieren	Teil — Ganzes
Generalisieren	Präzisieren	Unterklasse — Oberklasse
Benennen, Referenzieren	Dereferenzieren	Benanntes — Name

Bild 5.4 Begriffsbildende Operationen

In der letzten Spalte ist die jeweilige Art des Abstrahierens durch einen begrifflichen Übergang "von — nach" charakterisiert, beispielsweise das komponierende Abstrahieren durch den Eintrag "Teil — Ganzes".

In der zweiten Spalte sind die Operationen des *Konkretisierens* genannt, die dem Abstrahieren entgegengerichtet sind und das Abstraktionsniveau erniedrigen. Die Bezeichnungen der Abwärtsoperationen bedürfen kaum einer Erläuterung. Das *Ausprägen* oder *Instanzieren* entspricht dem Übergang von einer Klasse zu einem Element der Klasse. Der letzte Schritt des Instanzierens führt zu einem *konkreten* (materiellen) Objekt, zu einem *Exemplar* der betreffenden Klasse. In diesem Fall hatten wir in Verbindung mit Bild 2.1 von *materiellem Instanzieren* gesprochen. Dort war auch auf die Herkunft des Wortes "Instanzieren" eingegangen worden.

Wegen der großen Bedeutung des Generalisierens und Präzisierens im Bereich der Wissenschaft und speziell der Informatik soll auf diese Art der Begriffsbildung etwas ausführlicher eingegangen werden. Durch schrittweises Generalisieren und/oder Präzisieren lässt sich eine Hierarchie abstrakter Begriffe aufbauen. Ein Schulbeispiel ist die Systematik des Tierreiches, d.h. seine Unterteilung durch Präzisierung in Unterreiche, Stämme, Arten usw. In einem Abwärtsschritt in der Hierarchie wird die Beschreibung der zu klassifizierenden (systematisierenden) Objekte (Tiere) durch Hinzunahme eines oder mehrerer Merkmale präzisiert. Bei einem Aufwärtsschritt wird sie durch Streichung eines oder mehrerer Merkmale verallgemeinert (generalisiert).

Das Aufbauen einer Begriffshierarchie und das Arbeiten mit ihr soll am Syntaxbaum von Bild 5.2 veranschaulicht werden. Der Sprachunkundige muss beim Übersetzen des Satzes diesen zunächst syntaktisch analysieren, d.h. er muss die einzelnen Teile (die Bausteinrealeme) des Satzes als bestimmte Wort- bzw. Satzglieder erkennen (z.B. die Wortgruppe “Der Stein” als Satzsubjekt), er muss jedem Knoten des Graphen den richtigen grammatikalischen (metasprachlichen) Begriff zuordnen. Der so entstehende Syntaxbaum enthält zwei orthogonal zueinander liegende Hierarchien, eine senkrechte, komponierende Hierarchie mit vier Ebenen (die Buchstabenebene nicht mitgezählt) und eine waagerechte, generalisierende Hierarchie mit zwei Ebenen. Beim Generalisieren werden die Begriffe einer Ebene zu einem Oberbegriff (fett gedruckt in der rechten Spalte) zusammengefasst, d.h. aus Klassen werden Oberklassen gebildet. Beim Komponieren werden Kompositklassen der nächsthöheren Ebene gebildet. Für den jeweils obersten Begriff der Komponierungs- bzw. Generalisierungshierarchie gelten die folgende Aussagen:

Subjekt, Prädikat und Objekt bilden einen Satz.

Subjekt, Prädikat und Objekt sind Satzglieder.

Die beiden Sätze demonstrieren - ebenso wie die beiden Beispielsätze über Kleidungsstücke - das Komponieren (erster Satz) und das Generalisieren (zweiter Satz).

Der recht komplizierte Prozess der Syntaxanalyse, d.h. das Einordnens der einzelnen Wörter in metasprachliche Klassen, läuft im Gehirn jedes Menschen ab, der Sprache interpretiert. Doch falls der Interpretierer die Sprache perfekt beherrscht, braucht sein Oberbewusstsein dabei nicht bemüht zu werden, die Analyse ist reflektorisch überbrückt.

Der schrittweise Aufbau des Syntaxbaumes zeigt noch einmal sehr deutlich, wie Syntax und Semantik miteinander verflochten sind. Denn dadurch, dass man eine aus dem Satz herausgeschnittene Buchstabenkette als ein bestimmtes Wort- oder Satzglied erkennt, weist man ihr bereits “ein Stück Semantik” zu. Wenn z.B. die Kette “Der Stein” als Subjekt erkannt ist, dann folgt daraus, dass der Satz eine Aussage über den Stein macht.

Grundlage der Begriffsbildung ist, wie wir gesehen haben, die Beschreibung 17
(sprachliche Modellierung) von Objekten der Realität oder des Denkens (von Realemen oder Idemen) durch *Merkmalswerte*. Das gedankliche Verbinden (Assoziieren) bestimmter Gegenstände mit bestimmten Eigenschaften (und umgekehrt) ist ein wesentlicher Bestandteil der Anpassung des Menschen an seine Umwelt. Das Ergreifen und Begreifen der Welt durch den Säugling beginnt mit der Unterscheidung von Gegenständen aufgrund unterschiedlicher Eigenschaften wie Temperatur, Festigkeit, Farbe und Geruch, über die ihn seine Sinnesorgane informieren. Im Weiteren werden wir sehen, wie das *sprachliche Modellieren mittels Merkmalen und Merkmalswerten* zum Grundprinzip der Wissensrepräsentation in der technischen Informationsverarbeitung, speziell in der Datenbanktechnik [16.5] geworden ist. Es sei noch einmal

wiederholt, dass Begriffe durch Abstraktion entstehen und insofern Ideme sind, die
18 sich durch Merkmale auszeichnen. Wenn zum Ausdruck gebracht werden soll, dass ein Idem ein durch Merkmale charakterisiertes gedachtes Objekt ist, nennen wir es **Denkobjekt**. Das *Denkobjekt* der Human-IV (Informationsverarbeitung durch den Menschen) entspricht dem *Datenobjekt* in der Computer-IV (Informationsverarbeitung durch den Computer). Verallgemeinernd kann man sagen: ***Sprachliches Modellieren** ist das Hantieren mit Denkobjekten bzw. mit Datenobjekten und damit ein Operieren mit **Merkmalswerten**.*

In diesem Zusammenhang spielt der Begriff der *Assoziation* eine wichtige Rolle. Wir haben ihn wiederholt verwendet, ohne ihn zu definieren. In Kap.7 werden wir uns überlegen, wie der Assoziationsbegriff in der Human-IV im Zusammenhang mit der *menschlichen* Intelligenz verwendet wird. Hier geben wir eine Begriffsbestim-
19 mung, die sowohl in der Human-IV als auch in der Computer-IV anwendbar ist. *Unter **Assoziieren** verstehen wir das Aufrufen von Objekten (Denkobjekten oder Datenobjekten) über ihre Merkmalswerte oder das Aufrufen von Merkmalswerten über Objekte, welche sich durch diese Merkmalswerte auszeichnen, oder das Aufrufen mittels Kombination dieser beiden Aufrufmethoden.* Beim Aufrufen aus dem menschlichen Gedächtnis spricht man von “ins Bewusstsein rufen”, beim Aufrufen aus einem Computerspeicher von “zugreifen auf”.

Drei Beispiele sollen die verschiedenen Methoden des Assoziierens illustrieren. Wenn ich beim Anblick eines Bekannten an dessen Geburtsdatum erinnert werde, assoziiere ich mit einem Objekt einen Merkmalswert. Wenn mich die Beschreibung einer Person in einem Roman an einen Bekannten erinnert, assoziiere ich mit Merkmalswerten ein Objekt. Wenn mich eine Wolke an einen Hund erinnert, assoziiere ich mit einem Objekt ein anderes Objekt über gemeinsame Merkmale.

In Teil 3 werden wir sehen, welch bedeutende Rolle die begriffsbildenden Operationen in der Softwaretechnik spielen, konkret die komponierende Abstraktion als sog. *prozedurale Abstraktion* beim Aufbau sprachlicher Operatorenhierarchien (Kap.18.2 [18.3]) und die klassifizierende und generalisierende Abstraktion in der Datenbanktechnik (Kap.16.2 [16.7] und im objektorientierten Programmierparadigma (Kap.18.2 [18.4]).

An dieser Stelle unterbrechen wir unsere Überlegungen zu dem großen Thema “Gedanke und Sprache”, um eine Diskussion zum Begriff der “*syntaktischen Information*” einzuschieben. Er spielt in der Informatik-Literatur eine so große Rolle, dass wir ihn dem Leser nicht vorenthalten wollen, obwohl wir ihn nicht benötigen, um den Computer und die künstliche Intelligenz nachzuerfinden. Das Kapitel 5.6 kann ohne Beeinträchtigung der weiteren Lektüre übersprungen werden.

5.6* Umcodierungseffizienz und syntaktische Information

Die stürmische Entwicklung der Kommunikationstechnik und die explosionsartige Zunahme der Menge an Informationen, die über die Medien verbreitet werden, und andrerseits die zunehmende Formalisierung der Kommunikationssprachen (man denke an die vielen Fach- und Programmiersprachen), führt einem die gegenwärtigen Wirkungen des Selektionsdruckes, von dem zu Beginn des Kapitels 5.1 die Rede war, drastisch vor Augen. Freilich spielt dabei die bewusste, zielstrebige Tätigkeit der Menschen eine erheblich größere Rolle als bei der Herausbildung der natürlichen Sprachen, mit anderen Worten, aus dem Selektionsdruck werden technische Zielstellungen.

So wurde aus dem Druck in Richtung Beschleunigung der Kommunikation die Zielstellung, die Zeit zu verkürzen, die für die Übertragung einer Nachricht über einen *Nachrichtenkanal* notwendig ist. Dabei spielen zwei Faktoren eine Rolle,

- die semantische Dichte der Nachricht und
- die Anzahl der Zeichen, die der Kanal pro Zeiteinheit übertragen kann.

Der zweite Faktor hat eine technische Entwicklung ausgelöst, die durch zwei 20
Entscheidungen geprägt ist, die Entscheidung für binäre Codierung, also für ein Alphabet, das nur zwei Zeichen enthält, und die Entscheidung für elektromagnetische Wellen als Träger. Die Beschränkung auf zwei Zeichen hat in erster Linie ökonomische Gründe, denn je mehr Zeichen unterschieden werden müssen, umso höher wird der Aufwand, der getrieben werden muss, um die Störanfälligkeit des Nachrichtenkanals zu senken und die fehlerfreie Zeichenübertragung zu sichern. *Ein Zeichen des Binäralphabets heißt* ***Bit****,* so wie ein Zeichen des deutschen Alphabets *Buchstabe* heißt. Ein Nachrichtenkanal, der nur zwei verschiedene Zeichen, also Bits übertragen kann, heißt *Binärkanal*.

Die Nutzung elektromagnetischer Wellen ermöglicht die Annäherung an die maximal mögliche Geschwindigkeit, mit der sich Energie und folglich auch Nachrichten übertragen lassen, an die Lichtgeschwindigkeit. Dabei ist man bemüht, mit möglichst hoher *Trägerfrequenz* (Frequenz der tragenden Welle) zu arbeiten, denn je höher die Frequenz (genauer gesagt die *Bandbreite*) ist, umso mehr Zeichen können der Welle pro Zeiteinheit durch Modulation aufgeprägt und durch sie transportiert werden. Infolgedessen setzen sich immer mehr die sehr hochfrequenten Lichtwellen als Träger durch. Man ist heute in der Lage, Licht mit Hilfe von *Lichtleitern* über große Entfernungen und “um viele Ecken” dem Empfänger zuzuleiten. *Die Anzahl der Binärzeichen oder Bits, die ein Binärkanal pro Sekunde fehlerfrei übertragen kann, heißt (nicht sehr treffend)* ***Kanalkapazität****.*

Nach diesen kurzen Bemerkungen über die Entwicklungen zur Erhöhung der Kapazität von Nachrichtenkanälen wenden wir uns nun dem ersten oben genannten Faktor zu, der die Kommunikationsgeschwindigkeit mitbestimmt, der *semantischen Dichte*. Im vorangehenden Kapitel ist die semantische Verdichtung durch Begriffsbildung und in Kap.5.3 durch Syntaxregeln behandelt worden. In beiden Fällen wird

die semantische Verdichtung durch effektiveres *primäres* Codieren erreicht, d.h. dadurch dass mehr Ideme durch weniger Zeichenrealeme artikuliert werden. Jetzt interessieren wir uns dafür, wie Nachrichten, also bereits extern codierte Ideme, durch *Umcodierung* (durch *sekundäres* Codieren) verkürzt werden können, ohne dabei Semantik zu verlieren. Es handelt sich um eine rein *syntaktische Verdichtung*, d.h. um eine Erhöhung der Codierungseffizienz durch Umcodieren. Den Codierungsaufwand des Sekundärcodes zu dem des Primärcodes nennen wir **Umcodierungseffizienz**. Im Weiteren ist unter Codieren stets das sekundäre Codieren (Umcodieren) eines Zeichenrealems zu verstehen.

Je höher die Codierungseffizienz ist, umso kürzer ist das einem bestimmten Idem zugeordnete Zeichenrealem. Erhöhung der Codierungseffizienz bewirkt primär eine syntaktische und sekundär eine *semantische* Verdichtung. Ein Beispiel dafür ist der Übergang von der römischen zur arabischen Zahlendarstellung. Beim Umcodieren erniedrigt sich die mittlere Anzahl der Ziffern pro Zahl.

Soll eine arabische Zahl über einen Binärkanal übertragen werden, muss sie binär umcodiert werden, z.B. durch Übergang zu *Dualzahlen* oder dadurch, dass jede Ziffer durch ein zu vereinbarendes Binärwort ersetzt wird. Wenn eine Nachricht über einen Binärkanal übertragen werden soll, muss die gesamte Nachricht binär umcodiert werden, im einfachsten Fall dadurch, dass jedem Zeichen ein *Binärwort* zugeordnet wird, wie beispielsweise durch das *Morsealphabet*.

Beim Umcodieren lässt sich die Codierungseffizienz dadurch erhöhen, dass für häufiger auftretende Zeichen (Buchstaben), z.B. für das “e” und “n” im Deutschen, kürzere Codewörter reserviert werden. Umgekehrt kann man durch gezielte Verlängerung der Codewörter, durch Hinzunahme zusätzlicher Bits (durch Erhöhung der *Redundanz*, s.u.) Übertragungsfehler erkennen und die Zuverlässigkeit erhöhen, am einfachsten dadurch, dass jedes Zeichen wiederholt wird. Wo liegt der optimale Kompromiss zwischen Kürze und Sicherheit?

21 Die Technik verlangte exakte Antworten auf die gestellten Fragen. Auf der Grundlage der Wahrscheinlichkeitsrechnung hat CLAUDE SHANNON eine Theorie entwickelt, welche die gesuchten Antworten liefert. SHANNON selbst hat seine Theorie *“Kommunikationstheorie”* genannt. Leider wurde sie später in *“Informationstheorie”* umbenannt[3]. Diese Bezeichnung führte - ähnlich wie die Bezeichnung “Kybernetik” - zu manchen Missverständnissen.

Voraussetzung für den Aufbau einer Theorie war eine rigorose Idealisierung und zwar die vollständige Abstraktion vom semantischen Aspekt des Informationsbegriffs, sodass nur der syntaktische Aspekt übrig blieb. Wenn im Rahmen der “Informationstheorie” von “Information” und “Informationsinhalt” die Rede ist, handelt es sich stets nur um den syntaktischen Aspekt der Codierung, um “syntakti-

3 Gestraffte Darstellungen der Informationstheorie findet der Leser z.B. in [Kreß 77] oder [Werner 95].

sche Information" oder "syntaktischen Informationsinhalt". Der Informationsbegriff der Informationstheorie entspricht also nicht dem Informationsbegriff dieses Buches.

Auf ein Resultat der Informationstheorie, das hinsichtlich der Codierungseffizienz von besonderer Bedeutung ist, soll näher eingegangen werden, auf den shannonschen Codierungssatz. Der Satz gibt die untere Grenze für die Länge einer Nachricht bei optimaler binärer Codierung an. Für den bereits erwähnten Fall, dass den Alphabetzeichen einer Nachrichtenquelle (den Zeichen vor der Umcodierung) binäre Codewörter zugeordnet werden, und für sehr lange Mitteilungen lautet der Satz: *Die* ***minimale mittlere Codewortlänge*** $l_{\min}$ *pro Alphabetzeichen der Quelle, die sich durch optimale binäre Codierung erreichen lässt, ist gleich der* ***Entropie*** *H der Quelle,* formal notiert:

$$l_{\min} = H = \Sigma p_i \mathrm{ld}(1/p_i) \tag{5.5}$$

dabei bezeichnet p_i die Auftrittswahrscheinlichkeit des i-ten Quellalphabetzeichens. Die Summe ist über alle i zu nehmen. Mit ld ist der *Duallogarithmus*, also der Logarithmus zur Basis 2 bezeichnet. Man beachte, dass (5.5) eine Mittelwertbildung der $\mathrm{ld}(1/p_i)$-Werte darstellt. Wir werden Formel (5.5) nicht beweisen, jedoch versuchen, sie plausibel zu machen. Der Anschaulichkeit halber gehen wir von folgender Aufgabenstellung aus, die zwar etwas konstruiert, dafür aber leicht zu verstehen und zu durchschauen ist.

Aus einem Quellalphabet mit B Buchstaben werden sämtliche möglichen Ketten, sog. Quellwörter, der Länge L_Q gebildet. B sei eine ganzzahlige Potenz von 2, z.B. B=32. Die Quellwörter sollen nun in möglichst kurze binäre Zielwörter einheitlicher Länge umcodiert werden. Die notwendige Zielwortlänge L_Z lässt sich aus der Bedingung berechnen, dass die Anzahl der Zielwörter 2^{L_Z} gleich der Anzahl der Quellwörter B^{L_Q} sein muss. Durch Logarithmieren erhält man für die Zielwortlänge $L_Z = L_Q \mathrm{ld} B$. Für die mittlere Codewortlänge pro Quellalphabetzeichen $l = L_Z/L_Q$ ergibt sich damit

$$l = \mathrm{ld} B. \tag{5.6}$$

Bevor wir andeuten, wie man von (5.6) zu (5.5) gelangt, wollen wir der Größe $\mathrm{ld} B$, die auch *Entscheidungsgehalt* genannt wird, eine anschaulichere Bedeutung geben. Es gibt ein bekanntes Ratespiel zu Zweien, bei dem der Eine, der *Wissende,* sich ein Objekt ausdenkt, das der Andere, der *Fragende*, durch Fragen erraten muss, auf die der Wissende nur mit "ja" oder "nein" antworten darf. Wenn das zu erratende Objekt ein Element aus einer beiden Spielern bekannten Menge ist, kann der Ratende folgende optimale Ratestrategie anwenden. Er teilt die Menge in zwei gleiche Teilmengen und fragt, ob das Objekt Element der einen (genau zu beschreibenden) Teilmenge ist. Aus der Antwort (Ja oder Nein) erfährt er, in welcher Teilmenge sich

das Objekt befindet. Diese teilt er nun ihrerseits in zwei gleiche Teilmengen und fährt so fort, bis er das zu erratende Objekt identifiziert hat.

Wenn die Gesamtmenge $m = 2^n$ Elemente besitzt, muss er mindestens ld$m = n$ mal fragen, und der Befragte muss ebenso viele *Entscheidungen* treffen. In diesem Sinne sagt man, dass der Entscheidungsgehalt der Identifikation eines Elementes aus einer Menge von m Elementen den Wert ld$m$ bit besitzt. Die Folge der Antworten kann als Binärwort aufgefasst werden, welches das Objekt identifiziert ("codiert"). Es besitzt die Länge n Bit (man beachte die unterschiedlichen Maßeinheiten bit bzw. Bit, die gleich erklärt werden). Um nach dieser Strategie einen bestimmten Buchstaben des Quellalphabets zu erraten, muss man, falls B eine Potenz von 2 ist, ld$B$ mal raten, d.h. von dieser Länge ist das "codierende Binärwort" der Antworten. Dieser Wert stimmt mit dem von l in (5.6) überein. Wenn B keine Potenz von 2 ist, wenn also ld$B$ keine ganze Zahl ist, ergibt sich als Frageanzahl die nächsthöhere ganze Zahl. Dennoch wird als *Entscheidungsgehalt* der Wert von ld$B$ bezeichnet mit der "Maßeinheit" bit. Demgegenüber ist ein Wert mit der "Maßeinheit" Bit stets ganzzahlig (ein Bit ist ein Binärzeichen).

Betrachtet man nun noch einmal die Formel (5.6), erkennt man, dass die Entropie H als mittlerer *individueller* Entscheidungsgehalt aufgefasst werden kann, wobei der individuelle Entscheidungsgehalt eines Zeichens in Übereinstimmung mit (5.6) den Wert ld$(1/p_i)$ besitzt.

Offenbar ist die Entropie der mittlere Entscheidungsgehalt, d.h. die mittlere minimale binäre Codewortlänge pro Zeichen einer (sehr langen) Nachricht, in der die Alphabetzeichen mit unterschiedlicher Häufigkeit auftreten. Das genau beinhaltet der shannonsche Codierungssatz (5.5), dessen Gültigkeit durch unsere Überlegungen zwar plausibel gemacht, aber nicht exakt bewiesen worden ist. Doch lässt er sich beweisen, sodass es gerechtfertigt ist, die Entropie als verallgemeinerten Entscheidungsgehalt zu bezeichnen. Häufig wird die Entropie *syntaktischer Informationsgehalt* oder kurz **syntaktische Information** genannt.

Um den Codierungssatz (5.5) exakt zu beweisen, muss in Analogie zur Herleitung der Formel (5.6) die Anzahl aller Zeichenketten der Länge L berechnet werden, jetzt aber unter der Nebenbedingung, dass die einzelnen Buchstaben mit den relativen Häufigkeiten p_i auftreten. Der erhaltene Wert (bei Gleichverteilung ist er gleich B^L) ist zu logarithmieren und durch L zu teilen. Lässt man schließlich L gegen Unendlich gehen (dabei werden die Häufigkeiten zu Wahrscheinlichkeiten), erhält man den angegebenen Wert für die Entropie. Je stärker sich die p_i-Werte voneinander unterscheiden, umso kleiner ist die Entropie und umso größer ist der Entscheidungsgehalt und die mittlere Codewortlänge und umso mehr lässt sich die Codierungseffizienz durch optimale Umcodierung steigern. Bei *Gleichverteilung* (alle p_i sind einander gleich) nimmt *die Entropie den Maximalwert* $H_{\max}$ =ld$B$ an.

Die Differenz $H_{\max} - H$ gibt an, wieviele Binärzeichen pro Quellzeichen im Mittel eingespart werden können, wenn die Nachrichten einer Quelle mit der Entropie H optimal umcodiert werden. Die Differenz geteilt durch $H_{\max}$ wird *Redundanz* genannt

(vom englischen Wort für “Überfluss”,"Überzähligkeit"). Sie ist ein Maß für die erreichbare prozentuale Umodierungseffizienz. Die Redundanz der deutschen Sprache beträgt etwa 73%. Zwar vermindert Redundanz die semantische Dichte, doch hat sie, wie bereits erwähnt, auch Vorteile. Je größer sie ist, umso eher lässt sich erraten oder rekonstruieren, was mit einer fehlerhaften (z.B. bruchstückhaften) Nachricht gemeint ist und umso mehr Fehler lassen sich korrigieren.

Man beachte, dass Formel (5.5) *kein* Rezept dafür liefert, *wie* optimale Codierung zu erreichen ist. Es sind viele praktische Codierungsmethoden entwickelt worden, auf die wir jedoch nicht eingehen. Einige von ihnen findet man in [Kreß 77].

Die Bezeichnung *Entropie* ist aus der Physik und zwar aus der Thermodynamik übernommen, denn die thermodynamische Entropie berechnet sich (bis auf einen konstanten Faktor) nach der Formel (5.5). Aus diesem Grunde werden wir im Weiteren zwischen *thermodynamischer* Entropie und *informatischer* Entropie oder *Informationsentropie* unterscheiden.

Die Bedeutung der thermodynamischen Entropie liegt darin, dass ihr Wert für abgeschlossene Systeme (Systeme ohne Energieaustausch mit der Umgebung) ständig zunimmt, bis sie ihren Maximalwert erreicht hat.

Die Tatsache, dass sich die thermodynamische und die informatische Entropie nach der gleichen Formel berechnen, hat einen formalen oder besser syntaktischen, jedoch keinen inhaltlichen, keinen kausalen Grund. In beiden Fällen hat man es mit ein und demselben *kombinatorischen* Problem zu tun. Das soll an einem einfachen Beispiel für die Berechnung der thermodynamischen Entropie veranschaulicht werden. Die folgenden Überlegungen gehen auf LUDWIG BOLTZMANN zurück, dem es gelang, die Aussagen der Thermodynamik aus den Vorstellungen der kinetischen Gastheorie abzuleiten, d.h. sie mikroskopisch zu interpretieren.

Man stelle sich ein geschlossenes 1-Liter-Gefäß vor, in dem sich N gleiche 22
Gasmoleküle befinden. Gedanklich unterteilen wir das Gesamtvolumen in $B=10^6$ Zellen von je 1mm^3 Inhalt und machen eine mikroskopische Momentaufnahme, auf der zu erkennen ist, in welcher Zelle sich jedes einzelne Molekül befindet; wir nehmen also an, dass sich die Moleküle voneinander unterscheiden lassen. Die Aufnahme zeigt einen sogenannten *Mikrozustand* des Gasvolumens. Wir zählen nun die Moleküle in jeder Zelle, in der i-ten Zelle seien es n_i. Die Gesamtheit der n_i beschreibt die *Dichteverteilung* des Gases.

Eine bestimmte Dichteverteilung kann offensichtlich durch unterschiedliche Mikrozustände realisiert werden. Vertauschen nämlich zwei Moleküle ihre Plätze, so ändert sich die Dichteverteilung nicht, während der Mikrozustand in einen anderen übergeht, vorausgesetzt, die beiden Moleküle befinden sich in unterschiedlichen Zellen. Befinden sie sich dagegen in ein und derselben Zelle, so ändert sich bei ihrer Permutierung weder die Dichteverteilung noch der Mikrozustand. Das ist zu beachten, wenn man die Anzahl aller möglichen Vertauschungen (Permutationen) zählt, also aller Mikrozustände, die eine gegebene Dichteverteilung realisieren. Nach den Regeln der Kombinatorik ist die Anzahl gleich $N!/(n_1!n_2!...n_B!)$. Dabei ist $N!$ die

Anzahl aller möglichen Permutationen[4], $n_i!$ die Anzahl der (nicht zu berücksichtigenden) Permutationen in der *i*-ten Zelle und *B* ist die Anzahl der Zellen. Im Nenner steht das Produkt aller $n_i!$.

Auf das gleiche kombinatorische Problem stößt man, wenn man fragt, wie viele unterschiedliche Buchstabenketten (Wörter) sich aus *N* Buchstabenexemplaren bilden lassen, also wie viele verschiedene Wörter der Länge *N*. Man versetze sich in die Lage eines Schriftsetzers, dem von jedem Alphabetbuchstaben, im Weiteren *Buchstabentyp* genannt, mehrere Exemplare zur Verfügung stehen, und zwar vom *i*-ten Buchstabentyp n_i Exemplare. Er verkettet nun alle Buchstabenexemplare zu einem Wort der Länge *N*. Sodann bildet er neue Wörter durch Permutation, d.h. durch Vertauschen einzelner Buchstabenexemplare. Zählt man nun alle unterschiedlichen Wörter, die sich auf diese Weise durch fortgesetztes Permutieren bilden lassen, hat man - ebenso wie beim Vertauschen der Gasmoleküle - zu beachten, dass sich nur dann ein neues Wort ergibt, wenn *unterschiedliche* Buchstaben miteinander vertauscht werden. Werden gleiche Buchstaben vertauscht, bleibt das Wort erhalten. Demzufolge berechnet sich die Anzahl der unterschiedlichen Wörter bei vorgegebenen Buchstabenhäufigkeiten nach derselben Formel, nach der sich auch die Anzahl der Mikrozustände bei gegebener Dichteverteilung berechnet. (Hier liegt der Grund für die syntaktische Übereinstimmung der Formeln für die thermodynamische und die informatische Entropie.) Die Anzahl ist gleich $N!/(n_1!n_2!...n_B!)$. Für sehr große *N* (sehr viele Buchstabenexemplare bzw. Moleküle) und sehr große n_i (sehr viele Exemplare pro Buchstabentyp bzw. Moleküle pro Zelle) lassen sich die Fakultäten nach der Stirlingschen Formel in Exponentialausdrücke überführen. Durch Logarithmieren und Übergang von Häufigkeiten (Besetzungszahlen) zu Wahrscheinlichkeiten (Dividieren durch *N*) ergibt sich die in (5.5) angegebene Summe. In der Physik ist es üblich, nicht den dualen, sondern den natürlichen Logarithmus zu verwenden. Außerdem wird die Summe mit einem dimensionierten konstanten Faktor multipliziert, sodass die Entropie diejenige Dimension erhält (Energie durch Temperatur), mit der sie in der Thermodynamik ursprünglich eingeführt worden ist. Damit ist gezeigt, dass die formelmäßige Übereinstimmung von informatischer und thermodynamischer Entropie eine formale und keine inhaltliche (physikalische) Ursache hat.

Wir sind so ausführlich auf den Begriff der Entropie eingegangen, um der Gefahr vorzubeugen, den Begriff der informatischen Entropie oder des syntaktischen Informationsgehalts mit dem Begriff der thermodynamischen Entropie zu identifizieren und z.B. zu schlussfolgern, dass bei der Übertragung einer Nachricht mit einem bestimmten syntaktischen Informationsgehalt (einer bestimmten Menge informatischer Entropie) die entsprechende Menge thermodynamischer Entropie (unter Berücksichtigung einer dimensionierten Verhältniszahl) vom Sender an den Empfänger

4 Zur Bedeutung des Ausrufungszeichens als Operationssymbol siehe (8.12).

übertragen wird. Das wäre ein Trugschluss. Eine "*wirkliche*", d.h. physikalisch *wirksame* Beziehung zwischen physikalischer und informatischer Entropie kann nur über die Trägersysteme bestehen, die sprachlich miteinander kommunizieren. Denn nur diese, nicht aber die von ihnen produzierten Sprachen als solche, gehorchen den Gesetzen der Thermodynamik.

Ohne Frage hat der Umstand, dass informationelle Systeme den Gesetzen der Thermodynamik gehorchen, Konsequenzen hinsichtlich ihrer Entstehung und Verhaltensweise, beispielsweise hinsichtlich der phylogenetischen und ontogenetischen Entwicklung des menschlichen Gehirns und seines Funktionierens. Wenn man die Probleme des sprachlichen Modellierens konsequent von der Physik her, das bedeutet *sub*symbolisch, angeht und wenn es gelingt, auf diesem Weg einen Informationsbegriff zu definieren und eine Wissenschaft vom aktiven sprachlichen Modellieren zu entwickeln, könnte eine *Naturwissenschaft* "Informatik" entstehen. In ihr wird die *thermodynamische* Entropie zu den Grundbegriffen gehören. Erste Schritte auf diesem Wege sind bereits getan[5].

5 Siehe z.B. [Ebeling 82], [Nicolis 87], [Ebeling 91], [Ebeling 98].

6 Arbitrarität und Zirkularität

Zusammenfassung

Das Artikulieren (Zuordnen von Zeichenrealemen zu Idemen) ist seiner Natur nach *arbiträr*, d.h. es ist beliebig in dem Sinne, dass es durch keine Naturgesetze erzwungen wird. Die Arbitrarität hat zwei Wurzeln, die individuelle Freiheit einer Person, sich auszudrücken, und die kollektive Freiheit einer Sprachgemeinschaft, Zeichenrealeme mit bestimmten Bedeutungen zu vereinbaren.

Unter dem Begriff der Zirkularität werden beliebige Rückwirkungen realer Objekte auf sich selbst (*operationale Zirkularität*) und beliebige Rückbezüglichkeiten sprachlicher Ausdrücke auf sich selbst (*referenzielle Zirkularität*) zusammengefasst. Zirkularitäten können sinnvoll oder sinnlos sein, sie können widerspruchsfrei oder widersprüchlich sein. Infolge referenzieller Zirkularität kann eine Aussage *unentscheidbar* werden, dass heißt, es kann nicht bewiesen werden, ob die Aussage richtig oder falsch ist.

Die formale Untersuchung der Entscheidbarkeit von Aussagen hat KURT GÖDEL 1931 zu seinem *Unvollständigkeitssatz* geführt: In einem ausreichend komplexen Kalkül lassen sich Sätze formulieren, die zwar richtig, in dem betreffenden Kalkül aber nicht entscheidbar sind.

6.1 Arbitrarität des Artikulierens

Die Gedanken sind frei.
Wer kann sie erraten?
Volkslied

Die Gedanken sind frei, nicht nur, weil man denken kann, was man will, sondern auch, weil man sagen kann, was man will, sodass ein anderer aus dem, was man sagt, nicht unbedingt schließen kann auf das, was man denkt. Mit dieser Feststellung setzen wir die Überlegungen des Kapitels 2 zum Thema “Gedanke und Sprache” fort. Denken und Sprechen sind dem freien Willen untergeordnet, zumindest im vollbewussten Zustand. Daraus ergeben sich erfreuliche, aber auch recht problematische Freiheiten. Ob und wie ein Mensch das, was er denkt, in Worten wiedergibt, ist ihm durch kein ihm bekanntes Naturgesetz vorgeschrieben. Er kann einen Gedanken (ein Idem) in verschiedenen Sprachen und in ein und derselben Sprache auf unterschiedliche Weise artikulieren. Er muss sich allerdings, um verstanden zu werden, an die Sprachgewohnheiten und Konventionen seiner Umgebung halten. Für diese Unbestimmtheit verwenden wir das Wort *Arbitrarität*.

*Unter **Arbitrarität** des Artikulierens verstehen wir das Fehlen eines zwangsläufigen, durch Naturgesetze eindeutig diktierten Zusammenhanges zwischen einem Idem*

und dem ihm zugeordneten Zeichenrealem. Die Arbitrarität hat zwei Wurzeln, die Willensfreiheit des einzelnen und die Willkür sprachlicher Gewohnheiten und Konventionen innerhalb sozialer Gruppen.

In Kap.2 hatten wir festgestellt, dass Sprechen ursprünglich dem Austausch interner Modelle der Welt zum Zwecke der Kooperation dient. Aussagesätze sind also Modellaussagen über die Welt. Es müssen wahre Aussagen sein, sonst würden sie die Kooperation stören. Unter diesem Aspekt erscheint die Freiheit des Artikulierens in ziemlich fragwürdigem Licht. Man kann sich etwas ausdenken, was keine Entsprechung in der Realität hat, und kann das auch sagen. Man kann aber auch etwas anderes sagen, als man denkt. Mit anderen Worten, man kann phantasieren und kann schwindeln. Zwischen beidem besteht ein gleitender Übergang. Je nachdem, in welchem Maße man seine Zuhörer über den Wahrheitsgehalt einer Aussage im Unklaren lässt, ist es mehr Phantasie oder mehr Schwindel, was man ihm "aufbindet". In beiden Fällen macht man sich die Freiheit des Artikulierens zunutze. Anders ist es, wenn man irrt, also sich nicht bewusst etwas Falsches ausdenkt. Doch angesichts der Möglichkeit FREUDscher Verdrängung kann auch der Übergang vom Irrtum zur Lüge ein gleitender sein.

Schließlich ist es jedem freigestellt, ganz bewusst blanken Unsinn zu reden. Das kann beim Interpretierer Verständnislosigkeit, Unwillen oder Amüsement hervorrufen, wie z.B. folgende Frage: "Was ist der Unterschied zwischen einem Raben?" und die Antwort: "Er hat zwei gleichlange Beine, besonders das rechte." Die Arbitrarität des Artikulierens lässt so etwas zu. Und es hat durchaus seine soziale Bedeutung, ebenso wie das Phantasieren und das Schwindeln.

Nach dieser Abschweifung präzisieren wir die zentrale Aussage. Artikulieren ist seiner Natur nach arbiträr. Die Arbitrarität hat zwei Ursachen, die individuelle Freiheit, etwas zu sagen wie und wann man will, und die kollektive Freiheit zu vereinbaren, welche Symbole syntaktisch und semantisch wie zu verwenden sind, m.a.W. was wie codiert wird. Im Falle natürlicher Sprachen ist diese Festlegung im wesentlichen ein Ergebnis der kulturellen Evolution, im Falle künstlicher Sprachen beruht sie auf Vereinbarung. Das gilt insbesondere für Programmiersprachen. Das muss nicht so bleiben. Es ist immerhin denkbar, dass die Computertechnik in eine Phase eintritt, in der sie selber aktiv an der kulturellen Evolution teilnimmt, und dass Computersprachen unter Mitwirkung des Computers selber evolutionieren und nicht allein nach Maßgabe des Menschen.

6.2 Referenzielle Zirkularität

Eine andere problematische Freiheit ist die Möglichkeit des rückbezüglichen Denkens, Schließens und Artikulierens. Sie tritt in unterschiedlichen Formen auf und kann zu merkwürdigen inneren Widersprüchlichkeiten und zu paradoxen Aussagen führen, durch die sich seit eh und je scharfsinnige Philosophen herausgefordert

fühlen. Deren Markenzeichen ist daher die Schlange, die sich selbst von ihrem eigenen Schwanz her verschlingt. Dabei wird ein in sich "verschlungener" Denkprozess durch einen in sich "verschlungenen" realen Prozess versinnbildlicht. Das Gemeinsame ist der zirkuläre Charakter. Im Weiteren wird das Wort Zirkularität (und entsprechend die Wörter Zirkel und zirkulär) in folgendem Sinne verwendet.

*Alle Arten von Rückbezüglichkeit und Rückwirkung auf sich selbst (Reflexivität, Rekursivität, Rückkopplung) werden unter dem Begriff "***Zirkularität***" zusammengefasst.* Auf einige Arten der Zirkularität soll näher eingegangen werden. Wir beginnen mit der Selbstbenennung. Zunächst definieren wir das Wort *Referenzieren.*

Unter **Referenzieren** *versteht man das Verweisen auf ein Objekt der Realität oder des Denkens (auf ein Realem oder Idem), durch Nennung eines Stellvertreters, eines Namens oder Pronomens, oder durch Angabe seines Ortes (z.B. postalische Adresse, bibliographische Angaben, Speicheradresse).* In der Computer-IV ist Referenzieren das Verweisen auf ein Zeichenrealem mittels eines anderen Zeichenrealems.

Wir wollen die Konsequenzen der Freiheit des Referenzierens genauer untersuchen. Dieser Freiheit sind an sich keinerlei natürliche oder logische Grenzen gesetzt. Sie erlaubt auch falsches, sinnloses und sogar widersinniges Referenzieren. Sie erlaubt selbstverständlich auch das Selbstreferenzieren. Wir sprechen dann von **referenzieller Zirkularität**. Diese liegt beispielsweise vor, wenn ein Artikulierer von sich selbst spricht, wenn ein Satz etwas über sich selbst aussagt oder wenn ein Wort sich selbst "meint", wie der zweite der folgenden Sätze demonstriert.

Hans ist ein Junge von 4 Jahren.

Hans ist eine Zeichenkette von 4 Buchstaben.

Im ersten Satz verweist das Zeichenrealem "Hans" auf eine Person, im zweiten auf sich selbst. Betrachten wir nun die folgenden Sätze.

Dieser Satz ist ein Aussagesatz.

Dieser Satz ist ein Fragesatz.

Das Demonstrativpronomen "dieser" zeigt die Rückbezüglichkeit an. Beide Sätze sind erlaubt. Aber der zweite Satz ist falsch, während der erste wahr ist. Diese Eindeutigkeit ist nicht immer gegeben, wie der zweite der folgenden Sätze zeigt.

Dieser Satz ist wahr.

Dieser Satz ist falsch. 1

Wenn der erste Satz wahr ist, so stimmt das mit dem, was er über sich selbst aussagt, überein. Der Satz ist wahr, allerdings inhaltslos. Wenn der zweite Satz wahr ist, dann ist er nach dem, was er über sich selber aussagt, falsch, d.h. der Satz ist falsch. Wenn der Satz falsch ist, dann ist das, was er über sich selbst aussagt, richtig; der Satz ist also richtig. Es liegt ein Widerspruch, eine *Antinomie* vor und es lässt sich nicht entscheiden, ob der Satz wahr oder falsch ist, man sagt: der Satz ist nicht

entscheidbar. Wie man sieht, *kann referenzielle Zirkularität zu Antinomien und zu nichtentscheidbaren Aussagen führen.*

Ein referenzieller Zirkel kann sich auch über mehrere Objekte erstrecken, wie die beiden folgenden Sätze zeigen, die wir mit A und B benennen.

2 Satz A: “Satz B ist falsch.”

Satz B: “Satz A ist falsch.”

Wie man sich leicht überzeugt, kommt es nur dann zu *keinem* Widerspruch, wenn einer der beiden Sätze wahr, der andere falsch ist. Einer ähnlichen Situation werden wir in Kap.9.4 [9.20] im Zusammenhang mit dem Flipflop begegnen.

Schon im Altertum haben sich die Menschen mit Antinomien herumgeschlagen. Die berühmteste ist die *Lügnerantinomie* in der Form “Alle Kreter lügen.” Dieser Satz ist an sich nicht widersprüchlich; er wird es aber im Munde eines Kreters. Doch auch dann stellt er genau genommen keine Antinomie dar, denn er kann entschieden werden. Er ist nämlich falsch, sowohl, wenn der Kreter lügt, als auch, wenn er nicht lügt (vgl. [Vollmer 88] Bd.1).Dagegen ist der Satz “Jetzt lüge ich” eine Antinomie und nicht entscheidbar.

Zuweilen werden Sätze, die genau genommen Antinomien sind, nicht als solche empfunden, sondern “automatisch” widerspruchsfrei interpretiert. Ein Beispiel ist Platons berühmter Ausspruch “Ich weiß, dass ich nichts weiß”. Der Widerspruch kann durch eine kleine Erweiterung aufgehoben werden: “Ich weiß, dass ich sonst nichts weiß”, oder ausführlicher: “Ich weiß, dass ich nichts weiß ausser dem einen, dass ich nichts weiß”.

Der Verdacht liegt nahe, dass all diese Komplikationen die Folge der Ungenauigkeit des Sprechens oder auch des Denkens sind, dass sich Antinomien und die Unentscheidbarkeit von Sätzen durch semantische Objektivierung vermeiden lassen und dass nach Übergang zu formaler Semantik Antinomien ausgeschlossen werden können. Dieser Verdacht hat sich als falsch erwiesen. Die Unentscheidbarkeit von Sätzen ist unvermeidlich. Das ergibt sich aus dem *ersten Unvollständigkeitssatz*[1], den KURT GÖDEL bewiesen hat [Gödel 31]. Nach diesem Satz kann es in einem widerspruchsfreien formalen System wahre Sätze geben, die in dem System nicht entscheidbar sind, die sich also mit den Mitteln des formalen Systems weder beweisen noch widerlegen lassen. Wir kommen darauf in Kap.8.5 zurück.

6.3 Operationale Zirkularität

In der “Schlangenmetapher” wird die Vorliebe spitzfindiger Denker für rück*bezügliche* Zirkularität des Denkens durch rück*wirkende* Zirkularität einer Tätigkeit

1 Siehe z.B. [Schreiber 84], [Schöning 95]

versinnbildlicht. Dieser zweiten Art von Zirkularität wenden wir uns nun zu. Wir nennen sie **operationale Zirkularität**. Dabei ist zwischen *Prozesszirkularität* und *Operand-Operator-Zirkularität* zu unterscheiden. In der Schlangenmatapher handelt es sich um eine Rückwirkung, bei der das Subjekt der Tätigkeit (des Verschlingens) gleichzeitig deren Objekt ist. In mehr technischer Redeweise lässt sich die Situation folgendermaßen charakterisieren. *Der Operator, der die Operation ausführt, ist mit dem Operanden, an dem sie ausgeführt wird, identisch.* In diesem Sinne sprechen wir von **Operand-Operator-Zirkularität**.

Die Widersinnigkeit der Schlangenmetapher könnte vermuten lassen, dass Operand-Operator-Zirkularität immer etwas Widersinniges ist. Ganz einfache Beispiele zeigen aber, dass dies nicht der Fall ist. Der genannte Typ von Zirkularität liegt z.B. vor, wenn ein "Auto-mobil" fährt, oder wenn ich mich wasche, wobei das Reflexivpronomen "mich" die Zirkularität anzeigt. Der Zirkel kann sich auch über mehrere Operatoren schließen, beispielsweise wenn eine Hand die andere wäscht. Wenn dagegen eine Hand die andere zeichnet, wie in Bild 6.1 dargestellt, wird der Zirkel widersinnig, d.h. er ist zwar darstellbar, aber in Wirklichkeit unmöglich. Darin zeigt sich die Arbitrarität des "Artikulierens mittels *Zeichnen*", ähnlich der des Artikulierens mittels *Zeichen*. Bild 6.1 stammt von MAURITS CORNELIS ESCHER[2].

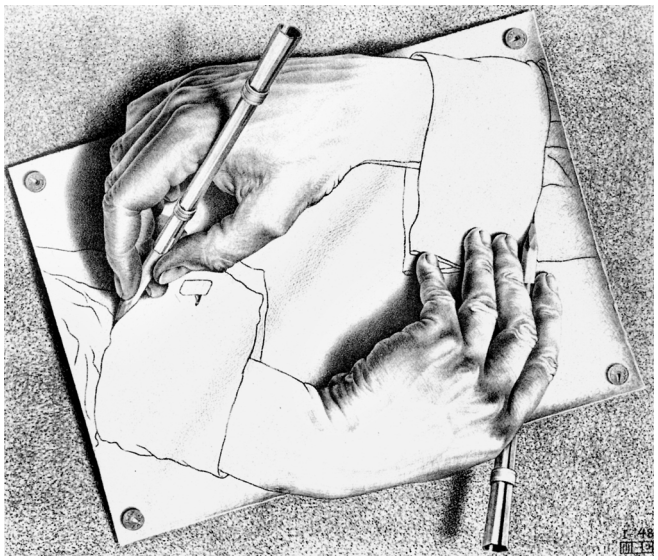

Bild 6.1 Widersinniger Operand-Operator-Zirkel

In einem Operand-Operator-Zirkel relativieren sich die Begriffe Operand und Operator. Diese Relativierung ist nicht an die Existenz eines Zirkels gebunden, wie am Beispiel einer Werkzeugmaschine deutlich wird. Aus der Sicht der Maschine ist das produzierte Werkzeug Operand, aus der Sicht der Operation, die mit dem Werkzeug ausgeführt wird, ist es Operator. Ein Zirkel läge vor, wenn die Maschine sich selbst produzieren würde. Es gibt zwar Selbstvernichtung, jedoch keine Selbstproduktion, sondern nur *Selbstreproduktion*, bei der ein Operator einen sich selbst gleichen aber nicht identischen Operator "der *nächsten Generation*" produziert. Hier kann man von Operand-Operator-Zirkularität im Sinn von Wiederholung des gleichen Prozesses sprechen, wobei der Operand eines Zirkeldurchlaufs zum Operator

2 Aus [Ernst 85]

des nächsten Durchlaufs (der nächsten Generation) wird. Selbstreproduktion ist mit Informationsübergabe von Generation zu Generation verbunden, wie wir in Kap.1 [1.2] in Verbindung mit der Evolution festgestellt hatten. Doch gilt das auch für Selbstreproduktion künstlicher Systeme, wie JOHN VON NEUMANN gezeigt hat [Neumann 66].

Eine Zirkularität liegt auch dann vor, wenn ein Operator sich selber aufruft. Dann spricht man von *Zyklus*. Wenn dabei ein und derselbe Operand bearbeitet wird, liegt Mehrfachverarbeitung vor, die im Falle eines informationellen Operators *Iteration* genannt wird. Dabei handelt es sich nicht um Operand-Operator-Zirkularitäten. Die Rückwirkung eines Operators auf sich selbst und speziell den Selbstaufruf während einer Operationsausführung (eines Prozesses), nennen wir **Prozesszirkularität**. Der Begriff der Prozesszirkularität kann in unterschiedlichen spezielleren Bedeutungen verwendet werden. Ein Regelungstechniker wird zuerst vielleicht an einen Regelkreis denken, ein Techniker allgemein an eine Rückkopplungsschleife, über welche Signale oder Informationen vom Ausgang eines Gerätes auf seinen Eingang zurückgegeben werden. Rückkopplungsschleifen sind in informationellen Systemen auf Schritt und Tritt anzutreffen, sowohl in der Hardware als auch in der Software, wo sie allerdings nicht *"verlötet"*, sondern *beschrieben (sprachlich modelliert)* werden.

Wir kehren noch einmal zur Operand-Operator-Zirkularität und speziell zur Relativität der Begriffe Operator und Operand zurück. Diese Relativität ist ein charakteristisches Merkmal der Computer-IV. In Anlehnung an den mathematischen Sprachgebrauch kann ein Programm als Operator aufgefasst werden, der die Eingabedaten (Operanden) verarbeitet, auf die es "angewandt" wird. Andrerseits kann ein Programm von einem anderen Programm bearbeitet, z.B. editiert, korrigiert oder übersetzt werden, also die Rolle eines Operanden spielen. Ein Programm, das übersetzt und anschließend ausgeführt wird, ist zuerst Operand und dann Operator. Wenn ein Übersetzerprogramm sich selbst übersetzt, liegt Operand-Operator-Zirkularität vor. Das ist durchaus möglich und weit weniger überraschend als die sich selbst bearbeitende Maschine. Denn Zeichenrealeme, also auch Programme lassen sich kopieren, und bei der Selbstübersetzung übersetzt ein Exemplar eines Programms ein anderes Exemplar desselben Programms.

4 Wenn man in einem Operand-Operator-Zirkel den Operator als Ursache und den Operanden bzw. dessen Veränderung als Wirkung ansieht, gelangt man zu dem *Ursache-Wirkung-Zirkel*, von dem in Kap.4.2 [4.6] im Zusammenhang mit den newtonschen Bewegungsgleichungen die Rede war.

5 Schließlich sei angemerkt, dass die Informatik selber eine zirkuläre Wissenschaft ist. Ihr *Gegenstand* ist das sprachliche Modellieren und wie jede Wissenschaft bedient sie sich des sprachlichen Modellierens als *Mittel*. Das sprachliche Modellieren bildet einen Operand-Operator-Zirkel.

7 Evolution der Intelligenz

Zusammenfassung

Die natürliche Intelligenz hat sich von der Fähigkeit des Erkennens bis hin zur Fähigkeit des Rechnens entwickelt. Die künstliche Intelligenz hat sich in der entgegengesetzten Richtung entwickelt. Diese scheinbar widersprüchliche Beobachtung nennen wir das *Gegenläufigkeitsphänomen* der Evolution der Intelligenz. Das Wort *Erkennen* kann *Wiedererkennen* oder *Erkenntnisgewinnung* bedeuten.

Intelligenz ist die Fähigkeit zum sprachlichen Modellieren. Sprachliches Modellieren der Welt besteht im Finden richtiger Aussagen über die Welt. Aussagen ordnen Objekten (genauer: Denkobjekten) Merkmale zu. Richtige Aussagen können durch Deduktion, Assoziation oder Intuition gefunden werden. *Deduzieren* (Synonym zu *Ableiten*) kann in Rechnen oder Schlussfolgern bestehen. *Rechnen* ist das Ableiten von Aussagen mittels Rechenregeln im Rahmen eines Kalküls. *Schlussfolgern* durch den Menschen ist das Ableiten von Aussagen mittels nicht formalisierter Schlussregeln. Deduktion ist eine Intelligenzleistung des Bewusstseins, Assoziation und Intuition sind Intelligenzleistungen, an denen das Unterbewusstsein wesentlich beteiligt ist. *Assoziation* beinhaltet das *Wiederauffinden* bekannter Aussagen, d.h. bekannter Zuordnungen zwischen Objekten und Merkmalen, ohne bewusstes Suchen. *Intuition* beinhaltet das *Erfinden* neuer Aussagen, also neuer Zuordnungen zwischen Objekten und Merkmalen. Ihr liegen unbewusste Erfahrungen des Individuums zugrunde. Intuition, Kreativität und Phantasie sind im Kern miteinander wesensgleich. Intuition ist eine produktive, Assoziation eine reproduktive Intelligenzleistung.

Der Mensch verfügt über *Metaintelligenz*, d.h. über die Fähigkeit, das eigene sprachliche Modellieren zu modellieren und zielgerichtet weiterzuentwickeln. Im Vergleich zum Menschen besitzt der heutige Computer kaum Metaintelligenz.

Ein fundamentaler Begriff der modernen Rechentechnik ist der uralte Begriff des Algorithmus. Ein *Algorithmus* ist eine Handlungsvorschrift (Prozessbeschreibung), die angibt, in welcher Reihenfolge mehrere, als bekannt vorausgesetzte Einzelhandlungen oder Operationsschritte auszuführen sind, um ein bestimmtes Ziel zu erreichen. Die Handlungsfolge muss eindeutig und endlich sein (aus endlich vielen Schritten bestehen). Ein Operationsschritt heißt *Aktion*, wenn die Operanden der Operation explizit angegebenen sind. Eine Aktionsvorschrift heißt *Anweisung* oder *Befehl*. Ein Algorithmus, der aus Anweisungen (Befehlen) besteht, heißt *imperativer Algorithmus*.

Die Universalität des Computers beruht auf der Tatsache, dass sich jede arithmetische, analytische und logische Operation aus Inkrementier- und Vergleichsoperationen komponieren lässt und diese sich ihrerseits aus denjenigen arithmetischen und

logischen Operationen komponieren lässt, welche die ALU (Arithmetisch-Logische Unit) ausführen kann.

7.1 Deduktive, assoziative und intuitive Intelligenz.

Intelligenz hatten wir als Fähigkeit zum sprachlichen (codierenden) Modellieren definiert [3.1]. Wir werden uns nun für die Evolution dieser Fähigkeit interessieren. Geht man der Frage nach, in welcher Richtung Intelligenz evolutioniert, so kommt man zu entgegengesetzten Antworten, je nachdem, ob man die natürliche oder die künstliche Intelligenz betrachtet. Die Entwicklungsrichtungen sind einander diame-
1 tral entgegengerichtet. Etwas verkürzt kann man sagen: *Die natürliche Intelligenz entwickelt sich* ***vom Erkennen zum Rechnen****, die künstliche Intelligenz* ***vom Rechnen zum Erkennen***. Diese Beobachtung nennen wir das ***Gegenläufigkeitsphänomen*** *der Evolution der Intelligenz*. Die Evolution der natürlichen Intelligenz ist phylogenetisch eine Komponente der genetischen und ontogenetisch eine Komponente der individuellen Evolution. Die Evolution der künstlichen Intelligenz ist eine Komponente der kulturellen Evolution.

Der gegenwärtige Stand der Technik lässt folgende grobe Charakterisierung der Fähigkeiten des Computers im Vergleich zu denen des Menschen zu. Der Computer kann besser *rechnen*, etwa ebenso gut *schlussfolgern*, aber schlechter *assoziieren* und *erkennen* als der Mensch. Die hervorgehobenen Wörter haben folgende Bedeutungen.

Das Wort *Erkennen* hat zwei Bedeutungen. Zum einen kann es das Erkennen eines Objekts (“Das ist Fritz”) oder das Erkennen einer Klassenzugehörigkeit (“Das ist eine Linde”) bedeuten. Dann sprechen wir von **Wiedererkennen** oder **Identifizieren**. Zum anderen kann “Erkennen” das Gewinnen neuer Erkenntnisse bedeuten, das “Begreifen” eines unbekannten Objekts, Phänomens oder Sachverhalts. Dann sprechen wir von Erkenntnisgewinnung. **Erkenntnisgewinnung** ist das Finden neuer Aussagen über die Welt. Die Fähigkeit des Menschen, Erkenntnisse zu gewinnen, zu sammeln und weiterzugeben ist zusammen mit der Fähigkeit zu abstrahieren und Begriffe zu bilden eine Grundvoraussetzung der kulturellen Evolution[1]. Damit eine Entwicklung stattfinden kann, müssen Erkenntnisse *sinnvolle* Aussagen sein, d.h. sie müssen in das bereits vorhandene Wissen (Modell der Welt) hineinpassen. Neue sinnvolle Aussagen (Erkenntnisse) verändern das vorhandene Modell so, dass ein umfassenderes, ein genaueres oder ein in sich konsistenteres Modell entsteht.

Rechnen *ist das Ableiten von Aussagen mittels Rechenregeln im Rahmen eines Kalküls*. Es setzt eine *formale Semantik* voraus. ***Schlussfolgern*** durch den Menschen ist das Ableiten von Schlüssen aus gegebenen Fakten, wobei aus introspektiver Sicht

1 Zu diesem Thema siehe z.B. [Klix 80].

des denkenden Menschen das Ableiten i.d.R. nicht explizit formalisiert ist. Verkürzt sagen wir: ***Schlussfolgern*** *ist das Ableiten von Aussagen mittels nichtformalisierter* ***Schlussregeln***. Unter Schlussregeln sind Regeln zu verstehen, nach denen Aussagen durch logisches Denken unter Verwendung einer beliebigen Sprache ohne Inanspruchnahme eines Kalküls in andere Aussagen umgeformt werden können. Beispielsweise kann man aus den Aussagen "A ist Kind von B" und "B ist Schwiegermutter von C" *schlussfolgern*, dass A und C miteinander verheiratet oder verschwägert sind (vgl. die Denksportaufgabe 1 in Kap.16.1 [16.2]). Rechnen und Schlussfolgern fassen wir unter der Bezeichnung **Deduzieren** zusammen. In Kap.16 werden wir sehen, wie sich das Schlussfolgern mathematisieren lässt, sodass die Grenze zwischen Schlussfolgern und Rechnen verschwimmt.

Deduzieren ist ein weitgehend bewusster Prozess, wir sagen: Deduktion ist eine Leistung *bewusster* Intelligenz. Der deduzierende Mensch weiß, wie er zu der deduzierten Aussage gekommen ist. Das gilt sowohl für das Rechnen als auch für das Schlussfolgern. Auch das Identifizieren kann ein bewusster Prozess sein, z.B. wenn man eine Pflanze durch Suchen im Gedächtnis oder in einem Pflanzenerkennungsbuch erkennt. Das Erkennen kann aber auch *spontan*, ohne Suchen und ohne 2
Nachdenken erfolgen, also weitgehend im Unterbewusstsein stattfinden, z.B. wenn man einen Bekannten erkennt. In diesem Fall sprechen wir von *Assoziation*.

Für die Erkenntnisgewinnung gilt Ähnliches. Erkenntnisgewinnung durch Deduzieren ist eine Leistung *bewusster* Intelligenz. Eine neue Erkenntnis kann einem aber auch "einfallen" (zufliegen, überkommen), ohne dass man weiß, wie der Einfall genau zustande gekommen ist. Der entscheidende Punkt eines solchen Erkenntnisprozesses liegt im Unbewussten. Es handelt sich um eine Leistung *unbewusster* Intelligenz. In diesem Fall sprechen wir von *Intuition*.

Da uns Assoziation und Intuition später noch beschäftigen werden, präzisieren wir ihre Definitionen (eingedenk der Tatsache, dass Aussagen Zuordnungen zwischen Objekten und Merkmalen sind): ***Assoziation*** *ist die Reproduktion (das* ***Auffinden*** *im Gedächtnis) bekannter Zuordnungen zwischen Objekten und Merkmalen ohne bewusstes Suchen.* ***Intuition*** *ist die Produktion (das* ***Erfinden****) neuer Zuordnungen zwischen Objekten und Merkmalen ohne bewusstes Deduzieren.* Worauf 3
Assoziation, Intuition und Erfinden beruht, welche Gehirnprozesse ihnen zugrunde liegen, ist weitgehend unbekannt. Die Introspektion sagt darüber wenig aus.

Wir fassen zusammen und ergänzen. Wir unterscheiden drei Wege, die zu Aussagen über die Welt führen, drei Methoden des sprachlichen Modellierens: Deduktion, Assoziation und Intuition. Die Fähigkeit, den einen oder anderen Weg des sprachlichen Modellierens zu gehen, nennen wir **deduktive** bzw. **assoziative** bzw. **intuitive Intelligenz**. Beim Wiedererkennen kommt in erster Linie assoziative Intelligenz zum Tragen, bei Erkenntnisgewinnung deduktive und intuitive Intelligenz.

Gegen die eingeführten Begriffsbestimmungen können verschiedene Einwände erhoben werden. Zunächst kann das Wort *Erfinden* irritieren. Denn üblicherweise

verbindet man damit das Erfinden eines Mechanismus, einer Maschine bzw. des Wirkprinzips einer Maschine. Danach passt das Wort Erfinden eher auf das *Konstruieren* als auf das sprachliche *Modellieren*. Doch ist zu bedenken, dass jedem Konstruieren sprachliches Modellieren vorausgeht. Das rechtfertigt unseren Gebrauch
4 des Wortes **Erfinden** im Sinne von intuitivem Finden neuer Zusammenhänge zwischen Objekten und Merkmalen. Ein ähnlicher Einwand kann gegen die Definition des Begriffs *Intuition* erhoben werden. Es wird weit öfter von intuitivem Handeln als von intuitivem Denken gesprochen. Wieder ist zu beachten, dass jedem Handeln ein Modellieren vorangeht. Ferner kann gefragt werden, worin nach der gegebenen Begriffsbestimmung die Unterschiede zwischen Intuition, Kreativität und Phantasie liegen. Die Antwort lautet: Sie liegen im Kontext, in welchem das eine oder andere Wort vorzugsweise benutzt wird. Diese und ähnliche Einwände zeigen, dass es vielleicht besser gewesen wäre, neue Wörter zu erfinden. Wir haben vorgezogen, die Wörter “Intelligenz”, “Assoziation” und “Intuition” beizubehalten, sie aber in einer nicht ganz üblichen Bedeutung zu verwenden.

Zur Vertiefung des Verständnisses der eingeführten Begriffe seien einige Probleme angeführt, deren Lösung die eine oder andere Intelligenzart erfordert. Viele Rätsel verlangen intuitive Intelligenz, z.B. das Rätsel der Sphinx aus dem Ödipusmythos: “Früh geht’s auf vier Beinen, mittags auf zwei, abends auf drei” (Lösung: der Mensch). Das Multiplizieren zweier mehrstelliger Zahlen verlangt deduktive Intelligenz. Die Erfindung der Differenzialrechnung verlangte gleichzeitig intuitive *und* deduktive Intelligenz. Das soll näher erläutert werden.

Newton hat neue Sprachelemente (Differenzial, Differenzialquotienten verschiedener Ordnung) eingeführt (*erfunden*), d.h. deren Syntax und Semantik definiert. Sein Modellieren schloß also Begriffs- und Sprachbildung ein. Aus der Semantik und den bereits bekannten Rechenregeln konnte er neue Rechenregeln *ableiten*. Das Resultat war ein Kalkül, die Differenzialrechnung, wozu Leibniz den “invertierten” Kalkül, die Integralrechnung, erfunden hat. Differenzial- und Integralrechnung werden unter der Bezeichnung *Infinitesimalrechnung* zusammengefasst. Das große Gebiet der Mathematik, das auf Differenzieren und Integrieren beruht - dazu gehören u.a. die Differenzialgleichungen - wird *Analysis* genannt.[2] Die Aufstellung einer Differenzialgleichung zur Beschreibung eines Prozesses (z.B. der Bewegung der Erde, eines Pendels oder eines Schiffes) verlangt intuitive Intelligenz; ihre Lösung verlangt, falls der Lösungsweg bekannt ist, deduktive Intelligenz, andernfalls deduktive und intuitive Intelligenz.

Es ist nicht zu leugnen, dass die getroffene Unterscheidung zwischen drei Arten natürlicher Intelligenz (Deduktion, Assoziation, Intuition) etwas unmotiviert, zumin-

---

2 Es sei schon jetzt darauf hingewiesen, dass wir in Kap. 15.8 mit “*analytischem Rechen*” jedes Rechnen mit Variablen bezeichnen werden, also weit mehr, als nur das Rechnen im “Kalkül der Analysis”.

dest unüblich erscheinen kann und dass sie sich auch nicht in allen Punkten mit dem gängigen Lehrbuchwissen im Einklang befindet. So ist es beispielsweise nicht üblich, zwischen produktiver und reproduktiver Intelligenz zu unterscheiden. Andrerseits wird gewöhnlich der Deduktion die *Induktion* gegenübergestellt, die hier gar nicht erwähnt worden ist (der Grund wird in Kap.21.4 [21.8] genannt). Doch ist die getroffene Klassifikation im Hinblick auf die künstliche Intelligenz zweckmäßig, denn auf ihrer Grundlage lassen sich die oft gestellten Fragen nach Inhalt, Wesen und Grenzen der künstlichen Intelligenz präzisieren und zumindest partiell sinnvoll beantworten. So werden wir uns in den Kapiteln 15 und 16 genauer überlegen, was Programme leisten müssen, um den Computer zum Rechnen und Schlussfolgern, d.h. zum Deduzieren zu befähigen.

Das eingangs beschriebene Gegenläufigkeitsphänomen der natürlichen und künstlichen Intelligenz lässt sich nun so formulieren: Die künstliche Intelligenz ist von der berechnenden zur schlussfolgernden Stufe "*aufgestiegen*", und manches spricht dafür, dass sie vielleicht auch die intuitive Stufe erklimmen wird. Die natürliche Intelligenz dagegen beginnt ihren "*Aufstieg*" auf der intuitiven Stufe, um über die Stufe des Schlussfolgerns schließlich die des Rechnens zu erreichen.

Da die Evolution der natürlichen Intelligenz nicht mit der Intuition begonnen haben kann, ist die Frage berechtigt, worauf diese ihrerseits fußt, woraus sie sich entwickelt hat. Die naheliegende Anwort lautet: aus den Instinkten. Instinkte sind - ebenso wie die Intuition - unbewusste Auslöser bestimmter Verhaltens- oder Handlungsabläufe (im Gegensatz zu Reflexen, die scharf begrenzte Reaktionen darstellen wie Augenzwinkern oder innere Sekretion), wobei Instinkte entwicklungsgeschichtlich sicherlich älter sind als die Intuition.

Wenn man Instinkte und Intuition nicht als "Eingebungen" vonseiten höherer Instanzen erklären will, müssen sie entweder auf genetischer oder auf individueller *Erfahrung* beruhen, genauer gesagt, die Mechanismen, auf denen Instinkte und Intuition beruhen, müssen das Ergebnis der genetischen Evolution bzw. der individuellen, intellektuellen Evolution sein. Wir werden im Weiteren von folgender (den biologischen Erkenntnissen nicht widersprechenden) Vorstellung ausgehen. ***Instinkten*** *liegen genetisch codierte Erfahrungen der Gattung zugrunde,* ***Intuitionen*** *liegen neuronal codierte erworbene, unbewusste Erfahrungen des Individuums zugrunde*.

Erworbene, unbewusste Erfahrungen können entweder unbewusst erworben sein oder sie können bewusst erworben (z.B. gelernt), sodann aber *interiorisiert* (verinnerlicht, aus dem Bewusstsein verdrängt) worden sein. In jedem Fall ist die Folge, dass Intuition ein nicht erklärbares Phänomen zu sein scheint. Die Unzugänglichkeit für die Introspektion und die daraus folgende Unerklärbarkeit ist ein charakteristisches Merkmal der Intuition, zumindest in der umgangssprachlichen Bedeutung des Wortes. Insofern scheint obige Definition hypothetischen Charakter zu haben, sodass folgende Formulierung vorzuzuziehen ist: *Interiorisierte Erfahrung führt zum* ***Erscheinungsbild*** *der Intuition*. Die Hervorhebung des Wortes "*Erscheinungsbild*" soll folgenden Umstand unterstreichen. Welche Gehirnvorgänge der Intuition zugrunde

liegen, wissen wir nicht oder nur andeutungsweise. Das Gehirn wird als *schwarzer Kasten* aufgefasst, d.h. als nichtdekomponierbarer Operator, der nur durch seine Input-Output-Relation beschrieben werden kann, weil nur sie in *Erscheinung* tritt, d.h. beobachtet werden kann.

6 Wir sind hier auf einen Sachverhalt von grundsätzlicher Bedeutung gestoßen. Er trifft nämlich nicht nur auf die Intuition, sondern auf jedes menschliche und insbesondere auf intelligentes Verhalten zu. *Die künstliche Intelligenz simuliert* ***Erscheinungsbilder***, zumindest die traditionelle KI, von der in diesem Buch die Rede ist. Unter diesem Blickwinkel gewinnt das Wort "Simulieren" seine umgangssprachliche Doppelbedeutung von "*Nach*machen *und Vor*machen", Vormachen im Sinne von Weismachen, von "so tun, als ob". Die traditionelle, symbolische Informatik, modelliert nicht die Natur, insbesondere nicht die natürliche Intelligenz, sondern Erscheinungsbilder der Natur, insbesondere die Erscheinungsbilder der natürlichen Intelligenz. Sie macht also genau dasselbe wie ein Mensch, der über Natur*erscheinungen* nachdenkt und sprachlich modelliert. Interessanterweise trägt das Wort *Erscheinung*, ebenso wie das Wort *Simulieren*, häufig eine Bedeutungskomponente von "*Anschein*" oder "*so als ob*".

Diese Überlegungen liefern zwar keine Argumente zugunsten der Simulierbarkeit menschlicher Intelligenz, doch schwächen sie die Gegenargumente. Insbesondere entkräften sie diejenigen Argumente, die von der Unerkennbarkeit dessen ausgehen, was *hinter* den Erscheinungen steht, *hinter* dem Phänomen des Lebens und *hinter* dem Phänomen der menschlichen Intelligenz. Denn der "eigentliche" Hintergrund, der Urgrund der Erscheinungen, "die Wahrheit" stehen gar nicht zur Debatte. Simuliert werden Erscheinungen. Was "wirklich ist", entzieht sich nicht nur dem Zugriff des Computers, sondern auch dem des Menschen.

Aus diesen Überlegungen ziehen wir die Berechtigung, bei der ursprünglichen Formulierung zu bleiben: *Der Intuition liegen neuronal codierte, unbewusste, vom Individuum erworbene Erfahrungen zugrunde*. Noch prägnanter formulieren wir:

7 ***Intuition*** *beruht auf nichtbewusster Verarbeitung von nichtbewusstem Wissen.* Diese Schlussfolgerung ist zwingend, wenn man von *Eingebungen* vonseiten höherer Instanzen absieht.

Wir werden die Intuition als Komponente der natürlichen Intelligenz zunächst nicht in unsere Betrachtungen einbeziehen. Lediglich zwei sehr spezielle Erscheinungsformen der Intuition werden näher untersucht, *reduzible* Intuition (Kap.16.3 [16.11]) und *scheinbare* Intuition (Kap.17.3 [17.6]). Erst in Kap.21.4 [21.5] werden wir einige allgemeinere Überlegungen hinsichtlich der Implementierbarkeit (Simulierbarkeit) der Intuition anstellen.

Die eingeführte Terminologie und die illustrierenden Beispiele provozieren folgende Frage. Die *Entdeckung* von Naturgesetzen, beispielsweise der newtonschen Prinzipien der Mechanik, ist nach obiger Systematik als intuitive Leistung, als *Erfindung* zu klassifizieren. Die Terminologie scheint die Begriffe *Entdecken* und *Erfinden* zu verwechseln oder zu vermischen. Wie sind die beiden Begriffe in die

Systematik des sprachlichen Modellierens einzuordnen und wodurch unterscheiden sie sich? Die Frage trifft in gewissem Sinne den Kern der "Evolutionären Erkenntnistheorie", und die Antwort liegt nicht auf der Hand. Die übliche Unterscheidung scheint klar zu sein: Entdecken bezieht sich auf existierende, Erfinden auf zuvor nicht existierende Objekte. Aus der Sicht des sprachlichen Modellierens wird die Grenze zwischen beiden Begriffen unklar. Man kann z.B. fragen, ob mathematisches Modellieren der Welt Entdecken oder Erfinden ist, ob physikalische Gesetze entdeckt oder erfunden werden.

Die Beobachtung eines unbekannten Naturphänomens, seine Bewusstwerdung als Idem, ist eine *Entdeckung*. Seine sprachliche Modellierung (z.B. durch eine Differenzialgleichung) ist insoweit *Erfindung*, als für die Artikulierung neue Begriffe und neue Namen erfunden werden müssen, wie z.B. der Begriff des Differenzialquotienten oder die Notation d*x*/d*t*. Auch Begriffe wie Masse oder Temperatur wurden *erfunden*. Allgemein sind Begriffe, also benannte Ideme, die keine unmittelbaren Entsprechungen in der Wirklichkeit haben (deren Idemen keine Urrealeme entsprechen), Erfindungen der Menschen. Bei der Modellierung mit Hilfe der Mathematik oder des Computers kommt als intuitive Komponente die *Interpretation* von Rechengrößen als Modellgrößen hinzu, also die Anbindung der externen an die formale (kalkülinterne) bzw. an die rechnerinterne Semantik. Auch sie ist nicht ableitbar, sondern muss erfunden werden. Insofern werden physikalische Gesetze *erfunden*.

Damit ist gezeigt, inwieweit Erfinden Bestandteil des mathematischen Modellierens und Bestandteil des Modellierens mittels Computer ist, nämlich insoweit, wie das Bilden von Begriffen und von Sprachelementen (ihrer Syntax und Semantik) und das Anbinden von externer an interne Semantik Bestandteil des Modellierens ist. 8

7.2 Beschriftung der tabula rasa. Algorithmenbegriff

Wenn man Gedächtnisleistungen, wie es üblich ist, als intelligente Leistungen auffasst, stellt sich die Frage, warum sie in den bisherigen Überlegungen kaum eine Rolle gespielt haben und wie sie in die Intelligenzsystematik einzuordnen sind. Die Antwort ergibt sich aus der Feststellung, dass wir bei der Begriffsbestimmung der Intuition und Deduktion, um die es vor allem ging, stillschweigend nur die *produktive* (modell*bildende*) Komponente als die für die Bildung *neuer* Aussagen wesentliche Komponente im Auge hatten, obwohl in jedem Falle eine *reproduktive* (modell*nutzende*) Komponente in Form von Gedächtnisleistungen beteiligt ist. 9

Gedächtnisleistungen *sind Leistungen der reproduktiven, assoziativen Intelligenz.* Ohne reproduktive Intelligenz, ohne Nutzung vorhandener Gedächtnisinhalte (vorhandenen Wissens) ist keine sinnvolle Erweiterung des Modells der Welt möglich. Darin liegt der evolutionäre Charakter der menschlichen Erkenntnis.

Hier tritt ein wesentlicher Unterschied zwischen natürlicher und künstlicher Intelligenz zutage. Der Computer ist offenbar nicht fähig, selber seine Fähigkeit zum

sprachlichen Modellieren (seine eigene Intelligenz) weiterzuentwickeln. Diese "übergeordnete" Intelligenz oder **Metaintelligenz** scheint dem Computer abzugehen, denn seine Intelligenz wird durch den Menschen gesteigert, indem dieser zusätzliche Daten und Programme einspeichert. Könnte es sein, dass der Schein trügt? In Kap.21.4 [21.7] kommen wir auf diese Frage zurück.

Wenn der Computer keine Metaintelligenz besitzt, kann seine Intelligenz nicht evolutionieren. Der Nutzer muss ihn "belehren". Sonst bleibt er so "dumm" wie er "geboren" (vom Produzenten mit Software ausgestattet) wurde. Welcher Computernutzer hat sich nicht schon über die Begriffsstutzigkeit seines Denkassistenten geärgert. Jedes Kind versteht, was man meint, der Computer nicht. Der Mensch bringt die Voraussetzungen für seine Dialogfähigkeit zum Teil in Form angeborener Gehirnstrukturen mit auf die Welt, zum anderen Teil erwirbt und verbessert er sie im Laufe seines Lebens als Folge zusätzlicher Strukturierung seines Gehirns unter dem Einfluss der Welt. Er besitzt Metaintelligenz.

In der Umgangssprache und auch in der Literatur ist mit *Intelligenz* sehr oft *Metaintelligenz* (in der Terminologie dieses Buches) gemeint. Als intelligent wird nicht derjenige bezeichnet, der sprechen (sprachlich modellieren) kann, sondern derjenige, der eine Sprache schnell lernt. Ein Schüler, dem Mathematik leichter fällt (der die Sprache der Mathematik schneller erfasst und beherrscht) als seine Mitschüler, wird als *intelligenter* eingestuft, womit gemeint ist, dass er seine Fähigkeit zum sprachlichen Modellieren schneller weiterentwickeln kann. Wir sind bei der Definition der Intelligenz stillschweigend davon ausgegangen, dass die Metaintelligenz Teil der Intelligenz ist. Dieser Standpunkt ist vernünftig, denn er führt zu einer ausreichend umfassenden Bestimmung des Intelligenzbegriffs, an der im Weiteren festgehalten wird.

Der Computer muss *vom Menschen* ausreichend intelligent gemacht werden, um dialogfähig zu sein, und zwar durch Anfüllung seines Speichers mit geeigneter Software, mit geeigneten Bitketten. Vorher ist er ein unbeschriebenes Blatt, eine tabula rasa. Mit der "angeborenen Ignoranz" des Rechners muss jeder Nutzer leben. Es wäre aber voreilig, aus dieser Erfahrung den Schluss zu ziehen, dass der Rechner prinzipiell "dumm" ist und niemals zu einem ebenbürtigen Dialogpartner des Menschen werden kann, oder, noch rigoroser, dass es überhaupt keine künstliche Intelligenz gibt und nicht geben kann. Derartige Meinungsäußerungen gründen sich auf dem genannten Umstand, dass der Nutzer selber dafür sorgen muss, dass sein Computer die notwendige Intelligenz besitzt. Er erreicht dies durch Entwicklung oder Kauf geeigneter Software, die er seinem Computer als Wissen "eintrichtert". Dabei handelt es sich primär um Produkte der *natürlichen* Intelligenz.

Diese Argumentation gegen die Bezeichnung "*künstliche Intelligenz*", gegen die sogenannte KI, ist richtig, wenn man Intelligenz von ihrer Entstehung her definiert. Wir haben sie jedoch als Eigenschaft ihres Trägers definiert. Wenn dieser künstlich, ein "Artefakt" ist, sprechen wir von künstlicher Intelligenz, ungeachtet ihres Ursprungs. Wo ihre Grenzen liegen, wissen wir damit freilich noch nicht. Die Antwort

auf die Frage, ob prinzipielle Grenzen der KI existieren, ist aus unserem derzeitigen Wissen und mit unseren derzeitigen Begriffen nicht ableitbar, sondern in gewissem Grade immer "Erfindung". Jede diesbezügliche definitive Aussage ist als Hypothese, möglicherweise als wertvolle Arbeitshypothese anzusehen.

Natürlich erscheint es fraglich, ob das Ergebnis der Jahrmillionen langen genetischen Evolution sich so "auf die Schnelle" technisch kopieren, vielleicht sogar übertreffen lässt. Aber wer kennt die Grenzen der Erfindungskraft der menschlichen Intelligenz? Wie man sieht, führt die Frage sehr schnell auf ein zirkuläres Problem: Kann der Mensch die Grenzen seiner Erkenntnisfähigkeit erkennen? Wir wollen dieser introspektiv offensichtlich nicht beantwortbaren Frage nicht weiter nachgehen, sondern noch einmal auf das Gegenläufigkeitsphänomen zurückkommen und es begründen.

Das "Eintrichtern" von Wissen und damit von Intelligenz in den Computer, das Beschreiben der tabula rasa, beginnt natürlicherweise mit dem Rechnen. Denn der Mensch kann dem Rechner nur das beibringen, was er selber genügend genau verstanden (durchschaut) und sprachlich modelliert hat. Mit "beibringen" ist hier nicht das Anlernen durch Gewöhnen ("Einpauken") gemeint, sondern das Eingeben von Daten und Vorschriften oder *Algorithmen*, nach denen der Computer zu verfahren hat, die ihn "schlau" machen.

Soeben ist einer der zentralen Begriffe der Informatik gefallen. Wir werden uns später eingehend mit Algorithmen beschäftigen, wollen den Begriff aber schon jetzt einführen. Darum unterbrechen wir die Überlegungen zur Beschriftung der tabula rasa für einen Augenblick.

*Ein **Algorithmus**[3] ist eine Handlungsvorschrift (Prozessbeschreibung), die angibt, in welcher Reihenfolge mehrere, als bekannt vorausgesetzte Einzelhandlungen oder Operationsschritte auszuführen sind, um ein bestimmtes Ziel zu erreichen. Die Handlungsfolge muss eindeutig sein und ihre Ausführung muss terminieren, d.h. nach endlich vielen Schritten enden.* Manche Autoren verzichten auf die Forderung der Endlichkeit.

Ziel der Handlung kann beispielsweise das Produkt zweier Zahlen sein. Das Ziel kann auch ein Kuchen sein; dann spricht man i. Allg. allerdings nicht von Algorithmus, sondern von Rezept. Prozessbausteine (Teilhandlungen) können beim Multiplizieren z.B. das Addieren, beim Backen das Rühren sein.

Die Entwicklung von Algorithmen und die Herausbildung des Algorithmenbegriffs haben ihre Ursache offensichtlich darin, dass der Mensch "nur *einen* Kopf" hat. Er kann sich immer nur auf die Ausführung einer einzigen Aktion voll konzentrieren. Er kann nicht gleichzeitig zwei Zahlen addieren und zwei andere voneinander subtrahieren. Eine Rechenvorschrift, die als Algorithmus artikuliert ist,

---

3 Die Bezeichnung Algorithmus soll sich von dem Namen des persischen Mathematikers und Astronomen MOHAMMED IBN MUSA AL-CHWARISMI (um 820) herleiten.

entspricht dieser Eigenschaft des Menschen. Das erklärt die Rolle, die Algorithmen seit Jahrhunderten in der Rechenkunst spielen.

Heutzutage wird der Algorithmenbegriff häufig mit der Rechentechnik in Verbindung gebracht. Beispielsweise findet man im Informatik-Duden [Duden 89] die Definition: "*Unter einem Algorithmus versteht man eine Verarbeitungsvorschrift, die so präzise formuliert ist, dass sie von einem mechanisch oder elektronisch arbeitenden Gerät durchgeführt werden kann.*" Wir bleiben bei der zuvor gegebenen "klassischen" Definition, ergänzen sie aber durch Einführung eines spezielleren Algorithmenbegriffs.

Der engere Begriff ergibt sich aus dem allgemeineren dadurch, dass in der obigen Definition das Wort *Operationsschritt* durch *Aktion* ersetzt wird. *Als **Aktion** bezeichnen wir eine Operation an einem oder an mehreren bestimmten, **explizit ausgewiesenen** Operanden.* Eine Aktionsvorschrift wäre beispielsweise "Addiere die Werte von *a* und *b*" oder "Verrühre 1 Esslöffel Quark mit 1 Esslöffel Mehl". Diese Sätze sind vollständige *Imperativsätze*. Je ein Imperativsatz eines Algorithmus schreibt je
10 eine Aktion vor. Darum nennen wir einen *Algorithmus, der eine terminierende Folge von Aktionen vorschreibt, einen **imperativen Algorithmus***. In der Literatur ist dieser Begriff nicht üblich. Wir führen ihn aus zwei Gründen ein, zum einen der Eindeutigkeit halber, denn der Algorithmenbegriff wird in unterschiedlichen Bedeutungen verwendet, oft im Sinne des imperativen Algorithmus, oft aber auch in einem allgemeineren Sinne, wie obiges Zitat aus dem Informatik-Duden zeigt. Zum anderen soll die Einführung des Begriffs des *imperativen Programms* in Kap.13.5.1 [13.9] vorbereitet werden. Die Begriffe "*Aktion*" und "*imperativer Algorithmus*" mögen im Augenblick gekünstelt erscheinen. Ein ganz einfaches Beispiels soll verdeutlichen,
11 worauf es ankommt. Es sei der Wert von *r* nach der Vorschrift $r = (a\text{-}b)+c$ für bestimmte Werte von *a, b* und *c* zu berechnen. Dazu müssen zwei Operationen ausgeführt werden, zuerst eine Subtraktion und anschließend eine Addition. Der erste Summand der Addition ist in der Vorschrift nicht explizit angegeben; die Vorschrift stellt also *nicht* die Beschreibung einer *Aktionsfolge* dar. Um sie in eine solche zu überführen hätte man beispielsweise zu schreiben: $a\text{-}b=d$; $d+c=r$. Die Notwendigkeit dieses Vorgehens wird sich in Verbindung mit dem Programmieren von Computern herausstellen (vgl. Kap.13.5.2 [13.11]).

Nach diesen Bemerkungen zum Begriff des Algorithmus wenden wir uns wieder der Beschriftung der tabula rasa des Computers mit Rechenvorschriften zu. Vorbild ist das "Beschriften der tabula rasa" des Menschen, das "Eintrichtern" von Wissen, speziell von Rechenvorschriften, in sein Gedächtnis. Besonders hilfreich ist es zu
12 beobachten, wie Kinder Rechnen lernen. Unabhängig davon, ob ein Kind im Klassenzimmer unter der Anleitung des Lehrers oder zu Hause am Computer lernt, beginnt alles mit dem *Zählen*, d.h. mit dem schrittweisen Erhöhen um Eins oder - in der Sprechweise der Mathematiker - mit *iterativem Inkrementieren*. Nach diesem Prinzip erfolgt beispielsweise das Addieren. Der ABC-Schütze addiert, indem er "mit

den Fingern zählt". Nach der gleichen Methode subtrahiert er auch, indem er zählt, wie oft er den Subtrahend inkrementieren muss, um den Minuend zu erreichen.

Ganz analog, nur "eine Stufe höher", wird multipliziert und dividiert, nämlich durch wiederholtes Addieren. Multiplizieren und Dividieren lassen sich also auf Addieren und weiter auf Inkrementieren zurückführen. Dabei muss ständig geprüft werden, ob die laufende Iteration fortzusetzen oder abzubrechen ist. Um das beispielsweise beim Subtrahieren zu entscheiden, muss der inkrementierte Subtrahend mit dem Minuend verglichen werden. Tatsächlich lässt sich *jede* arithmetische Operation auf Inkrementieren und Vergleichen zurückführen, oder anders ausgedrückt, *jede* ***arithmetische Operation*** *lässt sich aus Inkrementier- und Vergleichsoperationen* ***komponieren***. (Der exakte Nachweis der Richtigkeit dieser Aussage wird in den weiteren Kapiteln erbracht.)

Damit ein Computer rechnen kann, würde es im Prinzip also ausreichen, wenn er über Hardwareoperatoren (Schaltungen) für das Inkrementieren und Vergleichen verfügt und wenn ihm Vorschriften (Programme) für das Komponieren arithmetischer Operationen aus Inkrementier- und Vergleichsoperationen eingegeben sind und er diese auch ausführen (interpretieren) kann.

Computer besitzen jedoch eine "intelligentere" Hardware, als die soeben geforderte. Die Schaltung, die letzten Endes alle Operationen ausführt, heißt **arithmetisch-logische Einheit**, abgekürzt **ALU** ("U" von unit). Sie kann in der Regel die Addition, die Subtraktion, den Vergleich und einige weitere elementare Operationen ausführen.

Das Erstaunliche der Entwicklung der Rechentechnik besteht darin, dass es im Laufe der Zeit gelungen ist, die tabula rasa in einer Weise zu beschriften (Algorithmen in das Gedächtnis des Computers einzuspeichern), die ihm Fähigkeiten verleihen, die weit über das Rechnen hinausgehen, sodass man ihm Intelligenz im üblichen Sinne des Wortes zugestehen muss. Dass diese Entwicklung mit dem Addieren begonnen hat, ist nach den vorangehenden Überlegungen durchaus verständlich. Ebenso verständlich ist, dass die Entwicklung der natürlichen Intelligenz entgegengesetzt verlaufen ist und mit der Assoziation und Intuition begonnen hat. Denn die Überlebenschancen eines Individuums im Kampf ums Dasein sind wesentlich dadurch mitbestimmt, wie schnell es ihm gelingt, Gefahren zu erkennen oder nur zu wittern und richtig auf sie zu reagieren. Wie sich daraus aber die Fähigkeit zum Rechnen entwickelt hat, ist weitgehend unklar. Vielleicht könnte man es verstehen, wenn man genau wüsste, wie das Assoziieren neurophysiologisch ("hardwaremäßig") funktioniert.

Man kommt zu einer merkwürdigen Feststellung: Der Mensch kann sein eigenes Denken zwar simulieren, zumindest teilweise, er kann es aber bis heute im Grunde nicht verstehen. Verallgemeinert und etwas verkürzt kann man sagen: *Der Mensch kann mehr als er versteht.* Das war schon mit dem Fliegen so und mit der Erfindung der Dampfmaschine und mit vielen anderen Erfindungen, die zur Zeit, als sie gemacht

wurden, nicht nur nicht hergeleitet (aus bekanntem Wissen deduziert), sondern auch nicht erklärt, nicht verstanden werden konnten.

Damit ergibt sich ein zweites zirkuläres Problem: Inwieweit kann der Mensch verstehen, was er selber "kann", was er produzieren kann, und inwieweit kann er die Grenze zwischen dem, was er kann und was er nicht kann, erkennen? Offensichtlich handelt es sich auch hier um eine ausweglose Zirkularität, und offensichtlich ist der Satz "*Der Mensch kann die Grenzen des eigenen Könnens (Vermögens) nicht erkennen*" wahr. Der Satz bezieht sich nicht auf die Grenzen des Denkens und Handelns in konkreten Situationen, z.B. darauf, wie viele Zahlen ich mir merken oder wie hoch ich springen kann. Ich weiß beispielsweise, dass ich nicht 10 Meter hoch springen kann.

8 Formale Grundbegriffe und Methoden

Zusammenfassung

Ein Prozess, der sich als zeitlich diskrete Folge kausal verknüpfter Ereignisse beschreiben lässt, heißt *kausaldiskret*. Der Träger eines kausaldiskreten Prozesses heißt *kausaldiskretes System*. Es wird die *USB-Methode* (USB - Uniforme Systembeschreibung) für die Beschreibung kausaldiskreter Prozesse und Systeme eingeführt. Die Methode ist für die Komponierung hierarchisch strukturierter Hard- und Softwaresysteme in Form von Operatorenhierarchien entwickelt worden.

Ein Operator ordnet Eingabeoperanden bzw. den Werten einer Eingabevariablen x Ausgabeoperanden bzw. Werte der Ausgabevariablen y zu. Wenn die Zuordnung durch einen Menschen oder ein Gerät vorgenommen wird, heißt der Operator *real*; wenn sie durch eine Vorschrift festgelegt wird, heißt er *sprachlich*. Ein sprachlicher Operator bedarf eines realen Operators, eines *Interpretierers*, der die Vorschrift ausführt. Wenn alle Zuordnungen, die ein Operator trifft, eindeutig sind, d.h. wenn einem x-Wert (bzw. Wertetupel) genau ein y-Wert (bzw. Wertetupel) zugeordnet wird, heißt die Gesamtheit der Zuordnungen *Funktion* oder *Abbildung*. Zuweilen wird der Abbildungsbegriff in einem weiteren Sinne verwendet ("*Abbildung i.w.S.*"), der die Eindeutigkeit der Zuordnung nicht einschließt. Im Weiteren ist unter Abbildung im mathematischen Sinne stets eine Funktion, also eine "Abbildung i.e.S." (im engen Sinne) zu verstehen. Eine Funktion heißt *berechenbar*, wenn sie durch eine Vorschrift (einen sprachlichen Operator) festgelegt ist und ein Interpretierer der Vorschrift existiert oder angebbar ist.

Die Komponierung eines sog. *Kompositoperators* aus *Bausteinoperatoren* erfolgt durch deren Verbindung zu einem *Operatorennetz* (ON) mit einem Eingang und einem Ausgang. Über die Verbindungen werden Operanden übergeben. Der Operandenfluss durch ein ON wird durch sog. *Flussknoten* bestimmt, die z.T. steuerbar sind. Kompositoperatoren können als Bausteinoperatoren eines Kompositoperators höherer *Komponierungsstufe* dienen. Auf diese Weise kann eine *Operatorenhierarchie* mit *Schichtenarchitektur* aufgebaut werden. Die Operatoren einer Schicht (mit Ausnahme der untersten) werden aus Operatoren der nächst tieferen Schicht komponiert und können ihrerseits als Bausteinoperatoren zur Komponierung eines Operators der nächsthöheren Schicht dienen.

Operatoren (reale oder sprachliche), die aus einem vorgegebenen Satz elementarer Operatoren nach der USB-Methode komponierbar sind, heißen *USB-Operatoren*. Funktionen, die durch USB-Operatoren berechenbar sind, heißen *USB-Funktionen*. Jede rekursive Funktion ist als USB-Funktion unter Verwendung des Inkrementierers als einzigem elementarem Operator darstellbar.

Für die Entwicklung der elektronischen Rechentechnik waren die folgenden vier Ideen richtungweisend:

- die Idee, die boolesche Algebra als sog. *Basiskalkül* zu implementieren und alle anderen zu implementierenden Kalküle auf das Basiskalkül zurückzuführen;
- die Idee, die elementaren booleschen Operatoren als Schalternetze zu realisieren;
- die Idee, einen steuerbaren Bitkettentransformator zu realisieren, der in der Lage ist sprachliche Operatoren als Operationsvorschriften zu verstehen und auszuführen;
- die Idee, das Übersetzen sprachlicher Operatoren in die Maschinensprache durch den Computer ausführen zu lassen.

8.1 Uniforme Systembeschreibung (USB)

Als Ziel der Informatik wird zuweilen der Computer mit menschlicher Intelligenz proklamiert, die Konstruktion eines informationellen Systems, das deduktive und intuitive Intelligenz besitzt, das wie der Mensch rechnen, schlussfolgern und erfinden kann. Gemäß der historischen Entwicklung der Rechentechnik beschränken wir uns zunächst auf deduktive Intelligenz, konkret auf die Fähigkeit zum *Rechnen*. Hinsichtlich des Rechnens soll das System universell sein. Was das genau bedeutet, wird uns noch beschäftigen. Im Augenblick wollen wir damit die Vorstellung verbinden, dass das System imstande ist, alles zu berechnen, was der Mensch berechnen kann. In diesem Fall wollen wir von *universellem Rechner* sprechen, wohl wissend, dass dies eine sehr unscharfe Begriffsbestimmung ist, die früher oder später durch eine schärfere zu ersetzen ist.

Da unser Geist zu schwach ist, ein so mächtiges System in seinen Einzelheiten mit einem einzigen Gedanken zu erfassen und "in einem Stück" zu entwerfen, müssen wir versuchen, es schrittweise, am besten - nach dem Vorbild der Natur - hierarchisch (vgl.Kap.5.2 [5.2]) aufzubauen. Das bedeutet, dass das Produkt eine *Schichtenstruktur* besitzt. Jede Schicht enthält Bausteine, die bestimmte Operationen ausführen können. Diese Bausteine nennen wir **Operatoren**, abgekürzt op. Aufgabe eines Operators ist es, Eingabeoperanden in Ausgabeoperanden zu überführen. Die genaue Definition des Operatorbegriffs erfolgt zu Beginn des Kapitels 8.2.1. Die Bausteine der untersten Schicht heißen **elementare Operatoren**. Aus ihnen werden neue Operatoren komponiert, indem sie zu Netzen, sog.**Operatorennetzen**, abgekürzt ON, verbunden werden. Die ON werden mit je einem Eingang und einem Ausgang versehen, sodass sie die Rolle von Operatoren spielen und ihrerseits zu Netzen verbunden werden können.

Durch schrittweises Komponieren kann eine **Operatorenhierarchie** mit Schichtenstrukur, kurz eine **Schichtenarchitektur** aufgebaut werden. Die Operatoren einer Schicht (mit Ausnahme der untersten) werden aus Operatoren der nächsttieferen Schicht komponiert und können ihrerseits als Bausteinoperatoren zur Komponierung eines Operators der nächsthöheren Schicht dienen. Um zwischen den Bausteinen und

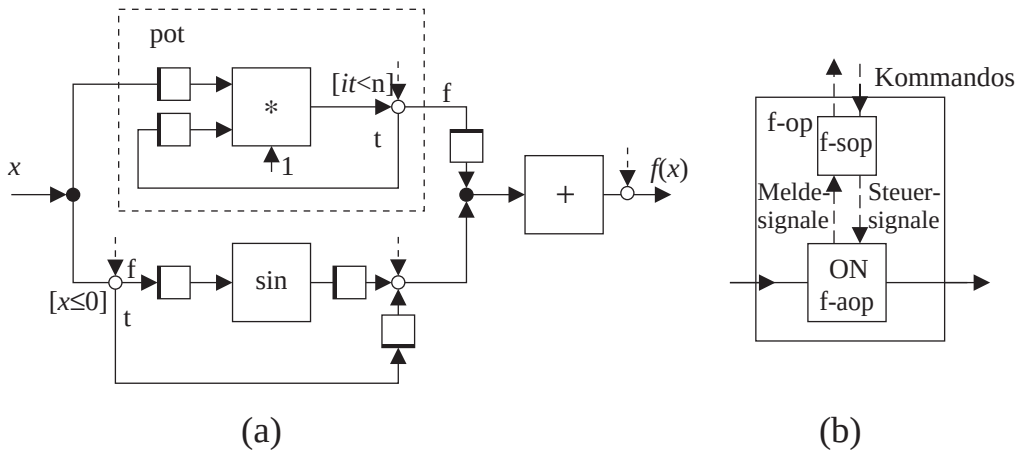

Bild 8.1 Kompositoperator zur Berechnung der Funktion (8.1). **(a)** - Darstellung als Operatorennetz (ON); große Quadrate - Operatoren; kleine Quadrate - Operandenplätze; kleine Kreise - Weichen. **(b)** - Darstellung als gesteuerter Operator; ON - das Operatorennetz von Bild (a); f-sop - Steueroperator des Operators f-op; f-aop - Arbeitsoperator des Operators f-op.

dem Ergebnis eines Komponierungsschrittes zu unterscheiden, sprechen wir von **Bausteinoperatoren** und **Kompositoperatoren** bezüglich des betreffenden Komponierungsschrittes. Je höher die Komponierungsstufe eines Kompositoperators ist, umso "höher" liegt die Schicht, der er angehört, und umso komplexer ist die Operation (die **Kompositoperation**), die er ausführt. Operationen zunehmender Komplexität sind beispielsweise das Inkrementieren, das Addieren und das Multiplizieren.

Das Komponieren eines Operatorennetzes soll anhand der beiden Funktionen

$$f_1(x) = x^n + x \quad \text{und} \quad f_2(x) = x^n + \sin x$$

demonstriert werden. Der Exponent n stellt keine Variable, sondern eine vorgebbare Konstante dar. Der erfahrene Leser wird wahrscheinlich sofort erkennen, dass sich mit Hilfe des in Bild 8.1a dargestellten Netzes die beiden Funktionen f_1 und f_2 berechnen lassen. Wir wollen die Funktionsweise des Netzes im Einzelnen verfolgen. Das wird etwas mühsam. Zur Motivation sei gesagt, dass wir nach der hier beschriebenen Vorgehensweise in Teil 2 einen universellen Rechner als Operatorenhierarchie entwerfen werden.

Das Netz von Bild 8.1a enthält je einen Bausteinoperator für das Addieren, für das Multiplizieren[1] und für die Berechnung der Sinusfunktion. Die Operatoren sind durch große Quadrate dargestellt. Die Operanden werden längs der Pfeile weitergegeben. In jedem möglichen Übergabeweg liegt ein Operandenplatz (Speicher) zur Ablage von Operanden, dargestellt als kleines Quadrat mit einer fetten Seite, durch

1 In der Rechentechnik wird für das Multiplizieren das Symbol ∗ verwendet.

die ein Operand in den Operandenplatz eintreten kann. Ein Operandenplatz kann als spezieller Operator aufgefasst werden, der einen Operanden nicht verändert, ihn aber ausreichend lange aufbewahren kann.

Der **Operandenfluss** durch das Netz kann durch **Steuersignale** gesteuert werden, die von einem **Steueroperator** (mit sop bezeichnet) generiert werden. Der Steueroperator ist Bestandteil des Kompositoperators, also einer seiner Bausteinoperatoren. Um die Operatoren des Netzes vom Steueroperator zu unterscheiden, nennen wir sie **Arbeitsoperatoren** (aop). Ein Kompositoperator besteht also in der Regel aus einem sop und einem oder mehreren aop, die zu einem Operatorennetz verbunden sind. In Bild 8.1a ist der sop nicht eingezeichnet, wohl aber in Bild 8.1b.

Die Darstellung in Bild 8.1b ist das Ergebnis einer *Abstraktion*. Es wird von den Details des Netzes abstrahiert. Das Netz wird zu einem *nicht* dekomponierten Operator, zu einem "schwarzen Kasten", dessen Inneres nicht sichtbar ist. Wir nennen diese Abstraktion **Operatorabstraktion**. Der (abstrahierte) Operator ist in Bild 8.1b mit ON bezeichnet. Er ist **steuerbar**; sein Verhalten ist unterschiedlich, je nachdem wie der Steueroperator den Operandenfluss durch das Netz der Arbeitsoperatoren steuert. Der abstrahierte Operator wird - ebenso wie seine (unsichtbaren) Komponenten - *Arbeitsoperator* des Steueroperators genannt. Das ON in Bild 8.1b ist also identisch mit dem Arbeitsoperator f-aop des Steueroperators f-sop. Durch Operatorabstraktion auf der nächsthöheren Eben gelangt man zu dem Operator f-op, indem man von der Dekomponierung in f-sop und f-aop absrahiert.

Wir wollen uns die Funktionsweise des Netzes klarmachen und uns dabei vorstellen, dass die Operatoren als elektronische Schaltungen realisiert sind. Die Pfeile stellen dann leitende Verbindungen dar. Über den *externen* Eingabepfeil, das ist der Eingabepfeil des Netzes, werde ein Eingabeoperand x eingegeben. An der durch einen dicken Punkt in der Eingabeleitung markierten Stelle, **Kopiergabel** genannt, erfolgt eine Duplizierung des Operanden x; das eine Operandenexemplar wird an den Multiplizierer weitergegeben, das andere an den Sinusoperator. Im Falle elektrischer Leitungen, welche die Operanden in Form elektrischer Spannungen weiterleiten, findet das Duplizieren durch die Existenz einer Gabelung des Leiters statt; ein spezieller Kopieroperator erübrigt sich.

Der Ausgabeoperand (das Resultat) des Multiplizierers wird mit dem des Sinusoperators in dem rechten dicken Punkt zu einem *Operandenpaar* vereinigt, zum Summandenpaar des nachfolgenden Addierers. Dieser Punkt wird **Vereinung** genannt. Es ist zu beachten, dass die Zusammenführung der Übergabepfeile *nicht* in dem "Zusammenlöten" der beiden Verbindungen besteht. Es handelt sich um eine begriffliche Vereinigung, um die *gedankliche* Zusammenfassung zweier Operanden zu *einem* **Kompositoperanden**. In Wirklichkeit laufen die beiden leitenden Verbindungen auch nach der Vereinung getrennt weiter. Allgemein kann ein Kompositoperand ein Tupel aus mehreren **Bausteinoperanden** sein.

Für das Addieren sind beide Summanden gleichzeitig erforderlich, sodass eventuell der eine Summand auf den anderen warten muss. Dafür steht ihm der Speicher

in der betreffenden Eingabeleitung des Addierers zur Verfügung. Das gilt allgemein. Die Operanden, die in einer Vereinung zu einem Paar oder einem Tupel vereint werden, müssen die Vereinung nicht gleichzeitig erreichen, sie müssen die Vereinung aber gleichzeitig verlassen. Die Vereinung fungiert also als **Synchronisierer**.

Das Pendant der Vereinung ist die **Spaltegabel**. Sie spaltet ein eintreffendes Operandentupel in Teiltupel auf, die dann getrennt voneinander weitergeleitet werden. Spalte- und Kopiergabel fassen wir unter dem Oberbegriff **Gabel** zusammen. Werden die Ausgänge einer Gabel mit den Eingängen einer Vereinung verbunden - evtl. über einen oder mehrere Operatoren -, ergibt sich eine **starre Masche**. Das Operatorennetz von Bild 8.1a besteht aus einer starren Masche und einem nachgeschalteten Addierer.

Der Sinusoperator ist durch eine steuerbare **Vorkopplung** überbrückt, d.h. er kann durch entsprechende Stellung der davor liegenden **Weiche** umgangen werden. Die Weiche ist als kleiner Kreis dargestellt. Das sich so ergebende Teilnetz stellt eine **steuerbare Masche** dar. Sie enthält zwei **Weichen**, eine **Zweigeweiche** (der linke kleine Kreis) und eine **Sammelweiche** (der rechte kleine Kreis).

Eine Weiche hat eine ähnliche Funktion wie beispielsweise eine Straßenbahnweiche. Durch das Stellen der Weichen wird bewirkt, dass entweder x oder $\sin x$ dem Addierer zugeführt wird, m.a.W. durch die Weichenstellung wird die *Alternative* "entweder x oder $\sin x$" entschieden. Darum wird eine steuerbare Masche auch **Alternativmasche** genannt. Wenn gesichert ist, dass zwischen den beiden Weichen einer Alternativmasche sich niemals mehr als ein einziger Operand befindet, genügt es, nur eine von den beiden Weichen zu stellen. Wenn die Zweigeweiche gestellt wird, kann die Sammelweiche ständig für beide Wege geöffnet sein, d.h. sie kann durch eine Vereinung ersetzt werden; wenn die Sammelweiche gestellt wird, kann die Zweigeweiche durch eine Kopiergabel ersetzt werden.

Vereinung, Gabeln und Weichen werden unter der Bezeichnung **Flussknoten** zusammengefasst. Weichen sind **steuerbare**, Vereinung und Gabeln sind **starre** Flussknoten. In Bild 8.2 sind die verschiedenen Flussknotentypen und die entsprechenden Symbole zusammengestellt. Das Symbol der Vereinung wird evtl. auch für eine Sammelweiche verwendet, wenn diese in einer Alternativmasche liegt. Der Vollständigkeit halber ist der Liste von Bild 8.2 das **Tor** als zusätzliches steuerbares Element hinzugefügt worden, obwohl es kein *Knoten* im eigentlichen Sinne ist. Mittels eines Tores kann ein Übergabeweg gesperrt oder geöffnet werden.

Es könnte sinnvoll erscheinen, die Spaltegabel als steuerbaren Flussknoten zu definieren, dessen Tupelaufspaltung durch eine Steuerinformation festgelegt wird. Doch ist das nicht notwendig. Es lässt sich nämlich zeigen, dass eine steuerbare Spaltegabel mit Hilfe der Flussknoten von Bild 8.2 realisiert werden kann und zwar in Form des in Bild 8.8b dargestellten steuerbaren Selektors. Gegebenenfalls werden die Ausgabeleitungen einer Spaltegabel mit den entsprechenden Teiltupeln beschriftet.

starre Flußknoten:

– Kopiergabel

– Spaltegabel

– Vereinung

steuerbare Flußknoten:

– Zweigeweiche

– Sammelweiche

– Tor

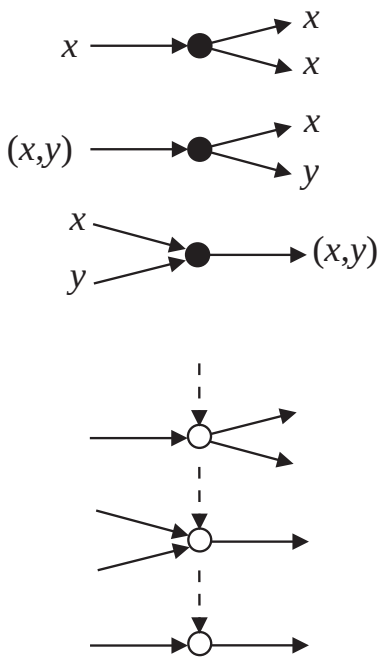

Bild 8.2: Flussknotentypen und ihre Symbole. Gestrichelte Pfeile stellen Steuersignaleingänge dar.

Der Multiplizierer in Bild 8.1a ist durch eine steuerbare **Rückkopplung** überbrückt. Im Gegensatz zu Maschen (Vorkopplungen) werden Rückkopplungen auch **Schleifen** genannt. Die Schleife in Bild 8.1a enthält eine Vereinung und eine Zweigeweiche. Die Vereinung ist allerdings ausnahmsweise nicht als Punkt dargestellt, m.a.W. die beiden Eingabeoperanden (die beiden Faktoren des Produktes) werden *nicht* gedanklich zu *einem* Kompositoperanden zusammengefasst. Diese Abweichung von obiger Regel ist eine Konzession an die bessere Lesbarkeit des Bildes. Die Darstellung der Vereinung als Punkt ist kein Zwang. Ebenso ist die Verwendung der Gabel kein Zwang, sondern ein Operator darf auch zwei oder mehrere Ausgabeleitungen besitzen. Die Einführung der starren Flussknoten dient der Einheitlichkeit der Darstellung, wodurch spätere Gedankengänge übersichtlicher werden.

Über die Rückkopplung kann das Resultat auf den Eingang zurückgeführt und die Multiplikation mehrmals ausgeführt werden. Durch entsprechende Weichenstellung wird die Anzahl der Multiplikationen, m.a.W. die Potenz bestimmt, in die x erhoben wird. Bei der ersten Multiplikation soll x ausgegeben werden, d.h. als zweiter Faktor muss als Anfangswert oder **Initialwert** zuvor eine 1 eingegeben worden sein. Das ist in der Abbildung durch den kleinen zusätzlichen Eingabepfeil am Multiplizierer angedeutet.

Welche der beiden Funktionen, f_1 oder f_2, der f-Operator berechnet und für welchen Exponenten n er sie berechnet, hängt davon ab, wie die Weichen gestellt

werden. Das erfolgt durch **Steuersignale**, die vom Steueroperator generiert und den Weichen über die gestrichelten Eingänge zugeführt werden. Dem Steueroperator muss mitgeteilt werden, für welche Funktion und für welchen Exponenten er die Steuersignale generieren soll. Damit er sie zeitgerecht generiert, kann es notwendig sein, dass ihm gemeldet wird, wann im ON eine Bausteinoperation beendet ist (durch den gestrichelten Pfeil vom ON zum f-sop in Bild 8.1b angedeutet). Der f-Operator umfasst das Operatorennetz (ON, identisch mit f-aop) und dessen Steueroperator (f-sop). Er ist ein *steuerbarer* Operator.

Ein genauerer Blick auf die Schaltung in Bild 8.1a unter Einbeziehung der Speicher zeigt, dass die Weichen dadurch gestellt werden können, dass die Eingänge der Operandenplätze (die dicken Seiten der kleinen Quadrate) als Tore ausgebildet werden, die durch Steuersignale geöffnet bzw. geschlossen werden. Eine Weiche wird dann durch entsprechende Steuerung der Tore gestellt, die in den beiden Zweigen der Weiche liegen. Dadurch wird aus der *Weichensteuerng* eine **Torsteuerung**. Diese ist der Weichensteuerung überlegen, denn sie kann nicht nur Weichen steuern, sondern in *jedem* Punkt des Netzes den Operandenfluss freigeben oder sperren. Bei der Realisierung von Operatorennetzen durch elektronische Schaltkreise (Teil 2) kommt die Torsteuerung zur Anwendung. Für die Steuerung *einer* Weiche sind zwei *Torsteuersignale* erforderlich.

Die beiden Funktionen f_1 und f_2 können zu einer Funktion f zusammengefasst werden, wenn in eindeutiger Weise festgelegt wird, für welche x-Werte f_1 und für welche f_2 berechnet werden soll, beispielsweise folgendermaßen:

$$f(x) = f_1(x) = x^n + x \qquad \text{für } x \le 0 \tag{8.1a}$$

$$f(x) = f_2(x) = x^n + \sin x \quad \text{für } x > 0. \tag{8.1b}$$

Der gesamte Kompositoperator von Bild 8.1 einschließlich seines Steueroperators kann seinerseits als Bausteinoperator eines Kompositoperators der nächsthöheren Schicht dienen. Auf diese Weise lässt sich Schritt für Schritt eine Hierarchie von Operatoren zunehmender Komplexität aufbauen, eine *Operatorenhierarchie*. Ein Operator einer bestimmten Schicht wird in der darunterliegenden Schicht in ein Operatorennetz mit seinem Steueroperator **dekomponiert**.

Der Operator von Bild 8.1 kann bereits als Hierarchie mit zwei Komponierungsebenen aufgefasst werden, denn der gestrichelt umrahmte Potenzieroperator ist in einen (allerdings nur in einen einzigen) Arbeitsoperator, den Multiplizierer und eine Rückkopplungsschleife dekomponiert. In Bild 8.1b ist der Steueroperator, der die Zweigeweiche in der Ausgabeleitung des Multiplizierers steuert, in den Steueroperator des gesamten Netzes (f-sop) integriert. Später werden wir auch den Sinusoperator dekomponieren. 2

Wir hatten uns vorgestellt, dass unser Kompositoperator als Schaltung existiert. Das Komponieren erfolgte *hardwaremäßig*. Demgegenüber war in Kap.7.2 dargelegt worden, wie der ABC-Schütze das *Komponieren nach Vorschrift* lernt. So führt er z.B. das Addieren oder Subtrahieren auf das Inkrementieren zurück, indem er “mit

den Fingern rechnet", d.h. zählt. Der universelle Rechner, den wir bauen wollen, muss offenbar auch zu dieser Art des Komponierens in der Lage sein, er muss diese Fähigkeit menschlicher Intelligenz simulieren können, denn es ist wegen des erforderlichen Aufwandes praktisch unmöglich, für jede denkbare Funktion eine Schaltung zu bauen, selbst wenn man den sparsamen Weg des Komponierens einer Operatorenhierarchie geht (siehe dazu die Abschätzungen (9.3) und (9.4) in Kap.9.2).

Das bedeutet, dass der angestrebte Rechner über mindestens einen Operator verfügen muss, der *Vorschriften ausführen* (interpretieren) kann. Die Konstruktion eines derartigen **interpretierenden Operators** oder **Interpretators** wird Inhalt von Kap.13 sein. Für ihn hat sich die Bezeichnung **Prozessor** eingebürgert. Wenn er zur Verfügung steht, kann das *hardwaremäßige* Komponieren *softwaremäßig* fortgesetzt werden, sodass eine Hardware-Software-Hierarchie entsteht.

Die graphische Darstellung der Struktur von Kompositoperatoren gemäß Bild 8.1 nennen wir **Operandenflussgraph**. Die Bausteinoperatoren können sowohl reale, z.B. elektronische, als auch sprachliche, z.B. mathematische Operatoren sein. Dabei hatten wir bisher Operatoren im Auge, die Informationen verarbeiten, genauer Zeichenketten, durch welche Informationen verschlüsselt (codiert) sind. Im folgenden Kapitel werden wir uns überzeugen, dass die Darstellungsmethode auf eine weit umfangreichere Klasse von Operatoren anwendbar ist, sodass es berechtigt erscheint, von **uniformer**, d.h. "*formal einheitlicher*" Beschreibung zu sprechen.

Wir vereinbaren: *Die Beschreibung komplexer Operatoren als Operatorennetze unter Verwendung der Flussknotentypen von Bild 8.2 bezeichnen wir als* ***uniforme Systembeschreibung****, abgekürzt* ***USB****. Ein* ***Operandenflussgraph*** *ist die graphische Darstellung der nach der USB-Methode beschriebenen Struktur eines Operatorennetzes bzw. eines Kompositoperators.*

Man beachte, dass aus einem Operandenflussgraphen nicht unbedingt die Operation abzulesen ist, die der Kompositoperator ausführt, sodass er nicht als Operationsvorschrift dienen kann. Es fehlen die Bedingungen für die Steuerung der Weichen. Um einen Operadenflussgraphen zu einer Operationsvorschrift zu vervollständigen, muss jeder steuerbare Flussknoten mit einer Steuerbedingung beschriftet werden. Dadurch wird aus dem Operandefluss*graph* ein **Operandenfluss*plan***.

Das Operatorennetz (der Operandenflussgraph) von Bild 8.1 ist ein Operandenflussplan, denn er enthält (in eckigen Klammern) die Bedingungen für die Stellung der beiden Zweigeweichen. Wenn die Bedingung $x \leq 0$ erfüllt ist, wird die Weiche vor dem Sinusoperator nach unten gestellt und der Sinusoperator umgangen. Das t an diesem Ast kommt von dem englischen true und bedeutet, dass die Bedingung erfüllt ist. Bei Nichterfüllung wird der ankommende Operand über den Zweig, der mit f (von false) beschriftet ist, dem Sinusoperator übergeben. Entsprechend ist die Bedingung $it<n$ an der Weiche nach dem Multiplizierer zu lesen, wobei *it* die *Iterationszahl* bezeichnet, d.h. die Anzahl der bereits erfolgten Multiplikationen.

Der Begriff des Operandenflussplans kann nun folgendermaßen definiert werden: *Ein* ***Operandenflussplan*** *ist ein mit den Steuerprädikaten beschrifteter Operanden-*

flussgraph, der eine vollständige Operationsvorschrift darstellt. Damit haben wir eine Abstraktionsebene bezogen, auf der ein Operandenflussplan als *zweidimensionale* Notation einer Operationsvorschrift aufgefasst werden kann. Auf dem gleichen Abstraktionsniveau können die Formeln (8.1) als *eindimensionale* (lineare) *Beschreibung* des Operandenflussplans des Kompositoperators f-op von Bild 8.1, aufgefasst werden, wobei allerdings der Potenzoperator nicht dekomponiert ist. Die Formeln (8.1) enthalten die Bedingungen für die Steuerung der Zweigeweiche vor dem Sinus-Operator in Form zweier Ungleichungen.

8.2 Kausaldiskrete Prozessbeschreibung

8.2.1 Reale und sprachliche Operatoren

Mit dem Operatorbegriff sind wir recht großzügig umgegangen, indem wir ihn für Objekte ganz unterschiedlicher Natur verwendet haben, für reale (materielle, physische) und für sprachliche Objekte. Für eine formal einheitliche, "*uniforme*" Beschreibung der Hard- und Softwarekomponierung ist ein entsprechend allgemeiner Operatorbegriff erforderlich, den wir nun herausarbeiten wollen.

Der Begriff des Operators ist sowohl in der Mathematik als auch in der Technik üblich, jedoch mit unterschiedlichen Bedeutungen. Der wesentliche Unterschied besteht darin, dass der technische Operator, z.B. ein **Fertigungsoperator**, der Werkstücke bearbeitet bzw. anfertigt, ein stofflicher Operator ist, der selbst *agiert* (wie die *agierenden* Modelle in Bild 3.1), während ein mathematischer Operator[2] nicht selbst agieren kann, sondern ein Zeichen oder eine Zeichenkette darstellt, die der Mensch als Vorschrift interpretieren und ausführen kann. Der *agierende* Operator ist in dem Falle der Mensch.

Bei der Komponierung informationeller Systeme müssen wir mit einem Operatorbegriff arbeiten, der Eigenschaften des mathematischen wie des fertigungstechnischen Operatorbegriffs in sich vereinigt. Vom mathematischen Operator übernimmt er die Natur der Operanden im Sinne der Mathematik. Jeder Operator der Hierarchie, auch das System als Ganzes, verarbeitet Zeichenketten. Sämtliche Ein- und Ausgabeoperanden sind Zeichenketten. Einen solchen Operator nennen wir **Zeichenkettenoperator**.

Vom Fertigungsoperator soll unser Operator die Fähigkeit übernehmen, selbst zu agieren, seine Operation selbst auszuführen. Beide Eigenschaften sind Voraussetzungen dafür, dass unser System zur *aktiven sprachlichen Modellierung* befähigt werden kann.

---

2 Hier und im Weiteren verwenden wir den Begriff des mathematischen Operators im weiten Sinne, d.h. nicht eingeschränkt auf die Überführung von Funktionen in Funktionen.

Etwas verwirrend ist der Umstand, dass der Begriff des Zeichenkettenoperators in der Rechentechnik nicht nur im *agierenden*, sondern auch im *nichtagierenden* Sinne verwendet wird. Je nach Kontext kann ein Operator ein Mensch, ein Gerät oder eine Vorschrift sein. Zur Unterscheidung führen wir die Begriffe des *realen* und des *sprachlichen* Operators ein und definieren:

*Ein **Operator** ist ein Mensch, eine Einrichtung (ein Gerät) oder eine Vorschrift, der/die einem Eingabeoperanden* x *einen Ausgabeoperanden* y *zuordnet. Wird die Zuordnung durch einen Menschen oder ein Gerät vorgenommen, heißt er **realer Operator**; wird sie durch eine Vorschrift festgelegt, heißt er **sprachlicher Operator**.*[3]

Die Operatordefinition sagt nichts darüber aus, ob einem bestimmten x stets dasselbe y zugeordnet wird, m.a.W. ob die Zuordnung *eindeutig* ist. Wenn sie eindeutig ist, d.h. wenn sie jedem x genau ein y zuordnet, wird die Gesamtheit aller Zuordnungen **Abbildung** oder **Funktion** genannt[4]. Zwei Notationsweisen sind üblich:

$$y = f(x), \tag{8.2a}$$

$$f : X \rightarrow Y. \tag{8.2b}$$

X bezeichnet die Menge aller x und Y die Menge aller y. Wir sagen, dass ein Operator, der die eindeutige Zuordnung (8.2) trifft, die Funktion *f realisiert*, und nennen ihn f-Operator. In der Mathematik ist es üblich, Bezeichner von Variablen kursiv zu schreiben, z.B. x und ein bestimmtes x als "Wert von x" zu bezeichnen.

Wenn wir ein informationelles System als Operatorenhierarchie entwerfen wollen, müssen wir offenbar verlangen, dass jeder Operator der Hierarchie eine eindeutige Abbildung realisiert, denn das System soll auf ein und denselben Auftrag immer gleich reagieren, es soll zu einer bestimmten Aufgabe stets dasselbe Ergebnis liefern und ein und dieselbe Frage stets gleich beantworten, allgemein, die Zuordnung der Ausgaben zu den Eingaben soll eindeutig sein. Nun lassen sich Gründe dafür angeben, dass die Zuordnungen eventuell nicht eindeutig sind. Drei Gründe seinen genannt:

1. Der Operator ist **steuerbar**. Das heißt, dass der Operator in der Lage ist, verschiedene Zuordnungen zu treffen, also verschiedene Operationen auszuführen. Vor einer Operationsausführung muss er durch ein oder mehrere Steuersignale auf eine bestimmte Operation *konditioniert* (eingestellt) werden. Der f-Operator in Bild 8.1 ist ein Beispiel dafür; er kann auf die Berechnung von f_1 oder f_2 konditioniert

---

3 Wir ziehen die Adjektive "real" und "sprachlich" den in der Literatur häufig verwendeten Adjektiven "physisch" und "logisch" vor.

4 In der Literatur wird zuweilen ein Unterschied zwischen Abbildung und Funktion gemacht. Entweder wird der Abbildungsbegriff allgemeiner oder der Funktionsbegriff wird spezieller definiert, z.B. als Abbildung zwischen Mengen von Zahlen oder als Abbildung, die durch eine Berechnungsvorschrift oder durch ein Prädikat (s.u.) festgelegt ist.

werden. Diese Art einer nichteindeutigen Abbildung wird zu einer eindeutigen Abbildung, wenn einem Eingabeoperanden *x* zusammen mit einem Steuersignal *u* (einem Kommando in Bild 8.1b) stets ein und dasselbe Resultat *y* zugeordnet wird. Dann realisiert der Operator eine Funktion

$$f: X \times U \rightarrow Y \tag{8.3}$$

Auf der linken Seite des Pfeils ist die Menge aller Paare (*x*,*u*) angegeben, notiert als sogenanntes *kartesisches Produkt* der Mengen *X* und *U*, wobei *U* die Menge aller Steuersignale (Kommanndos) *u* bezeichnet.

2. Der Operator besitzt ein **Gedächtnis**. Das bedeutet, dass außer dem momentanen auch frühere Eingabeoperanden darauf einwirken können, welches Resultat ausgegeben wird. Es stellt sich die Frage, unter welchen Bedingungen ein Operator mit Gedächtnis wie ein Operator ohne Gedächtnis behandelt werden kann, m.a.W. ob sich die Verhaltensweise eines Operators trotz seines Gedächtnisses als Funktion darstellen lässt. Es liegt der Gedanke nahe zu versuchen, die Notation (8.3) anzuwenden, worin *U* die Vergangenheit eindeutig charakterisieren müsste. Wenn wir ein- 3
deutig arbeitende informationelle Systeme als Operatorenhierarchien realisieren wollen, muss eine Notation der Form möglich sein, denn ein Kompositoperator enthält Speicherplätze für die Operanden, er ist also ein Operator mit Gedächtnis. Auf einer höheren Abstraktionsebene wird das Problem mit Hilfe des Automatenbegriffs behandelt (siehe Kap.8.2.3).

3. Der Operator ist ein **nichtdeterministischer** Operator. Ein Operator heißt **deterministisch**, wenn er unter gleichen Bedingungen (gleicher Eingabeoperand, gleiche Steuerung, gleiche Vorgeschichte) ein und demselben *x* stets das gleiche *y* zuordnet. Anderfalls heißt er **nichtdeterministisch**. Es lässt sich nicht vorhersagen, welches *y* ein nichtdeterministischer Operator einem gegebenen *x* zuordnet. Falls angegeben werden kann, mit welchen Wahrscheinlichkeiten die verschiedenen möglichen Ausgaben erfolgen, heißt der Operator **stochastisch**. Wenn nichts Gegenteiliges gesagt wird, ist im Weiteren unter einem Operator stets ein *deterministischer* Operator zu verstehen.

Hinsichtlich der ersten der genannten Interpretationen von (8.3) ist eine Ergänzung erforderlich. Bei Anwendung von (8.3) auf den Operator f-op von Bild 8.1b enthält *U* zwei Elemente, u_1 und u_2. Diese Steuersignale können dem f-sop eingegeben werden, der gemäß dem empfangenen Steuersignal die Berechnung von f_1 bzw. f_2 steuert, indem er die entsprechenden Steuersignale für die Weichensteuerung generiert. Offensichtlich ist (8.3) auch hinsichtlich dieser, vom f-sop generierten Signale anwendbar, also derjenigen Steuersignale, die nicht dem f-sop, sondern dem f-aop (dem ON) eingegeben werden. Die Elemente von *U* sind dann keine Steuersignale, sondern Tupel von Steuersignalen bzw. zeitliche Folgen solcher Tupel. Im Falle eines Operatorennetzes mit vielen steuerbaren Flussknoten können die einzelnen Steuersignaltupel und die Tupelfolgen sehr lang werden. Wenn der Steueroperator ein *interpretierender* Operator ist, der die Steuersignale für eine Operationsaus-

führung nach Maßgabe einer Operationsvorschrift (eines *sprachlichen* Operators, eines Programms) generiert, dann kann *U* eine Menge von Programmen bezeichnen.

Wenn ein sprachlicher Operator als Operationsvorschrift interpretiert werden soll, muss er den *Prozess* der Operationsausführung vollständig beschreiben. Das wirft eine unerwartete Frage auf. Prozesse verlaufen in der Zeit. Die Zeit muss also in der Beschreibung enthalten sein. Wo aber verbirgt sich die Zeit in einem Operandenflussplan, in einem sprachlichen Operator, in einem Algorithmus, in einem Computerprogramm, einem mathematischen Ausdruck oder gar in einem Operationssymbol?

Zur Beantwortung dieser Frage betrachten wir einen nichtdekomponierten Operator im Sinne der Operatorabstraktion als "schwarzen Kasten", d.h. wir abstrahieren von allem, was *in* ihm vorgeht. Das einzige, was wir "sehen", ist das Erscheinen von Operanden an seinem Eingang und Ausgang. Die *Ereignisse* des Auftretens von Operanden treten in bestimmten Zeitpunkten ein, durch welche die Zeit zerteilt (*segmentiert*) wird. Die Zeit kann als Folge dieser Ereigniszeitpunkte, beschrieben werden. Dies ist der Kern einer viel verwendeten Methode, den zeitlichen Ablauf von Prozessen zu beschreiben. Man spricht zuweilen von *ereignisorientierter Diskretisierung* der Zeit und von **ereignisorientierter Prozessbeschreibung**.

4 Wesentlich für diese Beschreibungsmethode ist ihre *kausale* Grundlage. Der Ausgabeoperand eines realen Operators ist die *kausale Folge* des Eingabeoperanden. Der Eingabeoperand ist die *Ursache*, der Ausgabeoperand die *Wirkung*. Dabei "*überspringt*" der kausale Zusammenhang die Prozesse im schwarzen Kasten. Die Beschreibung ist **kausaldiskret**.

Der *kausale* Zusammenhang zwischen Ein- und Ausgabeoperanden besteht auch im Falle sprachlicher Operatoren, wenn diese als Operationsvorschriften interpretiert werden. Denn sprachliche Operatoren werden zusammen mit ihren Interpretierern zu realen Operatoren und durch die Interpretation werden sprachliche Operanden zu realen Operanden. Eben dies ist gemeint, wenn von *Realisierung* sprachlicher Operatoren, Operationen und Operanden gesprochen wird.

Die Aussage des letzten Absatzes ist von tiefer, man kann sogar sagen von
5 philosophischer Bedeutung. Sie bedeutet nämlich, dass ***logische** Zusammenhänge im Grunde **kausale** Zusammenhänge sind*. Dieser Sachverhalt ist die Folge des Umstandes, dass ein sprachlich (z.B. mathematisch) artikulierter "logischer" Zusammenhang erst dann Sinn erhält, wenn er von einem realen Interpretator interpretiert wird. Die Schlussfolgerung mag überraschen und vielleicht sogar Protest hervorrufen. Wir werden uns mit ihr in Kap.8.2.5 auseinandersetzen.

Der Begriff der *kausaldiskreten* Prozessbeschreibung wurde bereits in Kap.4.2 [4.4] eingeführt, wo er der *kausalkontinuierlichen* Beschreibung gegenübergestellt wurde. Vergegenwärtigt man sich noch einmal die dortigen Überlegungen, erkennt man, dass Prozesse der Informationsverarbeitung nicht kausalkontinuierlich beschrieben werden müssen, sondern dass die kausaldiskrete Beschreibung ausreicht. Einem Ereignis, das einen Zeitpunkt der kausaldiskreten Beschreibung festlegt,

entspricht der Übergang des Trägermediums in einen neuen codierenden Zustand. Das wird uns im Weiteren noch beschäftigen. Im Augenblick konstatieren wir, dass Prozesse nur dann als Träger informationeller Prozesse geeignet sind, wenn sie kausaldiskret beschreibbar sind, und treffen folgende Vereinbarung: *Die Beschreibung des Ablaufs von Prozessen heißt* ***kausaldiskret****, wenn die Zeit durch Ereignisse des Auftretens von Operanden segmentiert wird und wenn jedes Ereignis die* ***kausale*** *Folge eines oder mehrerer vorangehender Ereignisse ist.* Danach ist die algorithmische Beschreibung und ebenso die Beschreibung durch einen Operandenflussplan eine kausaldiskrete Beschreibung der Ausführung einer Kompositoperation.

Die Bezeichnung “kausaldiskret” übertragen wir auf die beschriebenen Prozesse selber und werden (verkürzt) von *kausaldiskreten Operationen* und *Prozessen* sprechen, obwohl diese nicht selber, sondern ihre Beschreibungen kausaldiskret sind. Das kann zu keinen Missverständnissen führen, zumal es in Wirklichkeit gar keine kausaldiskreten Prozesse gibt, wenigstens nicht im Rahmen der klassischen Physik. Der Ursache-Wirkung-Zusammenhang ist im Grunde immer ein kontinuierlicher und kann seit NEWTON als solcher mit Hilfe der Infinitesimalrechnung beschrieben werden. Weiterhin übertragen wir die Bezeichnung auch auf Operatoren und nennen einen Operator kausaldiskret, wenn in ihm ein kausaldiskret *beschriebener* Prozess abläuft.

Die USB-Methode ist, soweit wir sie bisher kennengelernt haben, eine kausaldiskrete Beschreibung von Prozessen der Zeichenkettenverarbeitung. Bild 8.3 zeigt als weiteres Beispiel für die Anwendung der Methode das Operatorennetz eines dekomponierten, kausaldiskreten Operators, dessen Bausteinoperatoren im Gegesatz zu dem Netz von Bild 8.1 keine Zeichenkettenoperatoren, sondern *Fertigungsoperatoren* (*Werkstückoperatoren*) sind. Das Netz besitzt fast die gleiche Struktur, wie das von Bild 8.1a.

Der in Bild 8.3 dargestellte Kompositoperator dient der Herstellung von zwei verschiedenen Werkstücken. Zunächst zersägt (oder zerschlägt) der Trennautomat

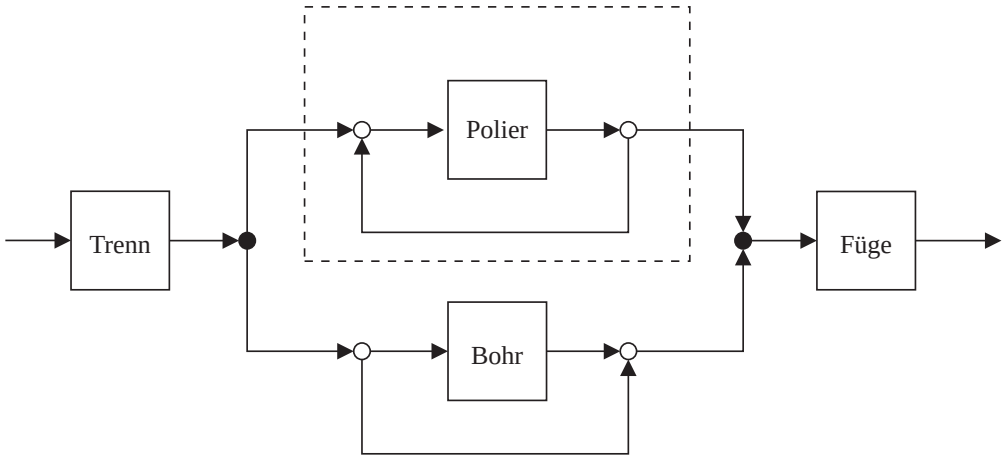

Bild 8.3: Operatorennetz eines Fertigungsoperators

(Trenn) ein Eisenblech in zwei Teile, die später der Fügeautomat (Füge) zu einem Eisenwinkel zusammenfügt, z.B. schweißt. Trenn liefert ein Operandenpaar, das in der nachfolgenden Spaltegabel aufgeteilt wird. Gabeln ohne Trennoperator, wie etwa die Gabelung des Leiters in Bild 8.1, kann es in Fertigungssystemen nicht geben.

Auf dem Wege zum Fügeoperator wird das eine Blech poliert (evtl. mehrmals) und das andere je nach Weichenstellung (je nach gewünschtem Produkttyp) entweder mit Bohrungen versehen oder unbearbeitet weitergeleitet. Vor dem Fügen muss eventuell das eine Blech auf das andere warten. Die Vereinung fungiert also ebenso wie in Bild 8.1a gleichzeitig als Synchronisierer.

Die Prozesse des Polierens und des Bohrens sind voneinander unabhängig, keiner ist die (kausale) Voraussetzung des anderen. Dasselbe gilt für das Potenzieren und das Berechnen der Sinusfunktion in Bild 8.1. Solche Prozesse heißen **nebenläufig**. Nebenläufige Prozesse können parallel (gleichzeitig) ausgeführt werden, wodurch die Arbeitsgeschwindigkeit eines Operators eventuell gesteigert werden kann.

Wenn beispielsweise das Bohren mehr Zeit in Anspruch nimmt als das Polieren, kann die Produktivität des Kompositoperators dadurch gesteigert werden, dass während des Bohrens gleichzeitig Eisenwinkel ohne Bohrungen hergestellt werden. Man hat es dann mit **parallelen** Prozessen zu tun. Dabei kann es vor der Sammel-
6 weiche nach dem Bohrer zu einem **Konflikt** kommen, wenn gleichzeitig zwei Operanden eintreffen, die von der Weiche nur der Reihe nach durchgelassen werden können. Die Sammelweiche fungiert dann als **Sequenzierer**. Vor einer sequenzierenden Sammelweiche muss eventuell ein Operand solange warten, bis für ihn die Weiche geöffnet wird. Das ist der Fall, sobald der nachfolgende Operator die zur Zeit laufende Operation beendet hat. Für das Warten muss eine Ablage (Operandenplatz, Speicher) vorgesehen sein, ebenso wie vor der Vereinung. In Bild 8.3 sind die Ablagen nicht eingezeichnet.

Wenn in Analogie hierzu die Berechnung der Sinusfunktion in Bild 8.1a länger dauert als das Potenzieren, kann eventuell die Rechengeschwindigkeit dadurch erhöht werden, dass der Operandenfluss durch die beiden Wege der Masche parallel hindurchgeleitet wird und während *einer* Operationsausführung des Sinusoperators *mehrere* Werte der Funktion f_1 (siehe (8.1)) berechnet werden. Die Sammelweiche nach dem Sinusoperator muss dann als Sequenzierer fungieren, und in ihren beiden Eingabeleitungen müssen Speicher vorgesehen sein.

Die beiden Operatorennetze in den Bildern 8.1 und 8.3 beschreiben Systeme ganz unterschiedlicher Natur, ein informationelles System und ein Fertigungssystem. Dennoch stimmen die Beschreibungen in zwei Punkten überein, sie sind beide kausaldiskret, und sie verwenden beide die gleichen Komponierungsmittel, nämlich die Flussknoten von Bild 8.2. Diese Übereinstimmung rechtfertigt die Bezeichnung
7 "*uniforme* Systembeschreibung". *Die **USB-Methode** ist auf jeden kausaldiskret beschreibbaren Prozess anwendbar*.

Die Methode kommt den Denkgewohnheiten von Informatikern einerseits und von Systemingenieuren andrerseits entgegen. Unter *Systemingenieuren* sollen hier

Ingenieure verstanden werden, die reale Systeme, die sich kausaldiskret beschreiben lassen, entwerfen, bauen oder warten. Für die Verständigung zwischen Systemingenieuren und Informatikern liefert die USB-Methode eine geeignete sprachliche Grundlage. Eine schnelle und fehlerfreie Verständigung kann für eine Produktion, in der Systemingenieure auf die Zusammenarbeit mit Informatikern angewiesen sind, von großer Bedeutung sein.

Neben der praktischen Stärke der USB-Methode kann ihre theoretische Schwäche nicht übersehen werden. Ihr Abstraktionsgrad reicht nicht aus, um kausaldiskrete Prozesse theoretisch zu untersuchen; doch reicht er aus, um einen Berechenbarkeitsbegriff zu definieren und ihn mit entsprechenden anderen Begriffen zu vergleichen. Darauf werden wir im Weiteren ausführlich eingehen. In Vorbereitung darauf wird in Kap.8.3 der Begriff der Berechenbarkeit eingeführt. Zuvor sollen noch zwei wichtige Methoden zur Sprache kommen, die ebenfalls der kausaldiskreten Prozessbeschreibung dienen, jedoch auf einer höheren Abstraktionsebene, als es bisher geschehen ist.

8.2.2 Petrinetze

Zunächst vergegenwärtigen wir uns, welche Rolle die Zeit in einer Prozessbeschreibung durch einen Algorithmus bzw. durch ein Operatorennetz spielt. Ein Algorithmus im ursprünglichen Sinne des Wortes sequenziert die Bausteinoperationen einer Kompositoperation vollständig. Bei der Operationsausführung gibt es keine gleichzeitige Operationsausführung, keine *parallelen Prozesse*. Die Bausteinprozesse sind *zeitlich vollständig geordnet*, sie bilden eine **vollständige zeitliche Ordnung**. In einem Operatorennetz dagegen können nebenläufige Operationen parallel (gleichzeitig) ausgeführt werden. Man sagt dann, dass die Bausteinprozesse eine **zeitliche Halbordnung** bilden. In jedem Fall beschränkt sich die Rolle der Zeit auf die einer “Ordnungsstifterin”. Eine andere Rolle spielt die Zeit nicht.

Bei mehrfacher, insbesondere bei geschachtelter Parallelität (Parallelität in geschachtelten Maschen) kann der zeitliche Ablauf (die zeitliche Ordnung) sehr unübersichtlich werden. Es kann zu kritischen Situationen kommen. Vor einer Vereinung (Synchronisierung) oder vor einer Sammelweiche (Sequenzierung) können sich die Operanden stauen und eine *Warteschlange* bilden (wie im Straßenverkehr). Es kann zu *Konflikten* und sogar zu *Blockierungen* kommen und zwar dann, wenn eine Bedingung für die Fortsetzung des laufenden Prozesses nicht erfüllt werden kann, z.B. wenn von zwei Operatoren jeder auf den Ausgabeoperanden des anderen wartet.

Konflikte können den Ablauf paralleler Prozesse empfindlich stören. Das gilt auch für informationelle Prozesse. Um derartige Störungen vorauszusehen und nach Möglichkeit auszuschließen, sind spezielle Analysemethoden entwickelt worden. Eine von ihnen ist die *Petrinetz-Methode*. Sie ist der USB-Methode verwandt, jedoch abstrakter und für theoretische Untersuchungen geeigneter. Sie bietet sich an, wenn man wissen möchte, ob der Prozessablauf in einem Operatorennetz infolge paralleler Operationsausführung behindert werden oder sogar zum Erliegen kommen kann. Die

Petrinetz-Methode beschreibt ausschließlich die zeitliche Ordnung der Bausteinprozesse und abstrahiert von allem anderen. Die Methode soll in aller Kürze beschrieben werden.

Es sind zwei Arten von Bedingungen zu unterscheiden, die erfüllt sein müssen, damit eine Operation ausgeführt werden kann, Operandenbedingungen und Operatorbedingungen. Die Operandenbedingung einer Operation ist erfüllt, wenn die erforderlichen Eingabeoperanden vorhanden sind. Die Operatorbedingung ist erfüllt, wenn der Operator, der Träger der Operation, frei ist. Von dem Unterschied zwischen beiden Arten von Bedingungen wird abstrahiert.

Außerdem wird davon abstrahiert, wodurch und auf welchem Wege Bedingungen erfüllt werden. Es wird lediglich das *Ereignis* betrachtet, *dass* eine Bedingung erfüllt ist, die vorher nicht erfüllt war, m.a.W. dass ein neuer **Bedingungszustand** eingetreten ist. Zur Fixierung dieser Idee wird der Begriff der **Transition** (in gewissem Sinne eine Idealisierung des Operatorbegriffs) eingeführt und gesagt: Wenn eine Transition *schaltet*, geht das Netz in einen neuen Bedingungszustand über. Ein Prozess wird als Folge von *Schaltereignissen* beschrieben. Die Beschreibung ist kausaldiskret.

Bild 8.4 demonstriert an zwei Beispielen die übliche graphische Veranschaulichung dieser Vorstellungs- und Redeweise. Transitionen werden durch Quadrate (zuweilen auch durch kurze fette Querstriche) und Bedingungen durch Kreise dargestellt, die **Plätze** genannt werden. Wenn ein Platz mit einer **Marke** (einem dicken Punkt) belegt ist, bedeutet dies, dass die betreffende Bedingung erfüllt ist. Zwischen zwei Plätzen liegt jeweils eine Transition und zwischen zwei Transitionen ein Platz. Wenn alle Eingabeplätze einer Transition belegt sind, kann sie "*schalten*" (oft wird auch "*feuern*" gesagt). Dabei wird den Eingabeplätzen je eine Marke entnommen und jeder Ausgabeplatz wird mit je einer Marke belegt. Wenn ein Petrinetz mit einer ausreichenden Anzahl von Marken gestartet wird, kommt es zu einem Markenfluss, d.h. zu einem *Fluss erfüllter Bedingungen.*

Eine Transition kann als *Markenoperator* und ein Petrinetz als *Markenoperatorennetz* aufgefasst werden, dessen Weichen in die Plätze und dessen Vereinungen und Gabeln in die Transitionen integriert sind. Die Steuerung des Markenflusses wird durch die Darstellung *nicht* erfasst.

Wie unschwer zu erkennen ist, befindet sich das Petrinetz von Bild 8.4a in einem blockierten Zustand; er wird in der Betriebssystemtechnik auch **Deadlock** genannt ("tödlicher" Konflikt). Jede Transition wartet darauf, dass die andere den mittleren Platz mit einer Marke belegt. Der dargestellte Bedingungszustand lässt sich durch die unterschiedlichsten realen Situationen interpretieren.

8 Zur Anregung der Phantasie des Lesers seien einige Situationen genannt, die sich durch Bild 8.4a beschreiben lassen.

- Zwei Autos begegnen sich auf einer Brücke, die kein Ausweichen erlaubt. Einer versperrten Wegstrecke entspricht im Petrinetz ein Platz ohne Marke. In den

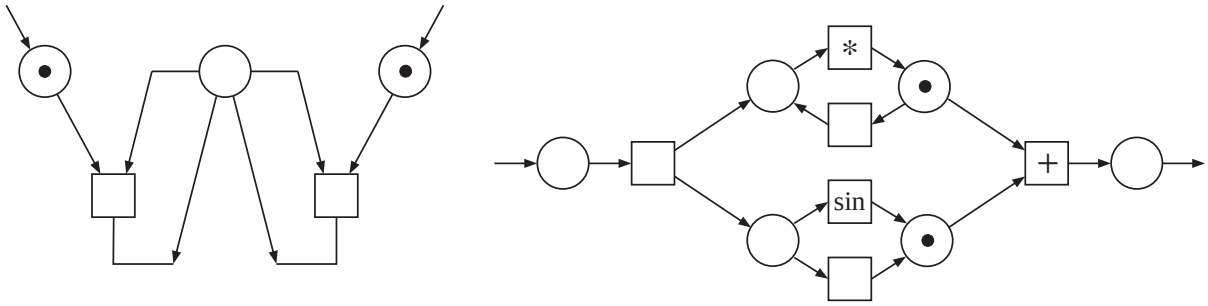

Bild 8.4 Petrinetze. **(a)** - Blockierungssituation dargestellt mittels Petrinetz; **(b)** - Petrinetz zu den Operatorennetzen der Bilder 8.1 und 8.3.

weiteren Beispielen werden die Wegstrecken auf der Brücke durch andere Objekte ersetzt (Werkzeuge, Gabeln, Geld, Betriebsmittel).

- Auf Montage ziehen zwei Monteure je ein Paar von Kontermuttern an. Dafür benötigt jeder von ihnen die beiden einzigen vorhandenen Schraubenschlüssel. Jeder hält bereits einen in der Hand.
- Wie im vorangehenden Beispiel, doch sind es zwei Fischesser (Fisch wird mit zwei Gabeln gegessen).
- Eine vorhandene Geldmenge ist für den Bau von zwei Fabriken ausgegeben worden. Um die Produktion aufnehmen zu können, benötigen beide einen zusätzlichen Betrag, der durch die Produktion erwirtschaftet werden könnte.
- In einem Computer laufen zwei Rechenprozesse ab. Beiden ist Speicherplatz zugeteilt worden. Beide benötigen zur Beendigung der Rechnung zusätzlichen Speicherplatz, der nicht vorhanden ist.

Die Vielfältigkeit der Beispiele macht die Bedeutung von Petrinetzen als Analysemittel unterschiedlichster Prozesse, insbesondere auch informationeller Prozesse verständlich. Um das noch deutlicher zu machen, soll Bild 8.4a unter Verwendung des Wortes *Betriebsmittel* kommentiert werden. *Unter **Betriebsmittel** oder **Ressource** wird in der Informatik jede Komponente der Hardware oder Software eines informationellen Systems verstanden, die der Ausführung von Operationen durch das System dient*. Zu den Betriebsmitteln gehören Programme, Daten (Dateien), Prozessoren, Speicher, Signale sowie Ein- und Ausgabegeräte.

Den Transitionen von Bild 8.4a mögen zwei Prozesse (Ausführungen von zwei Programmen) entsprechen, die gleichzeitig in einem Computer ausgeführt werden. Die Plätze entsprechen benötigten und die Marken verfügbaren Betriebsmitteln. Jeder der beiden Prozesse benötigt zwei Ressourcen, jedem Prozess ist bereits eine Ressource fest zugewiesen und beide warten auf eine gemeinsame Ressource, die

aber nicht verfügbar ist, solange nicht einer der beiden Prozesse stattgefunden (eine der beiden Transitionen geschaltet) hat. Das Betriebsmittel, auf das gewartet wird, kann z.B. das Resultat einer Rechnung sein oder - wie in obigem Beispiel - ein Speicherbereich, der bei Beendigung des einen oder anderen Prozesses frei wird.

9 Blockierungen müssen - wie alle Konflikte - durch Absprache (*dezentrale Steuerung*) oder durch eine höhere Instanz (*zentrale Steuerung*) gelöst werden. Das kann bei komplizierten und stark vernetzten Prozessen schwierig sein. Vernünftigerweise entwirft man ein System von vornherein so, dass derartige Schwierigkeiten nicht auftreten können. Dies ist eines der Ziele, denen die Theorie der Petrinetze und ihre Implementierung dient. Implementierung der Theorie bedeutet: Installation von Programmen auf einem Computer, die es erlauben, den Markenfluss in umfangreichen Petrinetzen nach den Vorgaben der Theorie durchzuspielen[5]. Auf diese Weise lässt sich voraussagen (berechnen oder simulieren), ob in einem vorgegebenen Netz gestartete Prozesse zu Blockierungen führen können.

Bild 8.4b zeigt das Petrinetz zum Operatorennetz (Operandenflussgraphen) von Bild 8.1. Es ist gleichzeitig das Petrinetz zum Operatorennetz von Bild 8.3. Die Operationssymbole ∗, sin und + sind in die entsprechenden Transitionen eingetragen. Wir gehen davon aus, dass für jede Operation ein spezieller Operator zur Verfügung steht. Das bedeutet, dass die Bedingungen ausschließlich in der Verfügbarkeit der jeweiligen Operanden bestehen. Hinter den Marken verbergen sich also Operanden und mit den Marken bewegen sich Operanden durch das Netz (obwohl Marken begrifflich etwas ganz anderes sind als Operanden). Dies ist der Grund dafür, dass Petrinetz und Operandenflussgraph einander recht ähnlich, wenn auch nicht miteinander identisch sind. Die Gabel in Bild 8.1a (ihr entspricht in Bild 8.3 der Trennoperator) ist im Petrinetz zu einer Transition geworden. Außerdem enthält das Petrinetz zwei zusätzliche Transitionen, denen keine Operatoren, sondern Operanden*wege* entsprechen und zwar die Rückkopplung vom Ausgang zum Eingang des Multiplizieres bzw. Polierers und die Vorkopplung, die den Sinusoperator bzw. Bohrer überbrückt. Die zusätzlichen Transitionen müssen eingefügt werden, um die Vorschrift zu erfüllen, dass zwischen zwei Plätzen eine Transition liegen muss. Die Markenbelegung entspricht einer Situation, in der ein Operand gerade den Multiplizierer durchlaufen und ein anderer den Sinusoperator durch- bzw. umlaufen hat. Es kann entweder die dem Addierer entsprechende Transition oder die unter dem Multiplizierer liegende Transition schalten.

Infolge der Ähnlichkeit von Petrinetz und Operandenflussgraph ist die Analyse des Petrinetzes in diesem Falle von begrenztem Wert. Zwar treten die Besonderheiten des Prozessablaufs im Petrinetz augenfälliger zutage, wie beispielsweise die Abwesenheit von Deadlocks, doch sind sie durchaus auch dem Operandenflussgraph zu entnehmen. Nichtsdestoweniger wird in Kap.20.3 bei der Überführung des Operan-

5 Siehe z.B. [Reisig 90]

denflussgraphen von Bild 8.1 in ein Programm ein Petrinetz wertvolle Dienste leisten. Dabei handelt es sich allerdings um ein *steuerbares* Petrinetz (Bild 20.10), das gegenüber dem Petrinetz von Bild 8.4b um *Steuertransitionen* erweitert ist.

Mit Hilfe von Petrinetzen lassen sich Operatorennetze nicht nur hinsichtlich der Gefahr von Blockierungen, sondern auch hinsichtlich anderer Eigenschaften analysieren. Beispielsweise kann untersucht werden, ob die Anzahl der Marken in einem markierten Petrinetz (einem Netz dessen Plätze teilweise mit Marken belegt sind) zunimmt und sich die Marken auf diesem oder jenem Platz ansammeln (Warteschlangenbildung). Es kann auch untersucht werden, ob die Anzahl der Marken mit der Zeit abnimmt, ob der eine oder andere Platz eventuell niemals belegt werden kann oder ob der Markenfluss irgendwann völlig versiegt ("stirbt"). Es kann auch untersucht werden, ob der Markenfluss in einem markierten Netz niemals abbricht. Ist dies der Fall, sagt man, dass das markierte Petrinetz *lebendig* ist.

Im Petrinetz haben wir ein Werkzeug zur kausaldiskreten Beschreibung nebenläufiger Prozesse kennen gelernt, also solcher Prozesse, die parallel in einem Kompositoperator ablaufen können. Dabei haben wir von den Operationen selber abstrahiert und nur die möglichen Reihenfolgen ihrer Ausführung betrachtet. Im folgenden Kapitel werden wir einen in gewissem Sinne "entgegegesetzten" Standpunkt beziehen. Wir werden von der Reihenfolge der Bausteinoperationen abstrahieren, uns aber für die Operationen selber interessieren. Operanden und Operandenplätze, die in Petrinetzen nicht in Erscheinung traten, werden demzufolge eine wesentliche Rolle spielen.

8.2.3 Abstrakter Automat

In Kap. 8.2.1 [3] wurde festgestellt, dass sich die Verhaltensweise von Kompositoperatoren als Funktion darstellen lassen muss, wenn als Operatorenhierarchie konzipierte informationelle Systeme eindeutig arbeiten sollen. Das ist insofern problematisch, als Kompositoperatoren Speicher enthalten, also ein Gedächtnis besitzen, sodass die Abbildung $X \rightarrow Y$ i.Allg. nicht eindeutig ist. Diesem Problem wenden wir uns nun zu.

Speicherplätze sind überall da erforderlich, wo Operanden eventuell warten müssen. Das ist, wie wir gesehen hatten, vor Vereinungen und Sammelweichen der Fall. Ferner kann sich vor jedem Operator eine Warteschlange bilden, wenn seine Operationsdauer größer ist, als die der vorangehenden Operatoren. Das bedeutet, dass Kompositoperatoren, insbesondere steuerbare, in aller Regel Operatoren mit Gedächtnis sind. Damit ergibt sich folgendes Problem. Unser erklärtes Ziel ist es, durch hierarchische Komponierung Systeme (Kompositoperatoren) zu schaffen, die eindeutige Abbildungen *Eingabemenge* $\rightarrow$ *Ausgabemenge* realisieren. Die Eindeutigkeit der Abbildung darf nicht durch die internen Gedächtnisse der Operatoren der Hierarchie zunichte gemacht werden.

Es gibt eine rigorose Methode, das Problem zu lösen. Sie besteht darin, vor jedem Start einer Kompositoperation sämtliche Speicherplätze mit einem ein für allemal

vorgeschriebenen Initialwert zu belegen, z.B. mit 0, d.h. die Speicher zu löschen, und außerdem dafür zu sorgen, dass jeder Speicherplatz während einer Kompositoperation nur ein einziges Mal belegt wird. Unter dieser Bedingung realisiert der Kompositoperator trotz seines Gedächtnisses eine eindeutige Abbildung und kann wie ein Operator ohne Gedächtnis behandelt werden. Diese Methode kommt in der Rechentechnik auf Schritt und Tritt zur Anwendung (siehe Kap.19.5), obwohl sie der Idee der Operatorennetze und damit der USB-Methode widerspricht.

Es gibt eine andere Möglichkeit, sich von dem Gedächtnis zu “befreien”, das in Form von Speichern über das Operatorennetz eines Kompositoperators verteilt ist. Die Befreiung ist zwar nicht vollständig, doch ausreichend, um theoretische Untersuchungen in weit umfangreicherem Maße anstellen zu können, als die USB-Methode sie erlaubt. Zwei Ideen liegen der nun zu behandelnden Methode zur Beschreibung kausaldiskreter Prozesse zugrunde.

Die **erste Idee** besteht darin, die einzelnen Speicherplätze, die das Operatorennetz eines Kompositoperators enthält, gedanklich zu einem einzigen Speicher zusammenzufassen und aus dem Netz herauszuziehen, ohne die einzelnen Verbindungen aufzutrennen. Dann ergibt sich Bild 8.5. Das ursprüngliche Netz möge *n* Bausteinoperatoren mit je einem Ausgabespeicherplatz enthalten. Es reicht aus, nur diese herauszuziehen, denn selbst wenn weitere Speicherplätze in dem Netz existieren, enthalten diese keine zusätzlichen Operanden.

Auch die Ein- und Ausgabeverbindungen der einzelnen Speicherplätze werden gedanklich zusammengefasst, sodass die Eingabeleitung und die Ausgabeleitung des herausgezogenen Speichers aus je *n* Einzelleitungen besteht. In dem Speicher ist in jedem Moment ein Tupel von *n* Operanden gespeichert. Dieses *n*-Tupel heißt **innerer Zustand** des Kompsitoperators und wird mit *z* bezeichnet. Der innere Zustand wird gedanklich über eine Rückkopplung auf den Eingang zurückgeführt, wie in Bild 8.5 gezeigt ist.

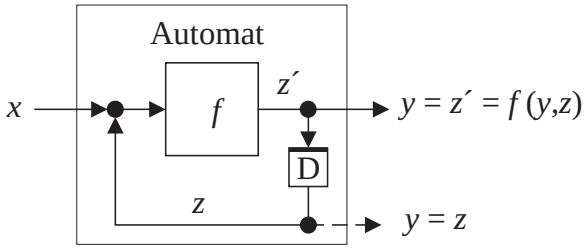

Bild 8.5 Operatorennetz des abstrakten Automaten. *f* - Folgefunktion; D - Taktverzögerer (Delay).

Auf den Eingang des mit *f* bezeichneten Operators wird ein Paar gegeben, das Paar (*x*,*z*), ähnlich wie im Falle eines steuerbaren Operators. Doch tritt an die Stelle des Steuersignals *u* der innere Zustand *z*. Wie aber lässt sich erreichen, dass der *f*-Operator eine eindeutige Abbildung $X \times Z \rightarrow Y$ realisiert? Man bedenke, dass die einzelnen Elemente von *z* zeitlich völlig ungeordnet im Speicher eintreffen, sodass der Begriff des Tupels genau genommen keinen Sinn hat.

Die **zweite Idee** bringt die Lösung des Dilemmas: Sämtliche Bausteinoperatoren arbeiten *synchron*, d.h. sie werden gleichzeitig gestartet. Der nächste Start erfolgt, nachdem alle Bausteinoperationen ausgeführt und die Resultate sich als Komponente von *z* im gemeinsamen Speicher befinden. Eine solche Arbeitsweise heißt **getaktet**. Der Speicher spielt dabei die Rolle eines **Taktverzögerers**. Die gesamte Anordnung heißt **abstrakter Automat**, was darauf hindeutet, dass es sich um eine gedankliche Konstruktion handelt. Im Gegensatz zum Petrinetz arbeitet ein abstrakter Automat von außen betrachtet rein sequenziell; es gibt keine parallelen Prozesse. Der Automat stellt einen *schwarzen Kasten* dar, dessen Innenleben - bis auf den inneren Zustand - unbekannt ist.

Die Abbildung, die ein abstrakter Automat realisiert bzw. die Funktion, die er berechnet, kann auf zweierlei Weise notiert werden:

$$X \times Z \rightarrow Z \quad \text{oder} \quad z' = f(x,z), \tag{8.4}$$

wobei *z* den alten und *z′* den neuen inneren Zustand bezeichnet. Beide Zustände sind Elemente von *Z*. Die Funktion *f* heißt **Folgefunktion**.

Hinsichtlich des Outputs *y* (im Zusammenhang mit Automaten wird i.Allg. von **In**- und **Output** gesprochen) sind verschiedene Vereinbarungen üblich. Für theoretische Untersuchungen wird der Output häufig mit dem alten oder mit dem neuen inneren Zustand identifiziert. Zuweilen wird für die Berechnung von *y* eine spezielle Funktion *g* eingeführt, **Ergebnisfunktion** genannt. In (8.5) sind vier verschiedene Möglichkeiten angegeben, von denen zwei in Bild 8.5 dargestellt sind.

$$\begin{aligned} y &= z \\ y &= z' \\ y &= g(z) \\ y &= g(x,z) \end{aligned} \tag{8.5}$$

Im Allgemeinen wird angenommen, dass die Mengen *X, Y* und *Z* endlich sind. Dann wird der abstrakte Automat auch **endlicher Automat** genannt. Wenn nichts Gegenteiliges gesagt wird, ist im Weiteren unter einem Automaten stets ein endlicher Automat zu verstehen. Die Verhaltensweise eines endlichen Automaten wird durch seine **Automatentafel** angegeben. Als Automatentafel wird die Wertetafel der Folge- und Ergebnisfunktion des Automaten bezeichnet. Jede Zeile der Tafel enthält ein (x,z)-Paar und das zugeordnete (z',y)-Paar.

Es mag befremdlich wirken, die gedankliche Konstruktion von Bild 8.5 als *Automaten* zu bezeichnen, also als etwas, das “sich selbst bewegt”. Man könnte hiermit das ständige Zirkulieren der Operanden in der Rückkopplungsschleife assoziieren, insbesondere dann, wenn es sich um einen **autonomen Automaten** handelt, das heißt um einen Automaten ohne externen Eingang.

Andrerseits erinnert der *Takt*betrieb an *Takt*straßen und damit an Produktions*automaten* oder an die *Takt*frequenz eines Prozessors, was die Vermutung nahe legt, dass Prozessoren als Automaten beschreibbar sind. Wir werden in Kap.13.6 [13.13]

sehen, dass die Vermutung zutrifft. Diese Andeutungen lassen erwarten, dass die beiden Ideen, die dem Automatenbegriff zugrunde liegen, also die Zusammenfassung von Speicherplätzen und der Taktbetrieb, beim Entwurf von Rechnern Pate gestanden haben. Tatsächlich wird der Computerbau ganz entscheidend von diesen beiden Ideen mitbestimmt.

Die besondere Bedeutung des Automatenbegriffs liegt jedoch auf theoretischem Gebiet. Er ist der Grundbegriff einer eigenständigen Theorie, der **Automatentheorie**[6]. In Kap.8.4 wird die Anwendung der Automatentheorie in der Algorithmentheorie und in Kap.16.4 ihre Anwendung in der Theorie formaler Sprachen angedeutet.

Aber auch ohne den Einblick in konkrete Anwendungen erhält man eine Vorstellung von der Breite der Anwendbarkeit des Automatenbegriffs, wenn man sich in ihren Kern vertieft, nämlich in den Umstand, dass die Zukunft irgendeines sehr abstrakt zu verstehenden Objekts mit seiner Vergangenheit verknüpft wird und zwar über seinen ebenso abstrakt zu verstehenden inneren Zustand. Im inneren Zustand, in welchem sich das Objekt im gegenwärtigen Augenblick befindet, ist die gesamte Vergangenheit enthalten, soweit sie sich auf die Zukunft auswirkt, soweit sie die *Ursache* der Zukunft ist.

Diese automatentheoretische Sicht auf Ursache-Wirkungszusammenhänge stellt gewissermaßen eine auf die Spitze getriebene *kausaldiskrete* Prozessbeschreibung dar. Aus dieser Sicht kann das Verhalten jedes Objekts oder besser jedes Trägers von Prozessen betrachtet werden, ja, die ganze Welt lässt sich so betrachten. Ob einem das etwas hilft, ist eine andere Frage.

Abschließend soll diejenige Besonderheit der Prozessbeschreibung mittels Automaten noch einmal hervorgehoben werden, durch die ihre Anwendbarkeit eingeschränkt wird. Die spezielle Art, einen kausaldiskreten Prozess als *Automatenprozess* zu beschreiben, d.h. als Prozess, der in einem Automaten abläuft, zeichnet sich dadurch aus, dass jede räumliche Vorstellung bezüglich des beschriebenen Prozesses (des modellierten Originals) eliminiert ist. Für Prozessbeschreibungen nach der USB- und Petrinetz-Methode trifft das nicht zu. Denn ihnen liegt die Vorstellung eines Flusses durch ein räumliches (zwei- oder dreidimensionales) *Netz* zugrunde.
11 Abstrakter kann man den Unterschied folgendermaßen kennzeichnen. *Die netzorientierte Prozessbeschreibung ist eine* ***raum-zeitliche****, die automatenorientierte eine* ***rein zeitliche*** *Beschreibung.*

8.2.4 Elementare kausaldiskrete Operatoren

In den bisherigen Überlegungen zur kausaldiskreten Prozessbeschreibung ist der Umstand nicht zur Sprache gekommen, dass eine kausaldiskrete Operatorenhierarchie eine *unterste Schicht* haben muss. Dieser Umstand ist so selbstverständlich, dass

6 Siehe z.B. [Stetter 88],.[Sander 92], [Schöning 95]

es überflüssig scheint, ihm besondere Aufmerksamkeit zu widmen. Dass dem nicht so ist, erkennt man, wenn man der Natur der *elementaren* Operatoren und dem Wort "nichtdekomponierbar" auf den Grund geht. Dann stellen sich nämlich sofort zwei Fragen: Wodurch zeichnen sich die elementaren ("nichtdekomponierbaren") Operatoren aus und welche elementaren Ereignisse segmentieren die Zeit?

Da die Operatorenhierarchie, die wir entwerfen wollen, die menschliche Fähigkeit zum Rechnen simulieren soll, liegt es nahe zu fragen, wieweit der Mensch beim Rechnen dekomponiert. Wir wissen, dass er beim Rechnen mit *Zahlen* bis zum *Zählen* dekomponieren kann. Soweit wird sich jedoch kaum jemand "herablassen". Wenn man einen Wert der Funktion (8.1) berechnet und dazu die Bausteinoperationen von Bild 8.1 der Reihe nach per Hand ausführt, wird man z.B. beim Addieren das Dekomponieren höchstens bis zur Addition zweier einstelliger Zahlen fortsetzen. Analog wird man beim Multiplizieren bis zur Multiplikation zweier einstelliger Zahlen dekomponieren, denn das kleine Einmaleins hat man "im Kopf". Beim Rechnen im Kopf oder mit Papier und Stift dekomponiert man bis zu den *interiorisierten* (verinnerlichten, "eingefleischten", gewissermaßen zu "Reflexen" gewordenen) Operationen.

Beim Schreiben eines Computerprogramms muss bis zu denjenigen Operationen dekomponiert werden, die der Computer zur Verfügung stellt. In Kap.15.2 werden wir uns überlegen, wie ein Programm zur Berechnung der Funktion (8.1) aussehen könnte. Die elementaren Operationen, mit denen wir als Programmierer dort arbeiten werden, sind durch den Konstrukteur des Computers festgelegt.

Das führt auf die nächste Frage: Wieweit lassen sich die Hardwareoperatoren von Computern dekomponieren? Diese Frage wird uns in Kap.9.2 beschäftigen. Im Augenblick geht es uns darum, das entscheidende Charakteristikum der elementaren Operatoren herauszufinden. Um dieses zu erkennen, fragen wir zunächst, wieweit sich die Operatoren (Operationen) von Fertigungsprozessen dekomponieren lassen, z.B. die Bausteinoperatoren in Bild 8.3.

Beispielsweise kann das Bohren schrittweise erfolgen. Auch das Fügen kann aus mehreren Schritten bestehen. Wieweit ein Operator (eine Operation) bei Beibehaltung der kausaldiskreten Beschreibung dekomponiert werden kann, hängt von der Konstruktion des Operators, z.B. der betreffenden Werkzeugmaschine, ab. Die Grenze der Dekomponierbarkeit ist durch den Konstrukteur der Maschine festgelegt. Für das Polieren durch den gestrichelt umrahmten Operator in Bild 8.3 ist die Grenze z.B. dadurch festgelegt, dass es in mehreren Schritten ausgeführt und insofern kausaldiskret beschrieben werden kann, dass eine kausaldiskrete Beschreibung jedes einzelnen Schrittes jedoch ihren Sinn verliert.

Will man eine Operation noch genauer beschreiben, will man beispielsweise den 12
Prozess des Spanabhebens beim Bohren oder den Prozess des Abtrennens kleinster Metallpartikel beim Polieren kausal beschreiben, ist man gezwungen, von der kausal*diskreten* zur kausal*kontinuierlichen* Beschreibung überzugehen. Die *Ereignisse* der kausaldiskreten Beschreibung sind Beginn und Ende eines Bearbeitungs-

schrittes. In dem zu Grunde liegenden kausalkontinuierlichen Prozess existieren solche Ereignisse nicht. Damit ist die erste Frage für Werkstückoperatoren beantwortet: Es gibt aus der Sicht der kausaldiskreten Beschreibung elementare Operationen, die durch die Konstruktion der Fertigungsoperatoren festgelegt sind. Es liegt aber jedem kausaldiskreten Prozess stets ein physikalischer, kausalkontinuierlicher Prozess zugrunde.

Das Gleiche gilt für die elektrischen Prozesse in einem Computer. Sie verlaufen aus der Sicht des Physikers kontinuierlich und werden durch Differenzialgleichungen beschrieben, zumindest solange die elektrischen Parameter keine Sprünge machen. Diese können z.B. durch das Öffnen oder Schließen von Schaltern hervorgerufen werden. Tatsächlich werden wir in Kap.10 den Schalter als elementaren Hardwareoperator einführen. Das Umschalten ist aber schon eine spezielle Art von Ereignissen. Das elementare Ereignis im allgemeinsten Sinne ist der Sprung als solcher, die *Unstetigkeitsstelle* im zeitlichen Verlauf einer physikalischen Größe, z.B. einer Spannung. Unstetigkeitsstellen zerstückeln die stetige Beschreibung physikalischer Prozesse und segmentieren damit die Zeit. *Unstetigkeitsstellen erzwingen - aus der Sicht des Physikers - eine* ***stückweise*** *kontinuierliche Prozessbeschreibung; sie ermöglichen - aus der Sicht des Informatikers - eine* ***kausaldiskrete*** *Prozessbeschreibung.*

Es fragt sich, wodurch die Sprünge, die wir für eine kausaldiskrete Beschreibung brauchen, produziert werden, wenn wir das nicht selber tun, z.B. durch Ausschalten der Netzspannung. Wie kann ein kontinuierlicher Prozess *selber* einen Sprung erzeugen? Vor genau diesem Problem steht der Naturwissenschaftler, wenn er Sprünge, welche die Natur macht (Brüche, Kippvorgänge, Katastrophen), "physikalisch", kausalkontinuierlich beschreiben will.

Auch wir sind in Kap.4.1 auf dieses Problem gestoßen, als von Messen und Digitalisieren die Rede war. Wir haben es dort sozusagen ad hoc mit Hilfe einer Einrichtung gelöst, die wir *Schwellenoperator* genannt haben. Ein *Schwellenoperator* ist *definiert* als Operator, der Sprünge auf folgende Weise produziert. Sobald der Eingabewert *x* (vgl. Bild 4.1) einen bestimmten Wert, den sogenannten *Schwellenwert* überschreitet, führt der Ausgabewert *y* einen Sprung aus. Nach den zugrunde liegenden physikalischen Prozessen hatten wir nicht gefragt.

In Kap.4.1 wurden Schwellenoperatoren für den Bau von Analog-digital-Konvertern benötigt, mit deren Hilfe nicht nur das Messen analoger Werte, sondern auch das Ankoppeln eines Digitalrechners an einen Analogrechner (siehe Kap.4.2[4.7]) möglich wird. Jetzt benötigen wir sie zum "*Ankoppeln*" der untersten (kausaldiskret arbeitenden) Schicht einer Operatorenhierarchie an die "darunterliegende" kausalkontinuierliche Schicht, d.h. an die physikalische Realität oder richtiger an die Welt der klassischen Physik.

Die Rückerinnerung an die Funktion des Schwellenoperators beim Ankoppeln diskreter an analoge Prozesse legt die Schlussfolgerung nahe, dass Schwellenoperatoren die elementaren Operatoren informationeller Systeme sind. Dieser Schluss

bedarf jedoch einer Korrektur. Die richtige Antwort auf die eingangs gestellte Frage lautet: *Die **Basisoperatoren** informationeller Systeme sind **Schwellenoperatoren***. 13
Als *Basisoperatoren* sind diejenigen Operatoren bezeichnet, die das Ankoppeln des kausaldiskreten Bereiches an den kausalkontinuierlichen bewerkstelligen. Sie gehören zu beiden Bereichen und bilden gewissermaßen das Niemandsland im Grenzbereich. Man könnte sie auch *Grenzoperatoren* nennen. Sie sind erforderlich, um die Grenze aus dem kontinuierlichen, nichtsprachlichen, physikalischen Bereich in den diskreten, sprachlichen, informationellen Bereich zu überschreiten. Es gilt also die fundamentale Aussage: *Ohne Schwellenoperatoren ist **keine Information** und **keine Informationsverarbeitung** möglich*, zumindest nicht auf dem Boden der klassischen Physik[7].

In Kap.9.2.2 werden wir sehen, dass Schwellenoperatoren nicht nur die *Basis*operatoren der Informationsverarbeitung sind, sondern auch als *elementare* Operatoren für die Komponierung von Kompositoperatoren verwendet werden können. Biologische informationelle Systeme demonstrieren diese Möglichkeit. Denn die Bausteine der grauen Materie des Gehirns, die *Neuronen,* sind offensichtlich Schwellenoperatoren. Dieser experimentelle Befund bestätigt die obige fundamentale Aussage.

8.2.5 Logisch-kausale Äquivalenz

In Kap.8.2.1 [5] waren wir zu der Schlussfolgerung gelangt, dass logischen Zusammenhängen kausale Zusammenhänge zugrunde liegen. Insofern besteht eine Äquivalenz zwischen Logik und Kausalität. Da dies eine *prinzipielle* Feststellung ist, sprechen wir vom **Prinzip der logisch-kausalen Äquivalenz**. Es ist die Folge des Umstandes, dass ein sprachlich (z.B. mathematisch) artikulierter "logischer" Zusammenhang erst dann Sinn erhält, wenn er von einem realen Interpretator interpretiert wird. Das Prinzip verlagert das Problem der Beziehung zwischen Logik und Kausalität in den Interpretator, in den Träger des betreffenden informationellen Prozesses. In Bild 8.6 sind die Beziehungen zwischen logischen und kausalen Zusammenhängen in einer Übersicht wiedergegeben.

Die Zusammenhänge werden auf den drei semantischen Ebenen beschrieben, auf denen sich sprachliches Modellieren vollzieht. Jeder Ebene entspricht ihre charakteristische Art von Semantik, externe bzw. formale bzw. interne Semantik. Auf der externen bzw. internen Schicht bezieht sich Semantik auf die trägerexterne bzw. trägerinterne Realität. Auf der formalen Ebene bezieht sich Semantik auf ein Kalkül (vgl. Bild 5.3). Wir betrachten zunächst die Beziehungen zwischen der externen und der formalen Schicht.

---

7 Für die "Quanteninformatik" gilt die Aussage nicht. Die Diskretisierung erfolgt dort durch die quantenmechanische Natur der Operatoren, der Bausteine des Quantencomputers (siehe z.B. [Briegel 99]). Er wird in diesem Buch nicht betrachtet. Überhaupt kommen quantenmechanische Phänomene nicht zur Sprache.

Semantische Ebene	Bedingung, Ursache	Grundlage des Zusammenhanges	Folge, Wirkung
extern	Zustand der Welt	$\xrightarrow[\text{(im Original)}]{\text{Kausalität}}$	Folgezustand
formal	Satz des Kalküls	$\xrightarrow[\text{(Deduktion durch den Modellträger)}]{\text{Logik}}$	Folgerung
intern	Zustand des Modellträgers	$\xrightarrow[\text{(im Modellträger)}]{\text{Kausalität}}$	Folgezustand

Bild 8.6 Kausale und logische Zusammenhänge auf den drei semantischen Ebenen

Bei der Modellierung eines Ausschnitts der Realität mittels einer formalen Theorie, also eines interpretierten Kalküls, stellt der Realitätsausschnitt eine Interpretation des Kalküls dar. Die Zuordnungen von Bezeichnern des Kalküls zu Objekten und Merkmalen der Realität trifft der modellierende Mensch. Dadurch werden kausale Zusammenhänge der Realität zu logischen Zusammenhängen der Theorie. In der Realität (auf der externen Ebene) wird durch einen realen Prozess ein realer Zustand z in einen *Folge*zustand z' als *kausale Folge* überführt. Auf der mittleren, formalen Ebene wird durch Deduktion ein Satz s des Kalküls in einen Satz s' als *logische Folge* (Schluss*folge*rung) überführt. Wenn z die Interpretation von s ist, muss z' die Interpretation von s' sein, falls die Theorie richtig ist.

Diesen Sachverhalt hat HEINRICH HERTZ im Vorwort zu den "Prinzipien der Mechanik [Hertz 1894] schöner ausgedrückt: "*Wir machen uns innere Scheinbilder oder Symbole der äußeren Welt, und zwar machen wir sie von solcher Art, dass die denknotwendigen Folgen der Bilder stets wieder die Bilder seien von den naturnotwendigen Folgen der abgebildeten Gegenstände*."

Wenn ein Computer die Rolle des Interpretators spielt, sind die Interpretationsprozesse bekannt, sodass die Zurückführung logischer Zusammenhänge, die das System liefert (z.B. durch mathematische Berechnung) auf die zugrunde liegenden kausalen (physikalischen) Zusammenhänge im Computer offen vor Augen liegt. Wenn aber der Mensch die Rolle des Interpretators spielt, ist die Zurückführung nicht möglich, weil die Gehirnprozesse nicht ausreichend bekannt sind.

Das Prinzip der logisch-kausalen Äquivalenz bildet die Grundlage der Rechentechnik und der Herangehensweise dieses Buches an die Probleme der Informatik. Gäbe es keine logisch-kausale Äquivalenz, ließen sich keine logischen Zusammenhänge hardwaremäßig realisieren und hätte das Trägerprinzip [Einleitung.3] keinen Sinn.

Beim Implementieren eines extern interpretierten Kalküls (beim Schreiben entsprechender Computerprogramme) wird dieser ein zweites Mal interpretiert, jedoch nicht extern, nicht durch den zu modellierenden Ausschnitt der Realität, sondern *intern*, indem den Bezeichnern des Kalküls codierende Zustände des Computers zugeordnet werden. Die Zuordnung trifft der Programmierer. Dadurch werden logische Zusammenhänge des Kalküls zu kausalen Zusammenhängen im Computer. Durch die zweifache Interpretation des Kalküls werden verschiedenen Zuständen der externen Realität verschiedene Zustände der trägerinternen Realität zugeordnet und ***externe** kausale Zusammenhänge werden zu **internen** kausalen Zusammenhängen*.

Es stellt sich die Frage, ob die dargelegten Beziehungen zwischen formaler und interner Schicht vielleicht auch dann ihren Sinn behalten, wenn an die Stelle des Computers der Mensch tritt und der Kalkül intern durch Gehirnzustände "interpretiert" wird. Das würde bedeuten, dass Idemen codierende Zustände im Gehirn, also **interne Realeme** entsprechen. Ob das zutrifft, ist gegenwärtig nicht experimentell gesichert. Doch nicht wenige Forscher meinen, dass die Annahme codierender Gehirnzustände zutrifft. Zumindest stellt sie eine vernünftige Hypothese dar. Wenn sie sich als wahr herausstellt, ist in Bild 2.1 zwischen der Realem- und der Idemspalte eine Spalte "Internrealem" einzufügen. Aus einer Zuordnung Realem → Idem wird eine Zuordnung Externrealem → Internrealem → Idem.

Durch diese Erweiterung der "Abbildungen des sprachlichen Modellierens" (des Bildes 2.1) ergeben sich viele neue "objektive" Fragen, d.h. Fragen, die den Bereich der Ideme nicht berühren und eine formale Behandlung erlauben. Lässt man die Welt der Ideme außer Acht, öffnet sich ein Weg zu einer exakten Wissenschaft vom aktiven sprachlichen Modellieren, zu einer exakten, *subsymbolischen* Informatik und zu einer exakten Erkenntnistheorie. Physiologie und Psychologie würden zu einer einheitlichen wissenschaftlichen Disziplin verschmelzen. Trägerexterne und trägerinterne Phänomene würden eine Einheit bilden und gemeinsam (auf der mittleren Ebene) durch eine formale Theorie beschrieben werden. Komponenten *der Welt und der Vorstellung* würden ein gemeinsames System miteinander wechselwirkender Komponenten bilden. All diese Konjunktivsätze artikulieren Möglichkeiten, der aus der Sicht eines ganzheitlich (monistisch) denkenden Menschen eine Selbstverständlichkeiten sind. Am Horizont erscheint die Vision einer naturwissenschaftlichen Untermauerung vieler Bereiche der Geisteswissenschaften.

8.3 Operation, Funktion, Prädikat, Berechenbarkeit

Nach unserem Ausflug an die Grenze zwischen Natur- und Geisteswissenschaften kehren wir nochmals zu den Begriffen Operator, Operation und Funktion zurück. Sie wurden in einigen Fällen wie Synonyme verwendet, was möglicherweise Unverständnis hervorgerufen hat, denn es handelt sich um unterschiedliche Begriffe. Die Unterschiede werden sinnfällig, wenn man, ausgehend vom realen Operator, die Begriffe der Operation und der Abbildung bzw. der Funktion durch zwei Abstraktionsschritte einführt[8].

Gegeben sei ein realer (stofflich realisierter) Operator, z.B. eine Addiermaschine oder die elektronische Realisierung des in Bild 8.1 beschriebenen Operators. Für eine Folge von Eingabewerten liefert der Operator eine Zuordnungstafel, d.h. eine Liste von (x,y)-Paaren, er *berechnet* diese sogenannte **Wertetafel**. Jede Zeile der Tafel ordnet einem x bzw. einem x-Wert genau ein y bzw. einen y-Wert zu. Eine Wertetafel stellt also eine geordnete Menge eindeutiger Zuordnungen dar.

Erste Abstraktion. Wir abstrahieren von der konkreten Realisierung (z.B. von der elektronischen Schaltung) des Operators, nehmen aber an, dass ein realer Operator mit eben der Zuordnungstafel existiert oder gebaut werden kann. Auf diesem Abstraktionsniveau sagen wir, dass die Zuordnungstafel eine **Operation** definiert und nennen die Tafel **Operationstafel** eines *gedachten* Operators.

Man beachte, dass auch der Begriff des *sprachlichen Operators* von dem ausführ-
14 renden (interpretierenden) realen Operator abstrahiert. In diesem Sinne sind die Bezeichnungen “sprachlicher Operator” und “Operation” Synonyme, doch werden sie nicht wie Synonyme verwendet. Wir werden die beiden Wörter vorwiegend in folgenden Bedeutungen benutzen: *Eine **Operation** ist eine durch ein Operationssymbol oder einen Operationscode benannte Zuordnungs**tafel**, ein sprachlicher Operator ist eine Zuordnungs**vorschrift**.* Der Unterschied zwischen beiden Begriffen verschwindet, wenn man ein Operationssymbol als Vorschrift auffasst. Umgangssprachlich wird unter Operation oft der *Vorgang* des Zuordnens verstanden. Wir werden, wenn der Vorgang gemeint ist, der Eindeutigkeit halber von **Operationsausführung** oder von **Prozess** sprechen.

Zweite Abstraktion. Wir abstrahieren nun nicht nur vom realen Operator, sondern von seiner *Existenz* und lassen die Möglichkeit zu, dass *kein* Operator angebbar ist, der die Zuordnung trifft. Dann sprechen wir von **Abbildung** und **Abbildungstafel** oder von **Funktion** und **Funktionstafel**. Jede Zeile einer Funktionstafel ordnet einem *Argumentwert* genau einen *Funktionswert* zu. Als “Werte” sind auch “Wertetupel” zugelassen. (Der Leser hat vielleicht bemerkt, dass die Definition

---

8 Die Definitionen sind an die USB-Methode angepasst und nicht unbedingt identisch mit denjenigen, die in Mathematikbüchern angegeben werden. Aber auch in der Mathematik werden die Begriffe nicht immer ganz einheitlich verwendet.

des Abbildungs- und Funktionsbegriffs das Trägerprinzip verletzt. Darauf kommen wir sogleich zurück.)

Mit der Abstraktion von der stofflichen Realität des Operators wird gleichzeitig von der stofflichen Realität der Operanden abstrahiert. Die Operanden können Objekte des Denkens, also Ideme sein, und der Mensch kann sich in seinem Geiste beliebige Abbildungen zwischen gedachten Objektmengen und er kann sich beliebige Funktionen ausdenken. Der Leser wird ahnen, welche Schwierigkeiten durch diese Abstraktion heraufbeschworen werden. Es lassen sich nämlich Funktionen definieren, die weder vom Computer noch vom Menschen berechnet werden können, denn offensichtlich werden mit der Abstraktion vom Träger *nichtberechenbare* Funktionen zugelassen. Wenn eine Funktion durch Worte festgelegt wird, hinter denen kein Träger, kein Gerät, kein Rechner stehen muss, der die Worte in konkrete Wertepaare (Argumentwert, Funktionswert) umsetzen kann, besteht immer die Gefahr, dass die Worte etwas Unausführbares, etwas nicht Berechenbares, vielleicht sogar etwas Nichtexistierendes oder Unsinniges definieren. Wie wir aus Kap.6 wissen, erlaubt Sprache, Wahres wie Falsches, Sinnvolles wie Sinnloses zu artikulieren.

Dieser Gefahr wirkt das *Trägerprinzip* entgegen. Würden wir uns konsequent daran halten, wäre für den Funktionsbegriff kein Platz in unserem Begriffsgebäude und in unseren Gedankengängen und die Begriffe der Berechenbarkeit und Nichtberechenbarkeit von Funktionen verlören ihren Sinn. Dann würde aber mit dem Funktionsbegriff eine entscheidende Schnittstelle zur Mathematik verloren gehen und die Kommunikation zwischen Informatikern und Mathematikern wäre in Frage gestellt. Das Trägerprinzip darf also nicht zum Dogma erhoben werden, sondern ist eher als unverbindliche Charakterisierung unserer Herangehensweise an die Probleme der Informatik anzusehen.

An dieser Stelle sei auf einen weiteren Begriff hingewiesen, der eine Verletzung des Trägerprinzips bedeutet, auf den wir aber ebenfalls nicht verzichten dürfen, weil die Mathematiker, mit denen wir kommunikabel sein müssen, auf ihn nicht verzichten dürfen, auf den Begriff des Unendlichen. Nach dem Trägerprinzip und dem aus ihm resultierenden Realisierbarkeitsprinzip dürften wir eigentlich nur endliche Wertetafeln zulassen, also nur Funktionen mit endlichen Argument- und Funktionswertemengen. Dann würden wir aber in Konflikt mit dem mathematischen Funktionsbegriff kommen, z.B. mit dem Begriff der Additionsfunktion, denn die Additionsfunktion ist für unendlich viele Summanden definiert, die Wertetafel ist unendlich lang in dem Sinne, dass sie beliebig lange fortsetzbar ist.

Über die Vorstellung der Fortsetzbarkeit führen wir folgenden in der Mathematik üblichen Unendlichkeitsbegriff ein. *Eine Menge, die dadurch entsteht, dass eine endliche Menge unendlich oft um endlich viele Elemente erweitert wird, heißt* ***abzählbar unendliche Menge***. Die Bezeichnung kann in demselben Sinne auf Folgen angewendet werden, z.B. auf Ziffernfolgen, Ereignisfolgen (d.h. auf kausaldiskrete Prozesse) oder auf Wertetafeln. Es ist offensichtlich, dass die Menge (Folge) der natürlichen Zahlen eine abzählbar unendliche Menge (Folge) in dem soeben defi-

nierten Sinne ist, denn man kann "beliebig lange" zählen, vorausgesetzt, man lebt beliebig lange. Das Unendliche ist immer nur *gedanklich*, d.h. "abstrakt realisierbar". Falls eine Menge abzählbar unendlich sein kann, aber nicht sein muss, sprechen wir von **unbeschränkter Menge**[9].

Wir verletzen also das Realisierbarkeitsprinzip und lassen abzählbar unendliche Funktionstafeln zu. Nicht nur die Anzahl, sondern auch die Länge der Zeilen darf abzählbar unendlich sein. Damit haben wir uns dem mathematischen Funktionsbegriff angepasst und der Begriff der Berechenbarkeit bekommt konkreten Sinn und muss definiert werden. Um den Träger eines Berechnungsprozesses nicht aus dem Auge zu verlieren unterscheiden wir zwischen der *Berechnung* und der *Realisierung* einer Funktion und vereinbaren:

*Eine Funktion heißt **realisierbar** bzw. **realisiert**, wenn ein realer Operator gebaut werden kann bzw. existiert, der jedem Argumentwert den Funktionswert zuordnen kann. Eine Funktion heißt **berechenbar**, wenn ein sprachlicher Operator (eine Operationsvorschrift, ein Algorithmus, eine Formel) und ein realer Operator existiert oder angebbar ist, der durch Interpretierung des sprachlichen Operators (durch Ausführung der Vorschrift) zu jedem Argumentwert den Funktionswert bestimmen kann.*

Danach ist *Berechenbarkeit* ein Spezialfall von *Realisierbarkeit*. Der Begriff der Realisierbarkeit setzt genau das voraus, wovon der Funktionsbegriff abstrahiert: den realen Operator bzw. Interpretator. Wie bereits festgestellt, ist es aus diesem Grunde nicht verwunderlich, dass es nichtberechenbare Funktionen gibt, genauer gesagt Funktionen, die sich zwar definieren, aber nicht berechnen lassen.

Man kann die Frage aufwerfen, unter welchen Bedingungen eine Funktion definiert, aber nicht realisiert bzw. berechnet werden kann. Im Rahmen der Algorithmentheorie wird diese Frage gestellt und versucht, sie zu beantworten. Auch wir werden uns in Kap.8.4 mit dem Problem der Berechenbarkeit beschäftigen, aber lediglich mit dem Ziel, einige Gedankengänge, Bergriffsbildungen und Schlussfolgerungen der Theorie der Algorithmen und der Berechenbarkeit zu verstehen. Das wird uns helfen, die USB-Methode mit dem wichtigen Begriff der *rekursiven Funktion* in Beziehung zu setzen. Außerdem wird es uns helfen, die Problemstellungen solcher Teilgebiete der Informatik wie Programmierungstechnik und Künstliche Intelligenz, die stark von der Algorithmentheorie mitgeprägt worden sind, deutlicher zu erkennen und auch die Wege, auf denen bestimmte Problemlösungen gefunden worden sind.

Es sei noch einmal wiederholt: *Der Funktionsbegriff ohne Bindung an einen realen Operator ist ebenso wie der Berechenbarkeitsbegriff für den Bau und die Nutzung von Computern nicht erforderlich.* Im Sinne des Realisierbarkeitsprinzips

9 In der Mathematik wird der Begriff der unbeschränkten Menge auch in einer anderen, spezielleren Bedeutung verwendet.

werden wir, wenn von einer Funktion die Rede und nichts Gegenteiliges gesagt ist, stillschweigend voraussetzen, dass sie realisierbar ist, d.h. *wir **identifizieren** den Begriff der Funktion mit dem der Operation und damit auch mit dem des sprachlichen Operators*, denn wir hatten oben [4] festgestellt, dass die Bezeichnungen "Operation" und "sprachlicher Operator" de facto Synonyme sind. Unter Voraussetzung der 16
Realisierbarkeit sind wir also berechtigt, *die Bezeichnungen **sprachlicher Operator**, **Operation** und **Funktion** als **Synonyme** zu verwenden*. Im Sinne der Trägerprinzips fragen wir nicht, ob eine Funktion berechnet werden kann, sondern wir fragen, welche Input-Output-Zuordnungen ein bestimmtes (existierendes, oder eindeutig beschriebenes) informationelles System treffen und welche Wertetafeln (sprich Funktionen) das System realisieren bzw. berechnen kann. Das hat zur Folge, dass viele interessante und schwierige Fragen der Algorithmentheorie in diesem Buch nicht zur Sprache kommen.

Der Begriff der nichtberechenbaren Funktion wirft die Frage auf, wie sich eine Funktion definieren lässt, wenn sie nicht berechnet werden kann. Es wird sich zeigen, dass dies möglich ist, indem man sie durch ein *Prädikat* definiert (der Prädikatbegriff wird weiter unten eingeführt). Ganz allgemein gibt es folgende Möglichkeiten für die Festlegung von Funktionen:

1. Festlegung durch eine Wertetafel,
2. Festlegung durch einen realen Operator,
3. Festlegung durch einen sprachlichen Operator,
4. Festlegung durch ein Prädikat.

Es ist Aufgabe der *technischen* Informatik, Techniken zu entwickeln, die den Entwurf, die Realisierung und die Nutzung der vier genannten Möglichkeiten der Funktionsfestlegung unterstützen. Es bietet sich an, den zu behandelnden Stoff in diesem Sinne zu gliedern. Das spiegelt sich in der weiteren Kapitelfolge wider. Die folgenden Bemerkungen sollen den Leser darauf vorbereiten.

Zur Festlegung durch Wertetafeln. Jedes Schulkind kennt die Tafeln des Kleinen Einmaleins. Mathematische Tafelwerke sind Sammlungen von Funktionstafeln wie z.B. Logarithmentafeln, Tafeln der trigonometrischen Funktionen u.ä.m. Durch die Stellenzahl der Wertangaben ist die Genauigkeit festgelegt. Im allgemeinen Falle einer Abbildung kann die Wertetafel beliebige Wörter enthalten, z.B. die Namen und Adressen der Arbeitnehmer einer Firma. Das Faktenwissen von *Datenbanken* ist in der Regel in Form derartiger Tafeln strukturiert[16.5]. Auch ein Telefonbuch, das jedem Abonnenten eine Telefonnummer zuordnet, stellt eine Funktionstafel dar. Wenn ein Abonnent zwei Nummern hat, besteht der betreffende Funktionswert aus zwei Nummern.

Zur Festlegung durch reale Operatoren. In Kap.9.3 werden wir uns überlegen, wie man zu einer bestimmten Wertetafel einen realen Operator konstruieren kann, der die Zuordnungen realisiert. Es wird sich zeigen, dass dies immer möglich ist, vorausgesetzt die Wertetafel liegt vollständig vor, d.h. sie enthält sämtliche x,y-Paare.

Das mag zunächst überraschen. Denn wie lässt sich z.B. eine Funktion realisieren, deren Wertetafel zufällig, z.B. durch Würfeln oder mit Hilfe eines Zufallszahlengenerators erstellt worden ist. Der Prozess lässt sich nicht wiederholen. Doch nachdem das Würfeln beendet ist, liegt eine vollständige Wertetafel der erwürfelten Funktion vor.

Zur Festlegung durch sprachliche Operatoren. Diesem Problem ist das Kapitel 8.4 gewidmet. Wir werden nach Methoden für das Formulieren von Operationsvorschriften suchen und fragen, ob es eine *universelle* Methode (ein Rezept, einen Algorithmus) gibt, nach der sich Vorschriften zur Berechnung *aller* realisierbaren Funktionen formulieren lassen, d.h. aller Funktionen, die sich durch *reale* Operatoren berechnen lassen.

17 **Zur Festlegung durch Prädikate.** Auf diese Art der Festlegung soll schon an dieser Stelle etwas ausführlicher eingegangen werden. Um zu verstehen, worin sie besteht, muss zuerst der Prädikatbegriff erklärt werden. Wir werden uns nicht damit begnügen, ihn zu definieren, sondern wir werden ihn ausführlich erläutern. Viele der folgenden Überlegungen berühren tiefreichende theoretische Fragen und müssen nicht unbedingt angestellt werden, wenn es "nur" darum geht, den Computer nachzuerfinden. Trotzdem wird der Prädikatbegriff auch für uns eine wichtige Rolle spielen.

Wir beginnen mit einer vorläufigen Bestimmung des Prädikatbegriffs mit Hilfe
18 umgangssprachlicher Begriffe: *Ein Prädikat ist ein sprachlicher Ausdruck mit der Struktur eines Aussagesatzes*, also eines Satzes, der seinem Subjekt ein *Eigenschaftsmerkmal* zuweist; das Merkmal kann auch eine Tätigkeit sein. Im Unterschied zur Satzlehre stellt ein Prädikat nach obiger Definition nicht einen Satzteil dar, sondern hat die Bedeutung eines vollständigen Satzes.

Das Eigenschaftsmerkmal kann auch eine Beziehung zu anderen "Subjekten" darstellen, die dann aber nicht zum Satzsubjekt gehören müssen, sondern auch Objekt oder Teil des Satzprädikats sein können. Darin liegt eine gewisse Inkonsequenz der Bezeichnungsweise, die wir dadurch ausmerzen, dass wir das Wort *Subjekt* durch das Wort *Objekt* (nicht im grammatikalischen, sondern im umgangssprachlichen Sinne) ersetzen und definieren:

19 *Einen sprachlichen Ausdruck, der ein Merkmal eines Objekts oder eine Relation zwischen Objekten artikuliert, nennen wir* ***Prädikat***. (Die Ersetzung des Wortes *Beziehung* durch *Relation* stellet eine Annäherung an den Sprachgebrauch der Mathematik dar.) Die Definition verlangt nicht, dass es sich um ganz bestimmte (konkrete) Objekte handelt; sie können unbestimmt bleiben. Unbestimmte Objekte werden **Individuenvariable** oder einfach *Variable* genannt; eindeutig bestimmte ("konkrete") Objekte werden *Konstante* genannt. Zur Illustration folgen sieben Beispielprädikate P_1 bis P_7. Um zum Ausdruck zu bringen, dass eine Zeichenkette als Prädikat zu interpretieren ist, setzen wir sie in eckige Klammern.

P_1 [Max ist 5 Jahre alt]
P_2 [3 = 5]
P_3 [z = 5]
P_4 [*Nr* ist die Telefonnummer des Abonnenten *Ab*]
P_5 [*k* ist das jüngste Kind von *v* und *m*]
P_6 [$y < x$]
P_7 [$y = x$]

Die Beispiele verdeutlichen die Tragweite des Prädikatbegriffs. Der Leser wird sie ohne Kommentar richtig verstehen, trotz der unterschiedlichen Gegenstandsbereiche und Notationsweisen. Unbestimmte Objekte sind durch kursive Buchstaben bezeichnet, um sie als Variable auszuweisen.

Als Symbol für ein Prädikat wird oft der Buchstabe *P* verwendet. $P(x_1,x_2,...x_n)$ bezeichnet ein Prädikat mit *n* Variablen. Es wird n-stelliges Prädikat genannt (in Analogie zur n-stelligen Funktion). Beispielsweise ist P_3 einstellig, P_4 ist 2-stellig und P_5 ist 3-stellig. Prädikate, die gar keine Variablen enthalten (0-stellige Prädikate), wie z.B. P_1 und P_2, heißen **Aussagen** (im Sinne der Prädikatenlogik). Man beachte, dass P_2, P_3, P_6 und P_7 rein formale Semantik besitzen, während sich P_1, P_4 und P_5 externsemantisch interpretieren lassen.

Mancher Leser wird vielleicht über das zweite Prädikat verwundert sein, weil es eine falsche Aussage ist. Formal ist P_2 jedoch richtig, denn es entspricht der Definition des Prädikats. Diese verlangt nämlich *nicht*, dass die artikulierte Eigenschaft bzw. Relation zutreffen muss, die Relation wird nicht konstatiert, sondern lediglich artikuliert. Wenn sie zutrifft, sagt man, dass das Prädikat **erfüllt** ist, andernfalls, dass es **nicht erfüllt** ist. Oft wird auch gesagt, dass ein Prädikat *wahr* bzw. *falsch* ist. Auch wir werden diese Redeweise verwenden, obwohl sie nicht ganz korrekt ist. Um korrekt zu sein, müsste man sagen, dass ein Prädikat für bestimmte Werte der Variablen zu einer *wahren* bzw. *falschen Aussage* wird. Eine analoge Unkorrektheit lässt man sich zu Schulden kommen, wenn man sagt: “Die Funktion *f*(*x*) nimmt den Wert 1 an”, ohne anzugeben für welche *x*-Werte. Ein Prädikat heißt **entscheidbar**, wenn für beliebige Variablenwerte entschieden werden kann, ob es erfüllt ist oder nicht.

Es ist wichtig, die Bedeutung der Gleichheitszeichen in einem Prädikat richtig zu verstehen. Wie bereits festgestellt wurde, konstatiert P_2 nicht die Gleichheit von 3 und 5. Das Entsprechende gilt für P_1 und P_3. P_3 ist *nicht* so zu verstehen, dass der Variablen *z* der Wert 5 *zugewiesen* wird, dass gewissermaßen *z* zur Konstanten 5 “gemacht” wird, genauso wie P_1 nicht aus Max einen Fünfjährigen “macht”. Das Gleichheitszeichen in P_2 und P_3 und überhaupt in jedem Prädikat artikuliert also keine *Wertzuweisung* sondern eine Beziehung, eine *Relation*, die Relation der Gleichheit. Darum sprechen wir von **relationaler Gleichung**. Wenn dagegen ein Ausdruck mit einem Gleichheitszeichen eine Wertzuweisung artikuliert, sprechen wir von **Ergibtgleichung**. Der Eindeutigkeit halber ist es zweckmäßig, das Gleichheitszeichen in einer Ergibtgleichung durch das **Ergibtzeichen** “:=” zu ersetzen.

In der mathematischen Literatur wird der Gleichungsbegriff in der Regel ausschließlich in relationalem Sinne definiert. Dann stellt die Bezeichnung "Ergibtgleichung" einen Widerspruch und die Bezeichnung "relationale Gleichung" einen Pleonasmus dar. Abweichend vom mathematischen Sprachgebrauch vereinbaren wir (in Konzession an die weit verbreitete doppeldeutige Verwendung des Wortes "Gleichung"): *Eine **relationale Gleichung** ist ein mathematischer Ausdruck, der das **Relationszeichen** "=" enthält. Aus einer relationalen Gleichung wird eine **Ergibtgleichung**, wenn sie nach einer Variablen aufgelöst und das Gleichheitszeichen durch das Ergibtzeichen ersetzt wird, wodurch zum Ausdruck gebracht wird, dass der Ausdruck als Vorschrift zur Berechnung des Wertes der eliminierten Variablen zu interpretieren ist.* Eine computerverständliche Ergibtgleichung heißt **Ergibtanweisung**. Es sei noch einmal wiederholt, dass das Gleichheitszeichen in einem Prädikat keine Gleichheit konstatiert oder fordert, sondern eine (unverbindliche) Aussage ist, die wahr oder falsch sein kann.

Wie man sieht, verbergen sich hinter dem Gleichheitszeichen logische Schwierigkeiten, die man vielleicht nicht erwartet hatte. Etwas einfacher liegen die Dinge im Falle von Ungleichungen. Denn es versteht sich von selbst, dass ein Symbol, das keine Gleichheit artikuliert (z.B. das Symbole ">"), kein Ergibtzeichen, sondern nur ein Relationszeichen sein kann und dass eine Ungleichung niemals eine Wertzuweisung ist, sondern stets eine Beziehung artikuliert.

Wir machen nun einen großen Sprung zurück zu Kapitel 7.2 und erinnern uns an den ABC-Schützen [7.12], der die Differenz 5-2 dadurch bestimmt, dass er 2 sooft inkrementiert (um 1 erhöht), bis er 5 erreicht. Er zählt die Inkrementierungsschritte und entscheidet nach jeder Inkrementierung das Prädikat [z=5], worin z das momentane inkrementierte Ergebnis ist. Er beendet das Inkrementieren, sobald das Prädikat erfüllt ist. Die Anzahl der Inkrementierungen ist die gesuchte Differenz.

Ganz ähnlich verfährt der Steueroperator in Bild 8.1b. Während der Berechnung der Potenz x^n durch iteratives Multiplizieren zählt er die Iterationsschritte und entscheidet in jedem Schritt das Prädikat [it<n]. Darin ist it das momentane Zählergebnis, die sog. Iterationszahl, und n ist die vorgegebene Potenz. Sobald das Prädikat falsch wird, generiert der Steueroperator ein Steuersignal, das die Zweigeweiche in der Ausgabeleitung des Multiplizierers umstellt, sodass sie nicht mehr auf den Eingang des Multiplizierers, sondern auf den des Addierers zeigt.

Der ABC-Schütze wie der Potenzierer berechnen beide den Wert einer 2-stelligen Funktion, einmal der Differenzfunktion Minuend minus Subtrahend, das andere mal der Potenzfunktion Basis hoch Exponent. In beiden Fällen ist das Entscheiden eines Prädikats in den Berechnungsprozess involviert und damit auch in die Berechnungsvorschrift (den sprachlichen Operator), der die Funktion festlegt. Wir haben es also mit zwei Beispielen dafür zu tun, dass eine Funktion unter Einbeziehung eines Prädikats festgelegt wird, wobei das Prädikat die Rolle eines Endekriteriums spielt. Ein solches Prädikat nennen wir **Ende-** oder **Abbruchprädikat**.

Die Frage liegt nahe, ob die Rolle von Prädikaten darauf beschränkt ist, ein mehr oder weniger wichtiger *Bestandteil* einer Berechnungsvorschrift zu sein, wie in obigen Beispielen, oder ob ein Prädikat selber die Berechnungsvorschrift darstellen kann, m.a.W. ob ein Prädikat allein eine Funktion festlegen kann. Betrachtet man unter diesem Gesichtspunkt das Prädikat P7, erkennt man, dass diejenigen (x,y)-Wertepaare, für die das Prädikat erfüllt ist, die Wertetafel der Funktion $y = f(x) = x$ bilden. Das Prädikat P7 legt also diese Funktion fest, wenn es als erfüllt vorausgesetzt wird. Um einem Argumentwert seinen Funktionswert zuzuordnen, muss die Ergibtanweisung $y{:=}x$ ausgeführt werden. (Am Rande sei angemerkt, dass der Prädikatbegriff in der Mathematik oft als "*Wahrheitsfunktion*" *definiert* wird.)

Bringt man in der Gleichung $y{=}x$ beide Variable auf die linke Seite, erhält man x-y = 0. Auf der linken Seite dieser Gleichung steht nun eine Funktion $f(x,y)$, die für alle Argumentwertepaare, für die P7 erfüllt ist, den Wert 0 besitzt. Folglich kann man das Prädikat P7 dadurch *entscheiden*, dass man die Nullstellen der Funktion $f(x,y)$ sucht. Das *Entscheiden* eines Prädikats wird in das *Berechnen* einer Funktion überführt.

Das Vorgehen scheint in diesem einfachen Fall nicht viel Sinn zu haben, doch lässt es sich verallgemeinern. Jedes Prädikat $P(x_1, x_2,...)$, das ein Relationszeichen enthält, lässt sich in die Form $f(x_1, x_2,...)$ # 0 überführen, worin # jedes Relationszeichen sein kann. Die Funktion auf der linken Seite heißt **charakteristische Funktion**. Ihr Wert zeigt an, ob das Prädikat erfüllt ist oder nicht. Folglich gilt: *Ein Prädikat ist entscheidbar, wenn seine charakteristische Funktion berechenbar ist*.

Anhand des Prädikats P7 wurde gezeigt, wie durch ein zweistelliges Prädikat eine Funktion festgelegt werden kann. Es soll nun gezeigt werden, dass auch durch ein einstelliges Prädikat eine Funktion festgelegt werden kann. Gegeben sei ein Prädikat $P(x)$. Dann kann man eine Funktion $f(x)$ dadurch festlegen, dass man ihr für alle x, für die P nicht erfüllt ist, den Wert 0 zuweist und für alle x, für die P erfüllt ist, den Wert 1. Auf diese Weise legt $P(x)$ eine Funktion $f(x)$ fest, die zwei Funktionswerte annehmen kann, also eine *binäre* Funktion ist.

Anhand eines Beispiels soll demonstriert werden, dass man auf diese Weise nichtberechenbare Funktionen definieren kann. Wir legen eine binäre Funktion $f(x)$ fest, die den Wert 1 annimmt, wenn das Prädikat

[in der Irrationalzahl x tritt die Ziffernfolge 0 bis 9 auf]

erfüllt ist. Beispielweise gilt $f(\sqrt{2}) = 1$, wenn bei der stellenweisen Berechnung der Wurzel aus 2 nach einem iterativen Algorithmus *irgendwann einmal* die Ziffern von 0 bis 9 in geordneter Reihenfolge auftreten. Diese Frage kann niemals mit "*nein*" beantwortet werden, denn "man kann nie wissen, was noch kommt". Man weiß nicht, ob die Ausführung des Algorithmus tatsächlich ein Ende findet, ob der Algorithmus *terminiert*. Diese Unsicherheit besteht immer dann, wenn gefragt wird, ob ein Ereignis, das nicht mit Sicherheit eintritt, tatsächlich eintritt. Die Frage kann nie verneint werden. Erst nachdem das Ereignis eingetreten ist, kann sie bejaht werden.

21 Prädikate, die erst dann erfüllt sind (“wahr” sind, “Wahres sagen”), sobald ein *nichtvorhersehbares* Ereignis eingetreten ist, vorher aber nicht erfüllt sind, nennen wir **Wahrsageprädikate** (eine unübliche, aber sinnfällige Bezeichnung).

Die vorangehenden Überlegungen führen auf das allgemeine Problem der *Terminierung* von Programmen, also auf die Frage, ob die Abarbeitung eines Programms früher oder später ihr Ende findet. Wir werden uns dem Problem nähern, indem wir die Frage untersuchen, welche Rolle Prädikate in der USB-Methode spielen und welche Bedingungen sie erfüllen müssen. Zur Beantwortung der Fragen führen wir den Begriff des Steuerprädikats ein: *Ein Prädikat, dessen Wahrheitswert eine Weiche oder ein Tor steuert, heißt* ***Steuerprädikat****.* Das Steuerprädikat der Weiche vor dem Sinusoperator in dem Operatorennetz von Bild 8.1a lautet beispielsweise [$x \leq 0$].

Damit ist der erste Teil obiger Frage beantwortet: Prädikate spielen im Rahmen der USB-Methode die Rolle von Steuerprädikaten. Der zweite Teil der Frage lautet damit: Welche Bedingungen müssen Steuerprädikate erfüllen? Zunächst ist festzustellen, dass Steuerprädikate vom Steueroperator entschieden werden, der die Steuersignale generiert. Wenn ein Kompositoperator eine bestimmte Operation ausführen soll, müssen alle Prädikate, die während der Operationsausführung zu entscheiden sind, entscheidbar sein. Um zu sichern, dass die Operationsausführung terminiert (ihr Ende erreicht), muss zusätzlich gefordert werden, dass Abbruchprädikate irgendwann einmal denjenigen Wahrheitswert annehmen, bei dem abgebrochen werden soll; m.a.W. ein Abbruchkriterium muss früher oder später erfüllt sein.

Wir fassen die Ergebnisse zusammen: *Im Rahmen der* ***USB-Methode*** *spielen Prädikate die Rolle von* ***Steuerprädikaten****, wozu auch die* ***Abbruchprädikate*** *gehören. Steuerprädikate müssen entscheidbar sein, und nach Beginn einer Operationsausführung muss jedes Abbruchprädikat in endlicher Zeit das Abbruchkriterium erfüllen.*

Als Steueroperator kann der Mensch fungieren. Andernfalls müssen technische Steueroperatoren komponiert werden. Unter ihren Bausteinoperatoren müssen solche sein, die Prädikate entscheiden. Wir nennen sie **Prädikatoperatoren**. Im Rahmen der USB-Methode werden Prädikatoperatoren und Steueroperatoren aus den gleichen elementaren Operatoren komponiert wie die Arbeitsoperatoren.

Aus der Rolle, die Prädikate in der USB-Methode spielen, ergibt sich folgender Tatbestand. *Der Grund dafür, dass ein Operatorennetz für einen Eingabewert des zugelassenen Typs* (beispielsweise für eine reelle Zahl oder ein Tupel reeller Zahlen) *keinen Ausgabewert liefert, kann nur in einer Rückkopplungsschleife liegen, deren Abbruchkriterium fehlt oder nie erfüllt wird,* vorausgesetzt, dass alle Steuerprädikate entscheidbar sind und dass keine Fehler der Struktur und Funktion des Netzes vorliegen, Deadlocks eingeschlossen. Beispielsweise hält die Berechnung des Quotienten 1:3 als Dezimalzahl niemals an, wenn die Vorschrift keine Angabe darüber enthält, auf wie viele Stellen genau der Quotient berechnet werden soll.

Das Gleiche gilt für die Berechnung jeder Irrationalzahl. Das Problem kann nicht dadurch aus der Welt geschafft werden, dass Irrationalzahlen deswegen verboten

werden, weil sie dem Realisierbarkeitsprinzip widersprechen. Ihre Benutzung in sprachlichen Modellen ist notwendig, wenn sie in der Wirklichkeit existieren, m.a.W. wenn sie durch die Wirklichkeit *definiert* werden. Das gilt z.B. für das Verhältnis der Länge des Umfanges zur Länge des Durchmessers eines Kreises (π) oder für das Verhältnis der Länge einer Diagonalen zur Länge einer Seite eines Quadrates ($\sqrt{2}$). Die Zahlenwerte dieser real existierenden Verhältnisse sind nicht exakt, d.h. nicht absolut genau berechenbar. Sie sind aber auch nicht absolut genau *messbar* (vgl. Kap.4.1 [4.3]). Damit muss man sich abfinden. *Nicht jede durch ein Prädikat definierbare Zahl ist exakt berechenbar*. Freilich kann die Ungenauigkeit beliebig herabgesetzt werden, wenn der Berechnungsprozess gedanklich nicht beschränkt wird, sondern als abzählbar unendliche Folge von Ereignissen angenommen wird.

Eine Berechnungsvorschrift, deren Abarbeitung nicht endet, die nicht terminiert, stellt keinen Algorithmus dar, denn ein Algorithmus terminiert definitionsgemäß. Erst durch Hinzufügen eines Abbruchkriteriums wird eine nichtterminierende Vorschrift zu einem Algorithmus. Im Sinne des Realisierbarkeitsprinzips verlangen wir, dass eine Berechnungsvorschrift stets ein Abbruch- oder Endekriterium explizit oder implizit enthält und sei es in Form einer Begrenzung der Rechenzeit.

In der Praxis ist der häufigste Grund dafür, dass ein Programm nicht terminiert, ein versehentlich falsch formuliertes Abbruchkriterium. Um das Programm "zum Laufen zu bringen", muss der Fehler gefunden und beseitigt werden. Das kann recht zeitaufwendig sein. Es wäre wünschenswert *vor* dem Start eines Programms zu wissen, ob es terminiert oder nicht. Leider gibt es keinen Algorithmus (kein Programm), nach dem man für jedes Programm feststellen kann, ob es für bestimmte Eingabedaten terminiert. Dieser Tatbestand wird als **Halteproblem** bezeichnet. 22
Damit kommen wir zu einem zweiten Beispiel einer binären Funktion, die durch ein Prädikat festgelegt ist. Es handelt sich um die sog. *Haltefunktion.*

Nach bewährtem Vorbild kann man mit Hilfe des Prädikats

[das Programm *p* terminiert für den Eingabeoperanden *x*]

eine binäre Funktion $h(p,x)$ definieren, wobei x ein Tupel sein kann. Die Funktion h nehme den Wert 1 an, wenn p für x terminiert, andernfalls den Wert 0. Die so festgelegte Funktion heißt **Haltefunktion**. Sie ist nicht berechenbar, d.h. nicht für jedes (p,x) ist der Wert von h berechenbar. Den exakten Beweis findet der Leser in fast jedem Lehrbuch über theoretische Informatik[10]. Wir führen zwei Argumente für die Unberechenbarkeit der Haltefunktion an.

Das Programm, das die Haltefunktion für ein Paar (p, x) berechnen soll - wir nennen es *Analyseprogramm* - muss feststellen, ob während der Ausführung von p jedes der in ihm enthaltenen Abbruchprädikate irgendwann (in endlicher Zeit) das Abbruchkriterium erfüllt. Das ist in vielen Fällen möglich, selbst wenn das analy-

10 Genannte seien [Stetter 88], [Sander 92], [Schöning 95].

sierte Programm *nicht* terminiert. Wenn beispielsweise ein Programm durch mehrmaliges Multiplizieren die fünfte Potenz von 3 berechnen soll, könnte das Abbruchprädikat lauten: Breche ab, sobald das Prädikat [z=5] erfüllt ist. Wenn das Potenzierprogramm nach jeder Multiplikation z inkrementiert, beginnend mit z=0, terminiert das Programm, und zwar nach 5 Multiplikationen (die erste Multiplikation ist 1∗3). Wenn das Programm das Inkrementieren - aus welchen Gründen auch immer - mit 6 beginnt oder wenn es mit 0 beginnt, aber nicht inkrementiert, sondern dekrementiert, terminiert das Programm nicht. Ob das Programm terminiert oder nicht, lässt sich feststellen, ohne es ausführen zu lassen.

In derartigen Fällen ist es also durchaus möglich, ein Analyseprogramm zu schreiben, das erkennt, ob das Potenzierprogramm terminiert oder nicht, das also den Wert der Haltefunktion berechnet. Es ist aber *nicht* möglich, ein *universelles* Analyseprogramm zu schreiben, und zwar aus folgendem Grunde.

Ein universelles Analyseprogramm muss für jedes zu analysierende Programm terminieren. Um sich zu überzeugen, dass ein Analyseprogramm diese Forderung erfüllt, benötigt man ein "übergeordnetes" Analyseprogramm, das seinerseits terminieren muss und so fort. Es könnte eingewendet werden, dass ein universelles Analyseprogramm, wenn es denn existiert, sich auch selber analysieren kann, sodass die Notwendigkeit einer unendlichen Kette von Analyseprogrammen entfällt und durch einen Zyklus ersetzt wird. Es liegt eine Operand-Operator-Zirkularität vor, und zwar eine Zirkularität ohne Ausweg, ohne Ende (vgl.Kap.6.3; man erinnere sich an die Schlange, die sich vom Schwanz her auffrisst). Das bedeutet, dass das Analyseprogramm nicht terminiert, also nicht universell ist und dass folglich die Haltefunktion nicht berechenbar ist. Diese Schlussfolgerung stellt zwar keinen mathematischen Beweis, aber doch ein plausibles Argument für die Nichtberechenbarkeit der Haltefunktion dar.

Man kann ein anderes Argument für die Nichtberechenbarkeit der Haltefunktion anführen. Es ist durchaus möglich, dass ein Programm als Abbruchprädikat ein Wahrsageprädikat enthält. Es macht beispielsweise keine Schwierigkeiten, ein Programm p zur Beantwortung der Frage zu schreiben, ob der als Dezimalzahl dargestellte Wert der Wurzel aus 2 die Zahlenfolge von 0 bis 9 enthält. Kein Analyseprogramm kann vorhersagen, ob das Programm terminiert. Ob das der Fall ist, weiß man erst, nachdem es terminiert hat, nachdem es die gesuchte Ziffernfolge gefunden hat.

In der Mathematik spielt eine andere Art von Nichtberechenbarkeit eine wichtige Rolle, die ihren Grund nicht darin hat, dass eine Operationsvorschrift nicht terminiert, sondern darin, dass keine Vorschrift angebbar ist. Das ist z.B. der Fall, wenn eine Funktion durch eine Relationsgleichung festgelegt wird, die nicht lösbar ist, also nicht nach der Funktionsvariablen aufgelöst werden kann. Dann ist keine Ergibtgleichung angebbar, nach der sich die Funktionswerte aus den Argumentwerten errechnen ließen. Ein Beispiel sind algebraische Gleichungen höherer als vierter Ordnung. Für sie existiert keine analytische Lösung in Form einer Ergibtgleichung (keine geschlos-

sene Lösung in Radikalen). Die Funktionswerte lassen sich allerdings numerisch durch eine Näherungsrechnung bestimmen.

Wir werfen nun noch einmal einen Blick zurück auf die vier Methoden für das Festlegen von Funktionen und fragen, nach welchen Methoden nichtberechenbare Funktionen definiert werden können. Die Antwort lautet: *Die Funktionsfestlegung durch Prädikate oder mit Hilfe von Prädikaten ist die einzige Methode, nach der nichtberechenbare Funktionen definierbar sind*. Die übrigen drei oben aufgezählten Methoden setzen die Berechenbarkeit bzw. Realisierbarkeit der Funktion voraus.

Wir beenden unsere Überlegungen mit folgender Bemerkung. Der Begriff der Berechenbarkeit ist als eine Eigenschaft mit zwei Werten definiert worden: berechenbar oder nichtberechenbar. Das Merkmal "*schwer* berechenbar" oder "*mit hohem Aufwand* berechenbar" wurde nicht eingeführt. Aber gerade der erforderliche Aufwand ist in der Praxis oft von Interesse. Der Aufwand kann eine Funktion *praktisch unberechenbar* machen. Das Aufwandsproblem ist nicht Gegenstand der *klassischen* Algorithmentheorie. Doch ist aus ihr ein spezielles Gebiet der Mathematik hervorgegangen, die **Komplexitätstheorie**. Sie ist dem Aufwandsproblem gewidmet und kann als moderner Zweig der Algorithmentheorie aufgefasst werden. Ihre Behandlung hätte sich im Rahmen des Kapitels 8 angeboten, doch verschieben wir sie auf das Kapitel 21, wo wir uns mit dem Begriff der Komplexität aus sehr allgemeiner Sicht auseinandersetzen werden.

8.4* Universelle algorithmische Systeme

8.4.1 Problemstellung

Wir haben gesehen, dass mittels Prädikaten nichtberechenbare Funktionen festgelegt werden können, für deren Berechnung also weder eine vom Menschen interpretierbare, terminierende Vorschrift noch ein terminierendes Computerprogramm existiert. Man könnte nun vermuten, dass es eine allgemeine Methode geben müsste, nach der sich für jede realisierbare Funktion ein Berechnungsalgorithmus formulieren lässt. Eine solche universelle Methode nennen wir **universelles algorithmisches System**.

Da Algorithmen sprachliche Gebilde, in unserem Begriffssystem sprachliche Operatoren sind, bedarf es einer Sprache, um sie zu artikulieren, und da die Sprache eindeutig interpretierbar sein soll, muss es eine *formale* Sprache sein. Eine formale Sprache für die Artikulierung von Algorithmen heißt **algorithmische Sprache**. Um in ihr Algorithmen artikulieren zu können, muss sie interpretiert (im Sinne der Mathematik), also eine *Kalkülsprache* sein (vgl.Kap.5.4 [5.2]). Sie heißt *universell*, wenn in ihr sämtliche berechenbaren Funktionen artikuliert werden können. Demnach ist ein universelles algorithmisches System (eine universelle Methode zur Artikulierung von Algorithmen) nichts anderes als eine *universelle* algorithmische

Sprache. Wenn es eine solche Sprache gibt, muss sich jeder, in irgendeiner anderen Sprache artikulierte Algorithmus in diese universelle Sprache übersetzen lassen.

Viele Methoden zur Artikulierung von Algorithmen sind vorgeschlagen worden. Es kann nicht das Ziel der folgenden Ausführungen sein, einen vollständigen Überblick über die diesbezüglichen Ergebnisse der Algorithmentheorie zu geben. Unser Ziel ist es vielmehr, die Schlussfolgerungen, die sich aus der USB-Methode hinsichtlich der Berechenbarkeit von Funktionen ziehen lassen, mit den Schlussfolgerungen der Algorithmentheorie in Beziehung zu setzen. Darüberhinaus soll der Leser mit einigen Ideen und Begriffen bekannt gemacht werden, die sich stimulierend auf die Entwicklung der Rechentechnik ausgewirkt haben und die bei der Entwicklung von Computerhardware und von Programmiersprachen Pate gestanden haben.

Daneben soll die Diskussion ein Schlaglicht auf eine Art geistiger Produktivität werfen, die scheinbar unsinnig ist. Sie soll demonstrieren, zu welchen Meisterleistungen der suchende Intellekt durch unerreichbare Ziele angespornt werden kann. Ein ähnliches Beispiel haben wir schon kennen gelernt, den gödelschen Unvollständigkeitssatz als Ergebnis der Suche nach einem allgemeinen Entscheidungskriterium hinsichtlich der Wahrheit bzw. Falschheit von Aussagen formalisierter Theorien, das es nicht gibt.

Die Suche nach einem algorithmischen System, in welchem jede “irgendwie” berechenbare Funktion artikuliert werden kann, verfolgt ein unerreichbares Ziel. Erst mit der Zeit wurde klar, dass ein solches System nicht angebbar ist, jedenfalls nicht auf der Basis unseres gegenwärtigen Erkenntnisstandes. Denn zu den realen Operatoren, die Funktionen berechnen können, gehören nicht nur die Computer, sondern
23 auch die Menschen. Was Menschen können oder nicht können, ist unbekannt, genauer gesagt, die Frage lässt sich - ebenso wie die nach der Willensfreiheit - introspektiv nicht beantworen. Um sie beantworten zu können, müssten die Berechnungsmechanismen des Gehirns vollständig bekannt sein, was nicht der Fall ist.

Es kann also nicht bewiesen und infolgedessen auch nicht unbedingt erwartet werden, dass sich für jede vom Menschen realisierbare Funktion ein Berechnungsalgorithmus angeben lässt. Dies ist der Grund dafür, dass die Quintessenz der (klassischen) Algorithmentheorie kein Theorem, sondern eine Hypothese ist, die *These von* CHURCH oder **churchsche These**. Sie wird in Kap.8.5 behandelt. Doch nehmen wir ihren Inhalt schon jetzt voraus. Sie wurde in vielen Varianten formuliert. Wir wählen folgende: *Jede **effektiv berechenbare** Funktion ist eine **rekursive** Funktion*.

Der Begriff der *effektiv berechenbaren* Funktionen ist ein *intuitiver* Begriff. Intuitiv wird eine Funktion als berechenbar bezeichnet, wenn *irgendwer irgendwie* ihre Funktionswerte bestimmt hat oder bestimmen kann. Ist das der Fall, spricht man von *effektiver Berechenbarkeit*. In unserer Sprechweise ist eine effektiv berechenbare Funktion eine *realisierbare* Funktion.

Rekursive Funktionen, sind solche, die in einem bestimmten algorithmischen System, das in Kap.8.4.6 eingeführt wird, definierbar sind. In den folgenden Kapiteln

werden einige algorithmische Systeme in unterschiedlicher Ausführlichkeit dargelegt und zwar:

1. Turingmaschine oder Turingautomat
2. Unbeschränkte Registermaschine (URM)
3. Markovalgorithmus
4. Lambda-Kalkül von CHURCH
5. Rekursive Definition von Funktionen
6. Methode der uniformen Systembeschreibung (USB-Methode).

Ausführlichere Darstellungen der Methoden 1 bis 5 findet man in der Literatur[11]. Die Methoden 1, 4 und 5 sind fast gleichzeitig um das Jahr 1936 vorgeschlagen worden. Die Funktionen, die in einem bestimmten algorithmischen System beschrieben und berechnet werden können, fassen wir zu einer Klasse zusammen, sodass 6 Klassen unterschieden werden können, die wir folgendermaßen bezeichnen:

- Klasse der Turing-Funktionen
- Klasse der URM-Funktionen
- Klasse der Markov-Funktionen
- Klasse der Church-Funktionen
- Klasse der rekursiven Funktionen
- Klasse der USB-Funktionen.

In der Algorithmentheorie sind Bezeichnungen üblich, die das Wort *berechenbar* enthalten. So wird z.B. nicht von Turing-Funktionen, sondern von Turing-berechenbaren Funktionen gesprochen.

In den folgenden Ausführungen darf der Leser ***keine Einführung in die Algorithmentheorie*** *oder in die* ***Theorie der Berechenbarkeit*** *sehen, sondern lediglich eine Erläuterung derjenigen Begriffe und Schlussfolgerungen der Algorithmentheorie, die erforderlich sind, um die gestellten Ziele zu erreichen. Hauptziel ist der Nachweis, dass USB-Funktionen rekursive Funktionen und rekursive Funktionen USB-Funktionen sind.*

8.4.2 Turingmaschine

Wenn die Frage, welche Funktionen berechenbar sind, auch nicht allgemein entschieden werden kann, so sollte es doch möglich sein anzugeben, welche Funktionen ein Computer bekannter Konstruktion berechnen kann. Das wirft die Frage auf, ob es ein allgemeines *Konstruktionsprinzip* gibt, nach dem alle Computer aufgebaut sind, und ob es elementare Operationen gibt, aus denen sämtliche Computeroperationen komponiert werden. Dieser Frage werden wir in Kap.9 nachgehen.

---

11 Siehe z.B. [Hermes 71], [Schnorr 74], [Cutland 80], [Stetter 88], [Penrose 91], [Sander 94], [Schöning 95], [Werner 95]. Die USB-Methode ist in [Jungclaussen 80-90] entwickelt worden, zunächst als Sprache zur Beschreibung kausaldiskreter Systeme, nicht als algorithmisches System. Zusammenfassende Darstellungen der USB-Methode sind in [Jungclaussen 85] und [Jungclaussen 90] veröffentlicht.

Von einer ähnlichen Frage ging ALLEN TURING aus. Er suchte nach dem denkbar *einfachsten* realen, universellen Zeichenkettenoperator (informationellen Operator) und erfand eine Maschine, die seitdem **Turingmaschine** oder **Turingautomat** genannt wird.

In gewisser Hinsicht ähnelt die Turingmaschine einem Magnetbandgerät oder Kassettenrecorder. Sie verfügt über ein unbeschränktes Speicherband, das als Träger einer Kette beliebig vieler Speicherzellen dient. Jede Zelle kann ein Zeichen des *Automatenalphabets* aufnehmen. Ein Tastkopf dient dem Lesen und Beschreiben des Bandes. Er kann durch die Steuereinheit bewegt werden. Das Band dient gleichzeitig als Speicher und als Ein- und Ausgabegerät.

Die Turingmaschine arbeitet taktweise. In jedem Takt liest die Steuereinheit das unter dem Kopf befindliche Zeichen, den *Input*, überführt es in ein neues Zeichen, den *Output*, und überschreibt auf dem Band das alte Zeichen durch das neue. Die Steuereinheit besitzt eine endliche Anzahl von Zuständen, darunter einen Endzustand. In jedem Schritt geht sie aus dem jeweils "alten" in den "neuen" Zustand über und kann gleichzeitig den Kopf bewegen.

Welche konkrete Aktion in einem Schritt ausgeführt wird, hängt vom momentanen Zustand und vom Input ab. Die Analogie zum abstrakten Automaten ist offensichtlich und die Bezeichnung Turing*automat* gerechtfertigt. Ebenso wie der abstrakte Automat verfügt der Turingautomat über eine *Automatentafel*. Diese ordnet einem (nicht unbedingt jedem) Paar (Input, alter Zustand) ein Tripel (Output, neuer Zustand, Verschiebung) zu. Die Verschiebung bezieht sich auf den Tastkopf, der um eine Zelle nach rechts oder nach links verschoben werden kann; die Verschiebung kann auch unterbleiben.

Um die Turingmaschine eine Zeichenkette verarbeiten (transformieren) zu lassen, wird das Band mit der Zeichenkette, dem Eingabewort *x*, beschriftet, der Kopf auf irgendeine Zelle des Bandes eingestellt und die Turingmaschine in dem so definierten **Anfangszustand** gestartet. Nach dem Start sucht die Turingmaschine das erste Zeichen des Eingabewortes und transformiert es dann schrittweise. Sobald die Steuereinheit in den Haltezustand übergeht, hält sie an. Die Zeichenkette, die nun auf dem Band steht, ist das Ausgabewort *y*. Wenn man der Reihe nach die Wörter einer Wortmenge *X* eingibt und wenn die Turingmaschine zu jedem Eingabewort *x* ein Ausgabewort *y* ausgibt, sagt man, dass sie die Funktion $f: X \rightarrow Y$ berechnet. Für die Berechnung einer anderen Funktion ist ein Turingautomat mit einer anderen Automatentafel, eventuell auch mit einem anderen Alphabet erforderlich.

Wenn das aktuelle Paar (Input, alter innerer Zustand) nicht in der Automatentafel enthalten ist, hält der Turingautomat an, obwohl er sich nicht im Endzustand befindet. Es ist auch möglich, dass er niemals anhält, weil er den Endzustand niemals erreicht. In beiden Fällen sagt man, dass die Funktion $f(x)$ für die betreffenden x-Werte nicht definiert ist und spricht von **partieller** Funktion. Ist sie für sämtliche x-Werte definiert, heißt sie **total**. Diese Unterscheidung ist nicht an die Turingmaschine gebunden. Allgemein hat man vereinbart: *Eine Funktion heißt* ***total*** *bzw.* ***partiell***

hinsichtlich einer vorgegebenen Argumentwertemenge, wenn die Funktionswerte für sämtliche bzw. nicht für sämtliche Argumentwerte definiert sind.

Was Turingmaschinen zu leisten vermögen, ist nicht ohne weiteres zu erkennen. Angesichts ihrer äußerst elementaren Funktionsweise ist der empirische Fakt überraschend, dass keine effektiv berechenbare Funktion gefunden worden ist, für die sich nicht eine Turingmaschine angeben lässt, die sie berechnet. Turing selber hat seinen Automaten zum **universellen** Automaten erweitert, der die Funktionsweise *jedes* **speziellen** Automaten simulieren kann, also jedes Automaten mit einer bestimmten Automatentafel. Dazu wird ihm per Bandinschrift die Tafel desjenigen Automaten mitgeteilt, den er simulieren soll.

Turing war überzeugt, das eingangs gestellte Ziel erreicht zu haben, und meinte, dass sich zu jeder effektiv berechenbaren Funktion eine Berechnungsvorschrift in Form einer Automatentafel angeben lässt, oder, wie man auch sagt, dass jede effektiv berechenbare Funktion *turingberechenbar* ist, oder in unserer Sprechweise, dass jede effektiv berechenbare Funktion eine *Turingfunktion* ist. Doch lässt sich das, wie wir wissen, nicht beweisen. Die Begriffe der Turingmaschine und der Turingberechenbarkeit sind zu Grundbegriffen der Algorithmentheorie und der theoretischen Informatik geworden.

Turing befolgte das Trägerprinzip; seine Maschine ist realisierbar. Doch braucht sie nicht gebaut zu werden, um mit ihr zu arbeiten, beispielsweise um zu untersuchen, welche Funktionen die *gedachte* Maschine berechnen kann.

8.4.3 Unbeschränkte Registermaschine (URM)

Bedeutend jüngeren Datums ist die **unbeschränkte Registermaschine**, kurz **Registermaschine**, abgekürzt **URM**. Sie wurde von NIGEL CUTLAND eingeführt und beschrieben [Cutland 80]. Vorgänger der URM war ein Vorschlag von J.C.SHEPHERDSON und H.E.STURGIS aus dem Jahre 1963. Auch die Registermaschine ist eine *fiktive* Maschine, jedoch nicht so elementarer Konstruktion wie die Turingmaschine, und die Operationen, die sie ausführt, sind bedeutend komplexer.

Die URM besteht aus einer unbeschränkten Menge von **Registern** (Speicherplätzen für Zeichenketten, z.B. für Zahlen). In einem Register kann eine ganze positive Zahl unbeschränkter Länge abgspeichert werden. Jedes Register hat einen Namen (eine Adresse). Der Inhalt eines Registers kann mittels bestimmter Befehle verändert werden. Ein Programm ist eine durchnummerierte Folge von Befehlen, die von einem fiktiven Interpretierer (einem Menschen oder einem Gerät) ausgeführt werden.

Es gibt 4 Befehle:

1. Register löschen (0 einspeichern);
2. Inhalt eines Registers inkrementieren (1 addieren);
3. Inhalte zweier Register austauschen;
4. bedingter Sprung, d.h. Vor- oder Zurückspringen im Programm, wenn zwei bestimmte Register die gleiche Zahl enthalten. Der anzuspringende Befehl und die zu vergleichenden Register werden im Befehl angegeben.

Die Ähnlichkeiten und Unterschiede zwischen Registermaschine und universeller Turingmaschine sind offensichtlich. Beide sind Automaten mit unbeschränktem Speicher. Ein Programm der Registermaschine entspricht der Automatentafel der Turingmaschine, erinnert aber schon sehr an Programme, die in der sog. Assemblersprache gängiger Computer (siehe Kap.15.4) geschrieben sind. Ein Programm legt - ebenso wie eine Automatentafel - eine Funktion fest. Es lässt sich zeigen, dass die Registermaschine dasselbe leistet (dieselben Funktionen berechnen kann), wie die universelle Turingmaschine. Eine Funktion, die durch ein URM-Programm festgelegt und berechnet werden kann, heißt **URM-berechenbar**; wir nennen sie **URM-Funktion**.

8.4.4 Markovalgorithmus

Eine andere, von der turingschen scheinbar sehr unterschiedliche Idee liegt dem *Normalalgorithmus* von A.A.MARKOV[12], kurz dem **Markovalgorithmus** aus dem Jahre 1954 zugrunde. Auch Markov suchte nach einer universellen Vorschrift für das Transformieren von Zeichenketten, wobei er - im Gegensatz zu Turing - nicht das schrittweise Ersetzen einzelner Zeichen, sondern längerer Teile von Zeichenketten (von Wörtern) im Auge hatte. Er führte keine fiktive Maschine ein, sondern ging offenbar davon aus, dass seine Normalalgorithmen von Menschen interpretiert werden.

Aufgabe des Interpretierers ist es, Zeichenketten nach folgender Vorschrift zu transformieren. Gegeben ist eine Liste (geordnete Menge) von Ersetzungsregeln, sogenannten **Substitutionsregeln**, z.B. 2+3→5 oder *ab*→*cd*, was bedeutet, dass 2+3 durch 5 bzw. *ab* durch *cd* ersetzt (substituiert) werden kann. Die Transformation erfolgt schrittweise. In jedem Schritt geht der Interpretierer die Regelliste durch. Er beginnt mit der ersten Regel und sucht die zu transformierende Zeichenkette von links beginnend nach der linken Seite der Regel durch. Wenn die Suche erfolglos ist, geht er zur nächsten Regel über. Sobald eine Regel anwendbar ist, wenn also ihre linke Seite in der zu transformierenden Zeichenkette enthalten ist, wird die betreffende Substitution vorgenommen. Wenn die Zeichenkette bis zum Ende durchsucht ist, wiederholt sich der Prozess mit der nächsten Regel. Wenn keine Regel mehr anwendbar ist oder wenn auf der rechten Seite der gefundenen Regel das *Stopzeichen* steht, wird der Transformationsprozess beendet. Das Stopzeichen gehört zum verwendeten Alphabet.

Das Transformieren einer Zeichenkette erinnert an die Arbeit mit einer Formelsammlung, in der man nach einer passenden Formel sucht. Dabei findet man z.B. sin$x$/cos$x$ = tan$x$, was bedeutet, dass in einem mathematischen Ausdruck sin$x$/cos$x$ durch tan$x$ ersetzt werden darf. Natürlich wird man mit dem Suchen nicht auf Seite 1 beginnen, sondern intelligenter vorgehen.

---

12 Sohn des Wahrscheinlichkeitstheoretikers A.A.Markov.

Ein Operator, der Zeichenketten gemäß Formeln umwandelt, wird **Formelmanipulator** genannt. Ein Markovalgorithmus ist also, zusammen mit seinem Interpretierer, ein Formelmanipulator. Als Manipulator kann ein Computer dienen, der über entsprechende Programme verfügt. Wenn der Programmierer sich nichts Intelligenteres hat einfallen lassen, beginnt der Computer jede Suche mit der ersten Formel, sozusagen auf Seite 1. In Kap.15.8 wird näher auf die Formelmanipulation eingegangen.

Transformiert ein Markovalgorithmus Wörter einer Wortmenge X eindeutig in Wörter einer Menge Y, so sagt man, dass er die Funktion $f: X \rightarrow Y$ berechnet. Eine solche Funktion wird **markovberechenbar** genannt; wir nennen sie **Markovfunktion**. Wieder stellt sich die Frage, ob die Methode universell ist, d.h. ob jede effektiv berechenbare Funktion auch markovberechenbar ist. Markov selber war, ebenso wie Turing, der Meinung, dass seine Methode universell sei. Es stellte sich heraus, dass er damit ebenso Recht hatte wie Turing und dass die Klasse der Markovfunktionen mit der Klasse der Turingfunktionen identisch ist. Wir kommen darauf in Kap.15.8 zurück.[13]

Zwischen Markovalgorithmus und Turingmaschine sind gewisse Ähnlichkeiten zu erkennen. Ein *spezieller* Markovalgorithmus, also ein Markovalgorithmus mit einer bestimmten Regelliste, entspricht einer *speziellen* Turingmaschine; der Regelliste entspricht die Automatentafel und dem Interpretierer des Markovalgorithmus die Steuereinheit der Turingmaschine. Die Arbeitsweise des Interpretierers von Markovalgorithmen ist jedoch nicht soweit “durchautomatisiert” wie die der Steuereinheit der Turingmaschine.

8.4.5 Rekursive Funktionen und USB-Funktionen

Historisch gesehen hätte die nun zu besprechende *rekursive* Methode, Funktionen festzulegen, am Anfang stehen müssen. Die nach dieser Methode beschreibbaren und berechenbaren Funktionen heißen **rekursive Funktionen**. Wir werden die Methode gemeinsam mit der jüngsten der in Kap.8.4.1 aufgezählten sechs Methoden der Algorithmenartikulierung, der USB-Methode, darlegen und die rekursive Definition von Funktionen vollständig in die Sprache der uniformen Systembeschreibung übersetzen, was einer Veranschaulichung der rekursiven Definition gleichkommt. Denn die USB-Methode arbeitet mit “anschaulichen Bildern”, ihre Sprache ist eine *zweidimensionale* Sprache, sie verwendet graphische Darstellungen. Auf diese Weise wird zugleich nachgewiesen, dass jede rekursiv definierbare Funktion eine USB-Funktion ist.

Der Idee der rekursiven Metode sind wir bereits in Kap.7.2 [7.12] begegnet. Dort war das Multiplizieren auf das iterative Addieren und dieses auf das iterative Inkrementieren (Erhöhen um 1) zurückgeführt worden. Man erinnere sich an den

13 Ausführlicher ist der Markovalgorithmus in [Malcew 74] beschrieben.

ABC-Schützen. Die Idee der Methode besteht nun darin, die Definition und die Berechnung *jeder beliebigen* Funktion auf das Inkrementieren zurückzuführen.

Es bedurfte der Bemühungen vieler Mathematiker - genannt seien J.HERBRAND, K.GÖDEL und S.C.KLEENE -, um diese Idee in eine mathematische Form zu gießen, und so die Begriffe der Funktion und des Algorithmus exakt zu definieren. Das Ergebnis war der Begriff der *rekursiven Funktion*. In ihm sind zwei Bedeutungskomponenten des Wortes "Rekursion" miteinander verknüpft, das Zurück*führen* und das Zurück*greifen*. In jedem Schritt einer rekursiven Berechnung wird auf das Resultat derjenigen Operation (z.B. des Addierens) *zurückgegriffen*, auf welche die auszuführende Operation (z.B. das Multiplizieren) *zurückgeführt* wird.

Um zu einer konstruktiven Methode zur Festlegung von Funktionen (Operationen, Operatoren) zu gelangen, m.a.W. um eine Komponierungsmethode beliebiger Kompositoperatoren zu entwickeln, ist es notwendig, eine Menge elementarer, d.h. nicht weiter dekomponierbarer Operatoren sowie eine Menge von Komponierungsregeln festzulegen. Beide Mengen sollten möglichst klein sein. Dafür gibt es viele Möglichkeiten. Eine Möglichkeit ist die USB-Methode. Eine andere Möglichkeit, die sehr häufig der Definition rekursiver Funktionen zugrunde gelegt wird (z.B. in [Hermes 71]) geht von *einer einzigen elementaren Operation* aus, dem **Inkrementieren** (Erhöhung um 1). Als Anfangswert des Inkrementierens wird die Null festgelegt. Es gibt vier Komponierungsregeln, nach denen aus dem Inkrementieren Kompositoperationen komponiert werden können:

1. Funktionale Substitution
2. Selektion
3. Rekursive Iteration
4. Minimalisierung.

Wir nennen sie *rekursive* **Komponierungsmittel** ; Funktionen, die sich mit ihrer Hilfe beschreiben lassen, nennen wir **rekursive Funktionen**. Die Operanden (Argument- und Funktionswerte) rekursiver Funktionen sind ganze nichtnegative Zahlen. Die Operandenmenge kann durch Umcodierung erweitert werden, wie in Kap.8.5 [37] gezeigt wird.

Den Begriff der *rekursiven Beschreibbarkeit* werden wir im Weiteren in folgendem verallgemeinertem Sinne verwenden: *Ein Komponierungsmittel heißt* ***rekursiv beschreibbar****, wenn es mit Hilfe der genannten vier rekursiven Komponierungsmittel beschreibbar ist.* In der Algorithmentheorie werden Komponierungsregeln eventuell als elementare Operatoren (Funktionen) aufgefasst. Die Wirkungsweise der Komponierungsregeln sollen der Reihe nach besprochen werden.

Funktionale Substitution

Substitutionen sind wir bereits begegnet, und zwar im Zusammenhang mit dem Markovalgorithmus. Dort wurden Zeichenketten durch andere Zeichenketten substituiert, ungeachtet der Bedeutung, die diese Zeichenketten in ihrem Kontext vielleicht

besitzen. Jetzt führen wir eine spezielle Art der Zeichenkettensubstitution ein. Sie besteht darin, dass eine Argumentvariable einer Funktion durch eine andere Funktion ersetzt wird, die den Wert der Argumentvariablen berechnet. In diesem Fall sprechen wir von **funktionaler Substitution**. Wenn beispielsweise in der Funktion $y = f(x)$ die Variable x durch $g(x)$ ersetzt wird, liegt eine funktionale Substitution vor; sie liefert die **geschachtelte** Funktion

$$y = f(g(x)) \tag{8.6}$$

Wenn nichts anderes gesagt wird, ist im Weiteren unter einer Substitution eine funktionale Substitution zu verstehen.

Eine Substitution, bei der eine einzige Argumentvariable substituiert wird, lässt sich nach der USB-Methode als *Sequenz* (Kette) zweier Operatoren darstellen. Gemeinsam bilden sie einen Kompositoperator, der eine *Kompositfunktion*, und zwar eine geschachtelte Funktion berechnet. Die Funktion (8.6) ist dafür ein Beispiel. Sie wird von einem *f*-Operator berechnet, dem ein *g*-Operator vorgeschaltet ist.

Auf diese Weise können auch mehrfach geschachtelte Funktionen dargestellt werden. Wenn beispielsweise der Operator h_1 in Bild 8.7a die Funktion $y = 2x$ berechnet, h_2 die Funktion $y = \sin x$ und h_3 die Funktion $y = x^2$, dann berechnet die Operatorenkette die doppelt geschachtelte Funktion $y = (\sin(2x))^2$. Ohne Angabe der konkreten Funktionen lässt sich die doppelte Schachtelung folgendermaßen notieren:

$$y = f(x) = h_3(h_2(h_1(x))) \tag{8.7}$$

Bild 8.7a zeigt das entsprechende Operatorennetz.

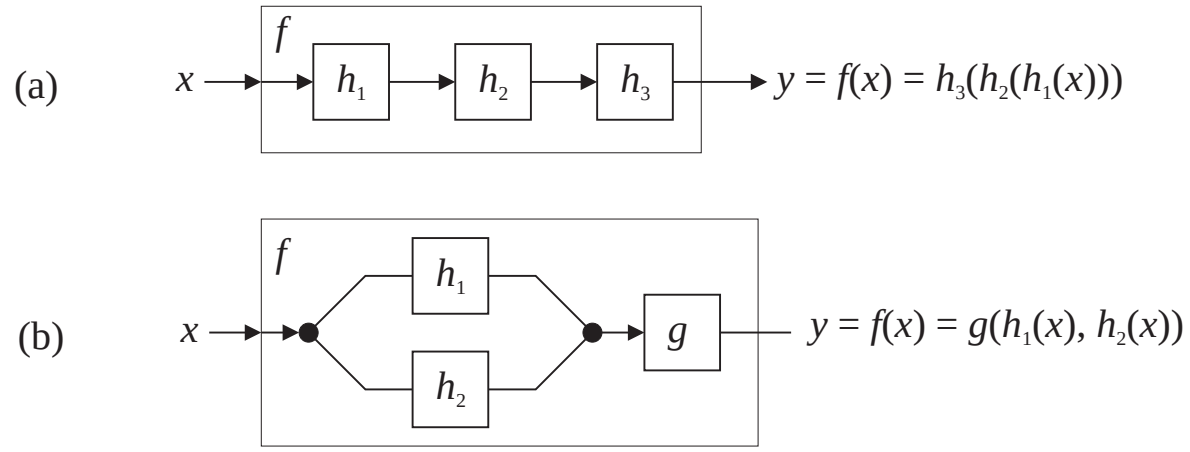

Bild 8.7 Funktionale Substitution; **(a)** - einstellig; **(b)** - zweistellig

In den bisherigen Beispielen für Substitutionen wird das Argument einer einstelligen Funktion substituiert. Solche Substitutionen nennen wir **einstellig**. Die funktionale Substitution lässt sich auch auf mehrstellige Funktionen anwenden. Dann sprechen wir von **mehrstelliger Substitution**. Bild 8.7b zeigt das Operatorennetz

einer zweistelligen Funktion. Die Argumente werden durch die Funktionen $h_1(x)$ bzw. $h_2(x)$ substituiert, die beide von derselben Variablen x abhängen. Die Gabel ist also eine Kopiergabel und die Kompositfunktion

$$f(x) = g((h_1(x),h_2(x)) \tag{8.8a}$$

ist einstellig. Das Operatorennetz von Bild 8.1 ist dafür ein konkretes Beispiel. Dabei ist h_1 der Potenzierer, h_2 ist je nach Stellung der Zweigeweiche vor dem Sinusoperator entweder der Sinusoperator oder der Identitätsoperator (der Übergabeweg, der den Sinusoperator überbrückt) und g ist der Additionsoperator. Wenn die substituierenden Funktionen h_1 und h_2 von verschiedenen Variablen abhängen, ist die Gabel eine Spaltegabel und die Kompositfunktion ist zweistellig:

$$f(x_1,x_2) = g((h_1(x_1),h_2(x_2)). \tag{8.8b}$$

Bild 8.7 und die Formeln (8.8a) und (8.8b) demonstrieren, wie sich starre Flussknoten mit Hilfe der rekursiven Komponierungsmittel beschreiben lassen. Den Gabeln entsprechen Funktionspaare und den Vereinungen zweistellige Funktionen.

An dieser Stelle sei auf folgenden Sachverhalt aufmerksam gemacht. Weder in den Operatorennetzen von Bild 8.7 noch in den entsprechenden geschachtelten Funktionen (8.7), (8.8a) und (8.8b) treten die Operanden, die übergeben bzw. substituiert werden, explizit auf, mit Ausnahme des externen Eingabeoperanden x bzw. (x_1,x_2). Es ist leicht einzusehen, dass dies nicht erforderlich ist. Denn die Operanden bzw. Argumente ergeben sich aus den Notationsregeln (Syntaxregeln) der jeweiligen Sprache. Die zweidimensionale Sprache von Bild 8.7 ist die Sprache der uniformen Systembeschreibung, die **USB-Sprache**. Die eindimensionale Sprache, in der die Ausdrücke (8.7) und (8.8) notiert sind, heißt **funktionale Sprache**, weil die Notation nur Funktionsbezeichner und Klammern verwendet, wenn man von den Bezeichnern der externen Eingabeoperanden absieht.

In der USB-Sprache zeigt eine Verbindung *von* einem Operator (Vorgängeroperator) *zu* einem anderen Operator (Nachfolgeoperator) an, dass der Eingabeoperand des Nachfolgeoperators mit dem Ausgabeoperanden des Vorgängeroperators identisch ist. Darum braucht er nicht angegeben zu werden. Die Rolle der Verbindungen der USB-Notation übernimmt in der funktionalen Notation die Schachtelung. Sie zeigt an, dass der Wert, den eine innere Funktion berechnet, der äußeren Funktion als Argumentwert "übergeben" wird.

24 Die Eigenschaft, dass innere Operanden nicht auftreten, wird uns im Weiteren wiederholt beschäftigen. Sie weist USB- und funktionale Sprachen als *nichtimperative* Sprachen aus, d.h. als Sprachen, in denen keine *imperativen Algorithmen* artikuliert werden, also Algorithmen, die in jedem Berechnungsschritt alle beteiligten Operanden explizit angeben (vgl. Kap.7.2 [7.10]). Ein imperativer Algorithmus zur Berechnung der Funktion (8.8a) könnte folgende Form haben:

$a := h_1(x)$
$b := h_2(x)$
$c := g(a,b)$
$f := c$

Man sagt, dass diese Vorschrift *imperativ*, die Vorschrift (8.8a) dagegen *funktional* notiert ist und spricht von **imperativer** bzw. **funktionaler Notation**.

In der Mathematik wird bekanntlich vorwiegend die funktionale Notation benutzt. Sie ist sehr kompakt, hat aber den Nachteil, dass sie zu Missverständnissen führen kann, die daher rühren, dass ein Funktionsbezeichner zum einen eine *Funktion* bezeichnet, und dass er zum anderen zusammen mit einem Argumentwert bzw. Argumentwertetupel den Wert der Funktion bezeichnet. Wenn ein Funktionsargument durch eine Funktion *f* substituiert wird, stellt *f* bei der Auswertung nicht die Funktion, sondern einen bestimmten Funktionswert dar. Dies ist die Wurzel des 25
bekannten Streits, ob mit *f*(*x*) ein *bestimmter* Wert oder *alle* Werte von *f* gemeint sind. Diese Zweideutigkeit hat CHURCH durch die Einführung des Lambda-Operators behoben (siehe Kap.8.4.7).

Selektion

Die Argumente von Funktionen können Tupel, z.B. Vektoren (Zahlentupel) sein. Um mit Vektoren zu rechnen, um z.B. das Skalarprodukt zweier Vektoren zu bilden, muss auf die einzelnen Komponenten der Vektoren (Tupel) zugeriffen werden. Dieses Zugreifen wird Komponentenselektion oder kurz **Selektion** genannt. Der ausführende Operator heißt **Selektor**. Er heißt ***n*-stellig**, wenn er aus einem Tupel *n* Elemente auswählt.

Bild 8.8 zeigt die USB-Darstellung von Selektoren und demonstriert damit, dass aus der Sicht der USB-Methode auch die Selektion keine Bausteinoperation, sondern ein Mittel des Komponierens ist. Sie kann durch eine Spaltegabel beschrieben werden, über deren einen Ausgang die zu selektierende Komponente weitergeleitet wird, während der andere Ausgang stillgelegt ist. Dabei handelt es sich um einen **starren** (nicht steuerbaren) Selektor (Bild 8.8a). Die Algorithmentheorie arbeitet i.Allg. nur mit *einstelligen* starren Selektoren.

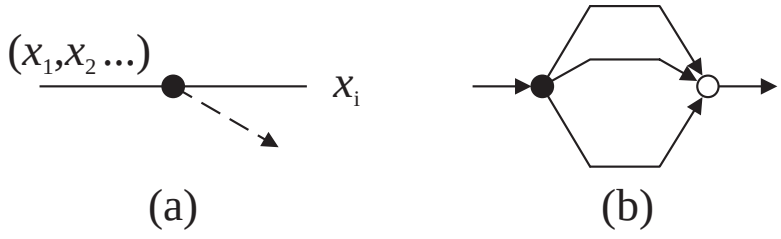

Bild 8.8 Selektoren. **(a)** - starrer Seletor; **(b)** - steuerbarer Selektor.

Bild 8.8b zeigt einen **steuerbaren Selektor**. Er besteht aus einer Spaltegabel und einer Sammelweiche mit soviel Eingängen, wie das Eingabetupel Elemente (bzw. Teiltupel) besitzt. Durch Steuerung der Weiche kann das gewünschte Element des Tupels ausgewählt werden. Steuerbare Selektoren kann man als "Entscheider" zwischen mehreren "Angeboten" auffassen. Der einfachste steuerbare Selektor, der aus einem Paar ein Element auswählt, entscheidet eine Alternative. Bild 8.8b zeigt, dass sich Sammelweichen mit Hilfe der Selektion rekursiv beschreiben lassen. Gleichzeitig zeigt es, dass sich Alternativmaschen rekursiv beschreiben lassen, vorausgesetzt, in der Masche befinden sich niemals mehrere Operanden gleichzeitig (vgl. Kap.8.1 [1])

Selektoren stellen **Kompositflussknoten** dar, also Flussknoten, die aus den "elementaren" Flussknoten von Bild 8.2 komponiert werden. Man kann sich die verschiedensten Kompositflussknoten ausdenken. Beispielsweise lassen sich mit Hilfe von Selektoren steuerbare Spaltegabeln realisieren. In Kap.12.3.2 werden umfangreiche Kompositflussknoten und ihre elektronische Realisierung besprochen.

Nachdem wir die *Gabel mit einem einzigen Ausgang* als Symbol für den *starren Selektor* eingeführt haben, bietet es sich an, die *Zweigeweiche mit einem einzigen Ausgang* (dargestellt als kleiner Kreis innerhalb einer Verbindungslinie) als Symbol für das *Tor* einzuführen. Ein **Tor** kann als Zweigeweiche mit einem toten Ausgabezweig angesehen werden.

Rekursive Iteration

Eine Kette identischer Operatoren führt ein und dieselbe Operation wiederholt (iteriert) aus. Der so entstehende Kompositoperator heißt **Iterator**. Eine Iteration kann demnach als *sequenzielle Substitution* aufgefasst werden, bei der jedes Mal die gleiche Substitution ausgeführt wird. *Iteration ist also eine spezielle Art der sequenziellen Substitution.* Die hardwaremäßige Realisierung kann bei hoher Iterationszahl (Wiederholungszahl) recht aufwendig werden. Der Aufwand lässt sich dadurch verringern, dass der Operator nur einmal realisiert, aber mit einer Rückkopplungsschleife versehen wird. Der Potenzierer (pot) in Bild 8.1 ist dafür ein Beispiel. Die Kette ist gewissermaßen zu einer Iterationsschleife *eingerollt.* Umgekehrt kann eine Iterationsschleife in eine Kette "*ausgerollt*" werden; bei ihrer Auswertung *muss* sie ausgerollt werden.

Die Möglichkeit, Iterationen durch Rückkopplung zu realisieren, besteht nicht nur für reale, sondern auch für sprachliche Operatoren und Algorithmen. Dann kann die Rückkopplung durch Zurückspringen im Algorithmus verwirklicht werden, oder dadurch, dass der Algorithmus sich selber enthält (sich selbst aufruft). Diese Art der Iteration nennen wir **rekursive Iteration**. Wenn nichts Gegenteiliges gesagt wird, ist im Weiteren unter Iteration stets *rekursive* Iteration und unter einem Iterator ein Operator zu verstehen, der eine *rekursive* Iteration ausführt.

Bild 8.9 zeigt den Iterator in seiner allgemeinsten Form als Operatorennetz. Der Leser wird erkennen, dass die Zweigeweiche Z2 und die Sammelweiche S2 die Weichen einer Alternativmasche und Z1 und S1 die Weichen einer Rückkopplungsschleife sind. Der Iterator führt die Operation *h* mehrmals aus. Die Rückkopplungsschleife, in der die eigentliche Iteration abläuft, ist fett gezeichnet. Mit *n* ist die Anzahl der ausgeführten Iterationsschritte bezeichnet, wobei die erste Operationsausführung mitgezählt wird, obwohl sie eigentlich noch keine Wiederholung darstellt. Die Funktion, die ein Iterator bei *n* Iterationsschritten liefert, bezeichnen wir mit f_n.

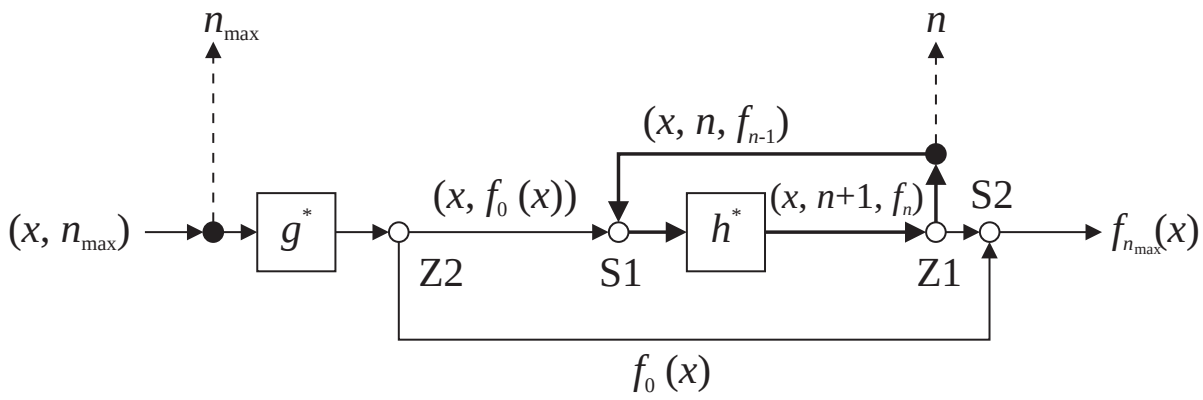

Bild 8.9 Rekursiver Iterator

Der Potenzierer von Bild 8.1 (pot-Operator, gestrichelt umrahmt) ist ein Beispiel für einen rekursiven Iterator, denn das Potenzieren wird durch wiederholtes Multiplizieren verwirklicht. Der *h*-Operator ist in diesem Fall ein Multiplizierer. Das Operatorennetz des Iterators in Bild 8.9 unterscheidet sich von dem des pot-Operators in Bild 8.1 in dreierlei Hinsicht. Erstens ist der *h*-Operator zu einem h^*-Operator erweitert worden (die Erweiterung wird sogleich erläutert), zweitens ist ein zusätzlicher *g*-Operator eingeführt worden und drittens ist die *Vereinung* vor dem Multiplizierer durch eine *Sammelweiche* (S1) ersetzt worden (in Bild 8.1 waren die beiden Faktoren ausnahmsweise nicht gedanklich zu einem Paar vereint worden). Mit den Änderungen werden zwei Ziele verfolgt. Zum einen soll der Iterator universell sein, zum anderen soll das Operatorennetz des Iterators die Notationsweise widerspiegeln, die in der Algorithmentheorie üblich ist, insbesondere die Notationsweise der Formeln (8.9). Dies ist auch der Grund dafür, dass die Iterationszahl, die in den übrigen Kapiteln mit *it* bezeichnet wird, in diesem Kapitel mit *n* bezeichnet wird.

Der h^*-Operator ist dem bisherigen *h*-Operator gegenüber in zweifacher Hinsicht erweitert. Erstens gibt der Operator *f* nicht nur den laufenden Funktionswert f_n, sondern auch den Argumentwert *x* aus. Wir sagen, dass der Operator den Eingabeoperand durchzieht oder *durchschleppt*. Das ist notwendig (und hardwaremäßig leicht realisierbar), wenn die Funktion *h* nicht nur vom laufenden Iterationsergebnis f_n, sondern auch von *x* selber explizit abhängt, wie z.B. bei der iterativen Berechung der

Potenzfunktion, wobei in jedem Schritt das Produkt $x^n * x$ berechnet wird. Das Iterationsergebnis kann auch von der Iterationszahl abhängen, wie beispielsweise bei der iterativen Berechnung der Fakultät $n!$ oder bei der Berechnung der Sinusfunktion durch Reihenentwicklung (siehe Formel (15.5) in Kap.15.2). Dazu rüsten wir den Operator h^* mit einem Zähler aus, der die Iterationsschritte zählt und das aktuelle Zählergebnis ausgibt. Darin besteht die zweite Erweiterung des Operators h. Der vollständige h^*-Operator gibt das Tripel $(x,n+1,f_n)$ aus.

Der g-Operators stellt den Funktionswert f_0 bereit, mit dem die Iteration beginnt, es gilt $g(x) = f_0$. Im Falle des Potenzierers ist der Anfangswert $g(x) = f_0 = 1$, hängt also nicht von x ab. In Bild 8.1 ist seine Eingabe durch den senkrechten kurzen Eingabepfeil des Multiplizierers angedeutet. Der h-Operator berechnet im ersten Iterationsschritt das Produkt $1*x$. Beim Multiplizieren durch Iteration (durch wiederholtes Addieren) ist $f_0 = 0$. Der h-Operator benötigt beim Multiplizieren und beim Potenzieren durch Iteration außer dem f_0-Wert auch den x-Wert. Das ist auch bei vielen anderen iterativen Berechnungen der Fall. Der g-Operator muss dann das Paar (x,f_0) dem h-Operator übergeben, d.h. er muss den x-Wert durchschleppen.

Das Ausgabetripel des h^*-Operators kann über die Sammelweiche S1 auf den Eingang des h^*-Operators zurückgegeben werden, der es dann im nächsten Iterationsschritt weiterverarbeitet. In der Rückkopplungsschleife muss also - ebenso wie im Falle des Automaten in Bild 8.5 - ein “Schrittverzögerer” liegen; er ist nicht eingezeichnet. Im ersten Iterationsschritt wird über S1 dem h^*-Operator das Ausgabepaar des g-Operators zugeleitet. Über Z1, S2 und einen Selektor (nicht eingezeichnet), der f_n aus dem Ausgabetripel von h^* ausselektiert, kann das Resultat der Kompositoperation (der Iteration) ausgegeben werden. Wenn kein Iterationsschritt stattfinden soll, kann der Anfangswert f_0 vom g-Operator über Z2 und S2 ausgegeben werden.

Die Weichen müssen vom ausführenden realen Operator (Interpretierer) gestellt werden. Wird die Iteration vom Menschen ausgeführt, tritt die Weichenstellung als Entscheidung darüber ins Bewusstsein, was mit dem gerade berechneten Wert weiter zu geschehen hat. Ein technischer Iterator muss einen *Steueroperator* enthalten, der die erforderlichen Steuersignale generiert (vgl. Bild 8.1b). Der Steueroperator des Operatorennetzes von Bild 8.9 muss feststellen, ob die maximale Iterationszahl n_{max}, nach der die Iteration abgebrochen werden soll, erreicht ist. Dazu vergleicht er in jedem Iterationsschritt die laufende Iterationszahl n mit n_{max} und entscheidet das Abbruchprädikat, das z.B. $[n=n_{max}]$ oder $[n<n_{max}]$ lauten kann. Im zweiten Fall[14] muss der Steueroperator ein Steuersignal ausgeben, sobald das Steuerprädikat nicht mehr erfüllt ist. Das Steuersignal stellt die Zweigeweiche Z1, die bis dahin auf *Rückkopplung* gestellt war, auf *Ausgabe*.

14 Dem entspricht das Steuerprädikat [*it*<*n*] in den Bildern 8.1 und 20.10 und das Steuerprädikat [*z*<n] in Bild 15.4.

Aus Bild 8.9 ist abzulesen, dass der Iterator die Funktion $f(x,n)$ rekursiv gemäß den Formeln

$$f(x,n) = h(x,n,f(x,n\text{-}1)) \quad \text{für } n > 0 \tag{8.9a}$$

$$f(x,0) = f_0 = g(x) \quad \text{für } n = 0. \tag{8.9b}$$

berechnet. Dies ist eine in der Algorithmentheorie häufig benutzte Notation der Iteration.

Beim Multiplizieren bzw. Potenzieren durch Iteration geht (8.9) in (8.10) bzw. (8.11) über. Die Ausdrücke stellen die Rekursionsformeln zur Berechnung des Produktes $n*x$ bzw. der Potenz x^n dar.

$$f(x,n) = x + f(x,n\text{-}1)); \; f_0 = 0 \tag{8.10}$$

$$f(x,n) = x * f(x,n\text{-}1)); \; f_0 = 1 \tag{8.11}$$

Eine Funktion, die sich mit Hilfe des Inkrementierens, der Substitution, der Selektion und der Iteration definieren lässt, heißt ***primitiv-rekursive Funktion****. Primitiv-rekursive Funktionen sind* ***USB-Funktionen***. Die Richtigkeit des letzten Satzes ergibt sich aus der Tatsache, dass sich Substitution, Selektion und Iteration mittels der USB-Methode beschreiben lassen.

Minimalisierung

Zunächst glaubte man, mit den primitiv-rekursiven Funktionen alle berechenbaren Funktionen erfasst zu haben. Diese Annahme stellte sich jedoch als Irrtum heraus, als es gelang, Funktionen zu konstruieren, die berechenbar, aber nicht primitiv-rekursiv sind. Das trifft z.B. für die Ackermann-Funktion[15] zu. Daraufhin suchte man nach einer Erweiterung der konstruktiven Funktionsdefinition durch Hinzunahme einer zusätzlichen Komponierungsregel und fand diese in der sog. *Minimalisierung*. Auch sie wird in der Algorithmentheorie i.Allg. nicht als Komponierungsregel, sondern als Operator (Funktion) aufgefasst.

Das Wort *Minimalisierung* bedeutet (im Zusammenhang mit der Definition rekursiver Funktionen) "*Bestimmung der minimalen Iterationszahl*". Minimalisierung ist anzuwenden, wenn n_{max} nicht vorgegeben ist, sondern sich erst während der Iteration ergibt. Der Sachverhalt lässt sich an der Vorgehensweise des ABC-Schützen aus Kap.7.2 [7.6] veranschaulichen, der eine Differenz, z.B. 8 minus 3 dadurch bildet, dass er den Subtrahend 3 so oft um 1 erhöht, bis er den Minuend 8 erreicht, wobei er die Anzahl der Inkrementierungen an den Fingern mitzählt. Der ABC-Schütze führt also eine Iteration *ohne* vorgegebene Iterationszahl durch. Er bricht die Iteration ab, sobald der Minuend *m* gleich dem erhöhten Subtrahenden wird, also sobald das

15 Siehe z.B. [Duden 89], [Werner 95].

Prädikat [*m = erhöhter Subtrahend*] erfüllt ist, m.a.W. sobald die charakteristische Funktion, *d.h. die Differenz* (*m* minus *erhöhter Subtrahend*), zu Null wird.

Analog kann das Dividieren ganzer Zahlen auf iteratives Addieren zurückgeführt werden. Wenn aber der Dividend kein Vielfaches des Divisors ist, kann das Resultat der wiederholten Addition niemals gleich dem Dividenden werden und der Iterationsprozess findet kein Ende. Damit er abbricht, muss das Prädikat lauten "Das Iterationsergebnis ist gleich oder größer als der Dividend". Der Iterationsprozess wird bei dem frühesten Iterationsergebnis (bei der *minimalen* Iterationszahl) abgebrochen, für welches das Prädikat erfüllt ist. Das Ergebnis ist ein aufgerundeter Quotient. Derjenige Operator, der die minimale Iterationszahl bestimmt, wird **μ-Operator** genannt.

Ein typisches Beispiel für die Anwendung des μ-Operators ist eine iterative Näherungsrechnung, die abgebrochen werden soll, sobald die Korrektur (absolut genommen) im laufenden Schritt eine vorgegebene Schranke nicht überschreitet. Man denke z.B. an die Berechnung der Sinusfunktion nach einer Näherungsformel (z.B. nach der Formel (15.5)). Auch in diesem Fall ist $n_{\max}$ nicht vorgegeben, sondern muss mittels μ-Operator aus dem Abbruchprädikat [$|f_n - f_{n\text{-}1}| \leq \varepsilon$] bestimmt werden. Mit ε ist die vorgegebene Schranke bezeichnet.

Damit der Steueroperator die Abbruchsituation erkennen kann, muss er außer der Schranke die Größen f_n und $f_{n\text{-}1}$ kennen. Sie müssen ihm über den rechten nach oben gerichteten gestrichelten Pfeil in Bild 8.9 übergeben werden. Der Steueroperator führt dann die Funktion eines μ-Operators aus, denn er bestimmt die minimale Iterationszahl, bei der die geforderte Bedingung erfüllt ist.

Die allgemeine Definition des μ-Operators lautet: *Ein Operator heißt* ***μ-Operator****, wenn er einem Prädikat P(x,n) den minimalen Wert von n zuordnet, für den P erfüllt ist, wobei n die Iterationzahl eines Iterationsprozesses darstellt und x ein Tupel beliebiger Werte sein kann, die während der Iteration bereits berechnet worden sind.* Die Operation des μ-Operators wird **Minimalisieren** genannt. Sie kann bei der Festlegung von Funktionen mittels Iteration zur Anwendung kommen.

Wir vereinbaren: *Eine Funktion, die sich mit Hilfe des Inkrementierens, der Substitution, der Iteration, der Selektion und der Minimalisierung definieren lässt* (wobei nicht alle Komponierungsmittel zum Einsatz kommen müssen), *heißt* ***μ-rekursive Funktion*** *oder kurz* ***rekursive Funktion***. Falls die Definition die Minimalisierung enthält, wird vorausgesetzt, dass das Prädikat *P(x,n)* entscheidbar ist. Damit
26 ergibt sich der folgende wichtige

Satz: Rekursive Funktionen können stets nach der USB-Methode komponiert werden, m.a.W. rekursive Funktionen sind USB-Funktionen.

Um zu beweisen, dass auch das Umgekehrte gilt, dass USB-Funktionen rekursive Funktionen sind, muss unter anderem gezeigt werden, dass sich die Komponierungsmittel der USB-Methode, also die Flussknoten, rekursiv beschreiben lassen. Für

starre Flussknoten und Sammelweichen ist das bereits geschehen. Dass auch Zweigeweichen rekursiv beschreibbar sind, zeigt folgende Überlegung.

Da ein Operator definitionsgemäß nur einen einzigen Ausgang besitzt, müssen die Äste einer Zweigeweiche in einem Operandenflussgraph (in dem Operatorennetz eines Kompositoperators) früher oder später zusammenlaufen. Dabei kann sich eine Alternativmasche oder eine Rückkopplungschleife ergeben. Beide lassen sich rekursiv beschreiben, die Alternativmasche mittels des steuerbaren Selektors (siehe Bild 8.8), die Rückkopplungsschleife mittels des rekursiven Iterators (siehe Bild 8.9).

Damit ist gezeigt, dass sämtliche Flussknoten rekursiv beschreibbar sind. Es ist 27
aber noch nicht gezeigt, dass USB-Funktionen immer rekursive Funktionen sind, denn in einem Operandenflussgraphen können sich sowohl Maschen als auch Schleifen überlappen, was rekursiv nicht beschreibbar ist. Auch Überlappungen von Schleifen mit Maschen sind möglich. Bild 8.10a zeigt als Beispiel zwei sich überlappende Alternativmaschen. Je nach Weichenstellung berechnet der Kompositoperator eine der rechts angegebenen Funktionen. Ein Operatorennetz, das keine Überlappungen enthält, heißt **wohlstrukturiert**. Die USB-Methode lässt *nichtwohlstrukturierte* Operatorennetze zu. Wir wollen uns überlegen, ob bzw. wie sich ein nichtwohlstrukturiertes Netz in ein wohlstrukturiertes überführen lässt. Das Nächstliegende ist, das Problem durch Duplizierung von Operatoren zu lösen, wofür Bild 8.10b ein Beispiel gibt. In das Operatorennetz von Bild 8.10a ist ein zweiter *h*-Operator so eingefügt, dass zwei sich nicht überlappende Maschen entstehen, der Kompositoperator aber trotzdem dieselben Funktionen berechnet, wie der ursprüngliche.

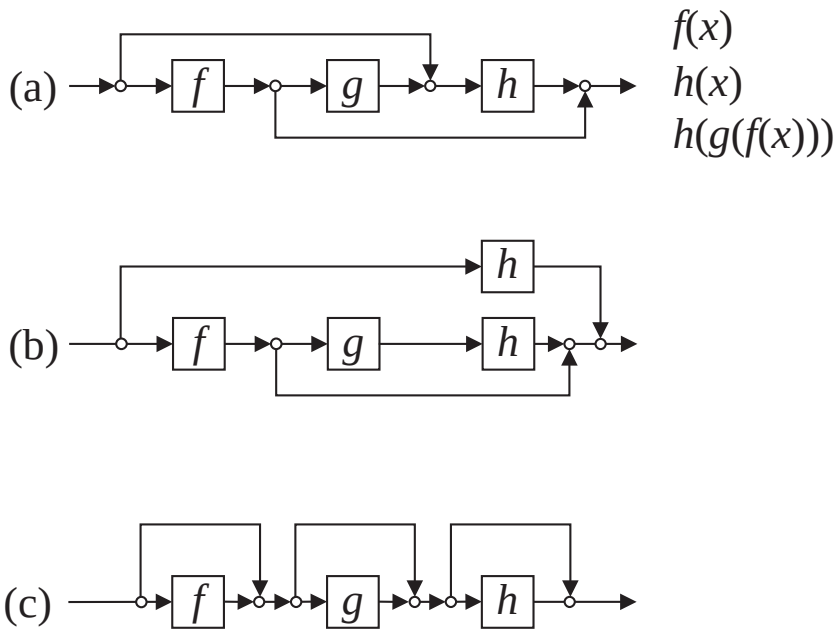

Bild 8.10 Wohlstrukturierung; **(a)** - Überlappung zweier Altenativmaschen; **(b)** - Wohlstrukturierung durch Operatorduplizierung; **(c)** - Wohlstrukturierung durch alternative Überbrückung

Bild 8.10c zeigt eine andere Methode der Wohlstrukturierung, die alternative Überbrückung. Sie besteht darin, dass die relevanten Operatoren durch einen Alternativzweig überbrückt werden. In dem Beispiel müssen alle drei Operatoren überbrückt werden. Dabei ergibt sich eine bedeutend höhere Variabilität des Kompositoperators. Da jeder der drei Bausteinoperatoren in den Kompositoperator einbezogen werden kann oder nicht, lassen sich 8 unterschiedliche Funktionen berechnen, darunter auch die identische Funktion $y = x$, was dem *Durchschleppen* des Eingabeoperanden durch den gesamten Kompositoperator entspricht.

Die beschriebenen Methoden der Wohlstrukturierung sind unabhängig davon
28 anwendbar, ob sich Maschen oder Schleifen überlappen. Daraus folgt, dass sich ***nichtwohlstrukturierte*** *Operatorennetze stets in* ***wohlstrukturierte*** *überführen lassen*. Es lässt sich also jeder Operandenlaufplan in eine Struktur überführen, die rekursiv beschreibbar ist. Wenn auch die Bausteinoperatoren (Bausteinoperationen) eines Operandenflussplanes rekursiv beschreibbar sind, dann beschreibt auch der Operandenflussplan eine rekursive Funktion. Daraus folgt der

29 **Satz:** Eine USB-Funktion ist eine rekursive Funktion, falls sie aus rekursiven Bausteinoperationen komponiert ist.

Bisher wurde eine einzige rekursive elementare Bausteinoperation verwendet, die Inkrementierung. In Kap.9.2.1 werden wir als elementare Operationen sog. boolesche Operationen wählen und damit das Fundament für die Komponierung des universellen Rechners legen. Die Wahl boolescher Operationen ist bedingt durch die Beschränkung auf binäre Codierung.

Abschließend sei bemerkt, dass die Begriffe der *partiellen* und der *totalen* Funktion, die im Zusammenhang mit der Turingmaschine eingeführt wurden, übernommen werden. Eine **partiell-rekursive Funktion** ist eine rekursiv beschriebene Funktion, deren Funktionswerte nicht für alle Werte einer vorgegebenen Argumentwertemenge existieren.

8.4.6 Einschub: Rekursives Berechnen

Wir unterbrechen die Behandlung der algorithmischen Systeme, um einige Erläuterungen zum Begriff der Rekursion und einige Überlegungen zum rekursiven Berechnen einzuschieben.

30 Das Wort “rekursiv” (zurücklaufend) bringt den zirkulären Charakter der Iteration zum Ausdruck, der sich in der Rückkopplungsschleife in Bild 8.9 und darin widerspiegelt, dass die Funktion f in (8.9a) von sich selber abhängt. Das erste f in der Gleichung bezeichnet einen (sprachlichen) Operator, das zweite einen Operanden. Es liegt ein Operand-Operator-Zirkel vor (siehe Kap.6.3).

Wir wollen uns überlegen, wie man bei der Berechnung eines Funktionswertes gemäß (8.9) vorzugehen hat. Will man z.B. die fünfte Potenz von 1,2 berechnen, wird man durch wiederholtes Multiplizieren mit 1,2 zuerst $1{,}2^2$, dann $1{,}2^3$ u.s.w. berechnen. Bei diesem Vorgehen wird der Exponent n schrittweise inkrementiert, insofern ist die Berechnung *aufwärts* gerichtet. Demgegenüber ist die allgemeine Rekursions-

formel (8.9) und ebenso die spezielle Rekursionsformel (8.11) für das Potenzieren *abwärts* gerichtet, denn zur Berechnung von f_n wird auf f_{n-1} *zurück*gegriffen. Wollte man auf diese Weise $1{,}2^5$ berechnen, müsste man mit $n = 5$ beginnen, was sinnlos zu sein scheint.

Unter bestimmten Umständen ist das Abwärtsverfahren jedoch möglich, es kann die effektivere oder sogar einzige Möglichkeit sein. Hinsichtlich der Programmierungstechnik hat das Abwärtsverfahren den Vorteil, dass es häufig eine kompaktere Ausdrucksweise ermöglicht. Die Informatiker sprechen dann von **rekursiver Berechnung** und von **rekursivem Programmieren**. Das rekursive Berechnen soll am Beispiel der Fakultät demonstriert werden[16].

Das Produkt $1*2*3*... * n$ wird in der Mathematik abgekürzt $n!$ notiert und als "n Fakultät" gelesen. Das Ausrufungszeichen wird also als Operationssymbol verwendet. Die Fakultät-Funktion $f(n) = n!$ ist nur für ganze nichtnegative Zahlen definiert. Die folgenden Formeln sind zwei unterschiedliche Definitionen der Fakultät-Funktion:

$$f(n) = 1 * 2 * 3 * ... * n = n! \quad \text{oder} \tag{8.12}$$

$$f(n) = n * (n - 1)! = n!; \quad 0! = 1. \tag{8.13}$$

In beiden Definitionen ist das linke Gleichheitszeichen als Ergibtzeichen, das rechte als Benennungszeichen (Definitionszeichen) aufzufassen. Wie unschwer zu erkennen ist, lässt sich die Definition (8.12) ohne weiteres in einen imperativen Algorithmus überführen, der folgende Berechnungsschritte vorschreibt:

a := 1*2
b := a*3
c := b*4
und so fort.

Es handelt sich um iteratives, nicht um rekursives Multiplizieren. Nach jedem Schritt muss geprüft werden, ob n erreicht ist.

Für die Definition (8.13) ist eine solche Überführung nicht möglich. Um sie als Operationsvorschrift benutzen zu können, muss man ihre Semantik, also das konkrete Vorgehen, das sie vorschreibt, verstehen. Die Semantik aber geht aus der Syntax der Formel nicht unmittelbar hervor. Sie ist durch "Vormachen" am anschaulichsten zu erklären. Dazu berechnen wir 5! rekursiv.

Nach (8.13) ist das Produkt $5 * 4!$ zu berechnen. Soweit ist die Semantik klar. 31
Doch wir können das Produkt nicht bilden, weil wir den Wert von 4! nicht kennen. Nun kommt die *interne Semantik* einer rekursiven Berechnung zum Tragen und es

16 In der Literatur wird die rekursive Beschreibung und Ausführung einer Operation häufig an einer bekannten Denksportaufgabe erläutert, dem "Turm von Hanoi", siehe z.B. [Duden 89].

beginnt der typische Prozess, der bei der Interpretation (Ausführung) der Definition (8.13) im Täger (Gehirn oder Computer) abläuft.

Wir notieren uns die 5 und berechnen 4! gemäß (8.13), wobei genauso verfahren wird, d.h. die 4 wird notiert und 3! berechnet. Der Prozess wird fortgesetzt, bis die Dekrementierung den Wert 0 liefert, für den der Wert der Fakultät als bekannt vorausgesetzt wird; es gilt 0!=1. Nun kann *n*! berechnet werden, indem die abgespeicherten Werte gemäß (8.11) iterativ miteinander multipliziert werden. Das Notieren der Zahlen, die zu multiplizieren sind, mag übertrieben erscheinen. Ein Computer dagegen muss sich *alles* notieren (abspeichern), da er immer nur diejenigen Werte "im Auge hat", mit denen er gerade eine Operation (z.B. eine Multiplikation) ausführt. Wenn jedoch die einzelnen Schritte kompliziertere Operationen beinhalten, kann auch bei rekursiven Rechnungen per Hand das Notieren zweckmäßig oder sogar notwendig sein.

In Bild 8.11 ist der wesentliche Teil eines Operatorennetzes dargestellt, das die beschriebene rekursive Berechnung der Fakultät durchführt. Der Operator op_1 führt die wiederholte Dekrementierung und op_2 die wiederholte Multiplikation aus. Das Netz zeichnet sich durch einen besonderen Speicher aus, einen sog. **Keller-** oder **Stapelspeicher**, auch **Stack** genannt (vom englischen stack = Stapel). In ihn können "von oben her" Daten eingespeichert (gekellert, gestapelt) und nach oben ausgelesen (**entkellert**) werden. Beim Kellern (Speichern) wird der bereits vorhandene Inhalt des Kellerspeichers um einen Speicherplatz (um ein Rechteck in Bild 8.11) nach unten verschoben. Beim Entkellern (Lesen) wird der Inhalt des obersten Platzes ausgegeben und der restliche Speicherinhalt um einen Platz nach oben verschoben.

In Bild 8.11 ist der Zustand des Kellerspeichers nach der zweiten Unterbrechung (Dekrementierung) bzw. vor der vorletzten Multiplikation dargestellt. Der Kellerspeicher kehrt die Reihenfolge, in der op_1 seine Resultate ausgibt, um, sodass op_2 die Zahlen in aufsteigender Reihenfolge multipliziert.

Um den Zusammenhang zwischen *rekursivem Berechnen* und *rekursiver Iteration* zu verdeutlichen, schreiben wir (8.13) um, indem wir die Fakultät-Funktion unspezifisch mit *f* und die Multiplikationsfunktion mit *h* bezeichnen. Dann geht (8.13) über in

$$f(n) = h(n,f(n\text{-}1)). \qquad (8.14)$$

Ein Vergleich mit (8.9) zeigt, dass (8.9) durch Streichung von *x* in (8.14) übergeht. Die Fakultät ist eine Iteration, in

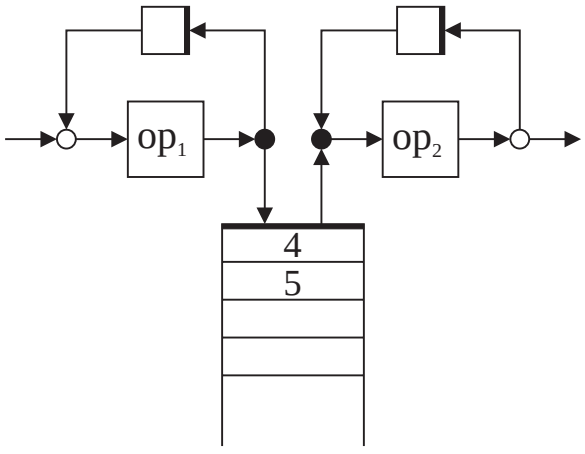

Bild 8.11 Operatorennetz zum rekursiven Berechnen mittels Kellerspeicher

die außer der Iterationszahl keine Variable eingeht.

Wie man sieht, ist rekursives Berechnen und rekursives Iterieren ein und dasselbe. Man beachte, dass sämtliche bisherigen Formeln für rekursives Berechnen oder Iterieren, also die Formeln (8.9), (8.10), (8.11), (8.13) und (8.14), funktional notiert sind. Das ist kein Zufall, sondern gilt offensichtlich allgemein. *Eine Vorschrift für* 32
eine rekursive Berechnung kann in Form einer Funktion, jedoch nicht in Form eines Befehls eines imperativen Programms notiert werden. Um sie imperativ zu notieren, muss sie in eine Befehlsfolge zerlegt und dadurch de facto in eine gewöhnliche (nichtrekursive) Iteration überführt werden. Die Formeln (8.13) und (8.12) und der nachfolgende imperative Algorithmus zur Berechnung der Fakultät gaben dafür ein Beispiel.

Abschließend soll die Beziehung zwischen rekursivem Berechnen und einem in der theoretischen Informatik häufig verwendeten Begriff angedeutet werden, dem Begriff des *Kellerautomaten*. Dazu fassen wir die Operatoren op_1 und op_2 von Bild 8.11 gedanklich zu einem Operator zusammen (Operatorenabstraktion), dessen Ausgang auf den Eingang zurückgekoppelt ist. In der Rückkopplungsschleife liegt der Kellerspeicher, der auch die Funktion der Verzögerungsglieder von Bild 8.11 übernimmt. Von den konkreten Operationen des rekursiven Ab- bzw- Aufstiegs ist abstrahiert worden. Das resultierende Operatorennetz ähnelt demjenigen des abstrakten Automaten von Bild 8.5. Wir ersetzen nun die Sammelweiche im Eingang durch eine Vereinung, sodass der entstandene Kompositoperator - ebenso wie der abstrakte Automat - in jedem Takt einen neuen Eingabewert empfangen und verarbeiten kann. Schließlich verallgemeinern wir den Kompositoperator von Bild 8.11 dahingehend, dass f jede Funktion sein kann und der *Folgefunktion* des abstrakten Automaten entspricht. Der resultierende Kompositoperator heißt **Kellerautomat** (siehe Bild 8.12).

Der Kellerautomat zeichnet sich dadurch aus, dass er in jedem Takt seinen Zustand kellern kann, ohne die vorangehenden Zustände zu löschen, und dass sein neuer Zustand nicht nur vom letzten Zustand abhängen kann, sondern dass "in den Keller hinabgestiegen und Vergangenes heraufgeholt" werden kann. Die Folgefunktion f kann also von mehreren, im Prinzip von unbeschränkt vielen früheren Zuständen abhängen. In Kap.16.5 werden wir auf die Bedeutung des Kellerspeichers für die Theorie formaler Sprachen zu sprechen kommen.

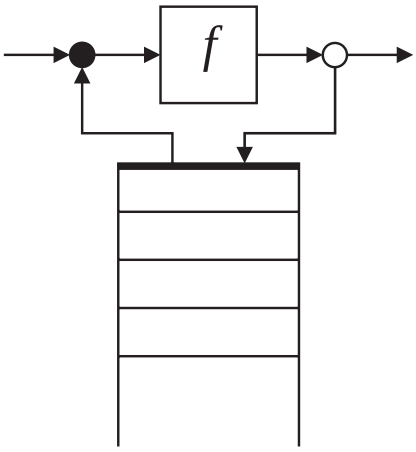

Bild 8.12 Kellerautomat; *f* Folgefunktion (vgl. Bild 8.5).

8.4.7 Lambda-Kalkül

Wir setzen nun die Besprechung universeller algorithmischer Systeme fort. In der Mitte der 30er Jahre entwickelte A.CHURCH seinen λ-Kalkül (Lambda-Kalkül). Auch er hatte sich die Aufgabe gestellt, den intuitiven Begriff der berechenbaren Funktion zu objektivieren. Dazu hat er zunächst die Zweideutigkeit der funktionalen Notation ausgemerzt (man erinnere sich an die Bemerkung am Ende der Behandlung der funktionalen Iteration [25]) und kam auf diesem Wege zu einer universellen Methode, neue Funktionen aus gegebenen Funktionen auf formalem Wege abzuleiten, sie zu "berechnen", wobei hier unter Berechnen *nicht* das Berechnen der Werte bereits definierter Funktionen zu verstehen ist, also *nicht* das *Ausführen* von Operationsvorschriften, sondern das Definieren selbst, das Artikulieren von Operationsvorschriften. In den bisher behandelten algorithmischen Systemen waren zwar Komponierungsregeln definiert, die Operationsvorschriften selbst waren jedoch verbal beschrieben. Church suchte nach einer *Methode zum* ***Berechnen****, zum* ***formalen Ableiten*** *funktionaler Operationsvorschriften im Rahmen eines Kalküls*.

Dieses Ziel lässt sich offenbar nur auf einer hohen Abstraktionsebene erreichen, auf der die Unterscheidung zwischen den Begriffen *Operator* und *Operand* nur noch relative Bedeutung hat (begriffliche Konvergenz). Nach der Intention von Church ist eine Operationsvorschrift, d.h. ein sprachlicher *Operator* gleichzeitig *Operand* desjenigen Operators, der ihn "*berechnet*", d.h. durch Rechnung artikuliert. Zu diesem Zweck entwickelte Church seinen Lambda-Kalkül.

33 *Der* ***Lambda-Kalkül*** *ist ein Kalkül zum Definieren von Funktionen durch* ***Berechnung*** *ihrer Zuordnungsvorschriften. Funktionen, die sich im Rahmen des Lambda-Kalküls definieren lassen, nennen wir* ***lambda-definierbare Funktionen*** *oder* ***Church-Funktionen***. Die zentrale Lösungsidee ist diese: *Notiere Funktionen (Operationen, Operatoren) funktional und komponiere Kompositfunktionen aus Bausteinfunktionen* ***allein durch Substitution***.

Funktionsdefinitionen nach dem Lambda-Kalkül beruhen also ebenso wie Funktionsdefinitionen nach der rekursiven Methode oder nach der USB-Methode auf der Rekursionsidee. Doch werden sie durch die Beschränkungen allein auf die Substitution (bei funktionaler Notation) *berechenbar* (*ableitbar* durch analytisches Rechnen). Nachträglich scheint diese Lösungsidee gar nicht so fern zu liegen. Denn auf ihr beruht die Ausdrucksstärke der funktionalen Notationsweise, wie sie in der Mathematik üblich ist. Außerdem stellt die Substitution das entscheidende Komponierungsmittel der rekursiven Definitionsmethode dar, denn auch die rekursive Iteration beruht auf Substitution (vgl. (8.9) und Bild 8.9). Und auch der Markovalgorithmus beruht auf Substitution.

Um das Vorgehen des Lambda-Kalküls verständlich zu machen, gehen wir wieder von unserer Funktion (8.1) aus. Zunächst vereinbaren wir eine leicht abgewandelte Form der funktionalen Notationsweise. Dazu schreiben wir die Funktion f_1 aus (8.1) in zwei verschiedenen funktionalen Notationen an:

$f_1 = +(\text{pot}(x,n),x)$ (8.15) 34

$f_1 = (+\ (\text{pot}\ x\ n)\ x)$ (8.16)

Die beiden Notationen unterscheiden sich durch eine geringfügige syntaktische Änderung voneinander. In (8.16) ist die öffnende Klammer dem Operationszeichen (dem Funktionsbezeichner) nicht wie üblich *nach*gestellt, sondern *voran*gestellt. Anstelle von $f(x)$ wird also $(f\,x)$ notiert, wobei f der Funktionsbezeichner ist. In diesem Sinne unterscheiden wir zwischen **Präfixnotation** (8.15) und **Listennotation** (8.16). Beide Notationsarten kommen in der Literatur und in Programmiersprachen zur Anwendung. In diesem Kapitel werden wir die Listennotation bevorzugen.

Damit die Ausdrücke (8.15) und (8.16) *ausgewertet* werden können (als Operationsvorschriften dienen können) muss das Addieren und Potenzieren ebenfalls im Rahmen des Lambda-Kalküls durch Substitution definiert werden. Das ist möglich, indem sie mittels rekursiver Iteration festgelegt werden. Das Resultat ist zwangsläufig funktional notiert.

Nehmen wir an, die Definition der Funktionen $(+\ x\ y)$ und $(\text{pot}\ x\ n)$ sei erfolgt. Die Bezeichnung der Argumente ist beliebig (ihre Codierung ist arbiträr). Dann stehen wir vor folgendem Problem. Wie kann man angeben, dass das erste Argument in der Additionsfunktion $(+\,x\,y)$ durch $(\text{pot}\,x\,n)$ und das zweite durch x zu substituieren ist? Wegen der Kommutativität der Addition ist diese eindeutige Zuordnung zwar nicht notwendig, doch soll die Methode auch auf nichtkommutative Operationen anwendbar sein. (Man beachte, dass nicht das Ergebnis der Substitution, sondern eine Vorschrift für die Substitution gesucht ist.) Die Zuordnung zwischen den Variablen und den substituierenden Funktionen ließe sich durch die Ergibtgleichungen $x := (\text{pot}\ x\ n)$ und $y := x$ festlegen. Das aber ist keine funktionale, sondern eine imperative Notation, also nicht zugelassen.

Church hat eine Methode entwickelt, nach der Substitutionen *funktional* und trotzdem exakt als Klammerausdrücke notiert werden. Sie soll in Anlehnung an die Bezeichnungsweise von Church erläutert werden. Wir notieren einen zunächst unverständlichen Ausdruck und erklären anschließend, was er bedeutet.

$(\lambda x.\ +\ x\ 3)2 =: (+\ 2\ 3) =: 5.$ (8.17)

Die Zeichenfolge $(\lambda x.$ kennzeichnet x als das *zu substituierende* Zeichen in dem 35
nachfolgenden Ausdruck, der zwischen dem Punkt und der schließenden Klammer steht und als "x+3" zu lesen ist. Man sagt auch, dass x durch $\lambda x.$ in dem Ausdruck x+3 *gebunden* wird, und nennt x **gebundene Variable**. Nichtgebundene Variablen heißen **freie Variablen**. Die Ziffer 2 nach der schließenden Klammer gibt an, dass die gebundene Variable durch 2 zu substituieren ist. Bis zu dieser 2 einschließlich handelt es sich um einen Ausdruck des Lambda-Kalküls.

Die umgekehrten Ergibtzeichen gehören nicht zum Alphabet des Kalküls, sondern veranschaulichen die beiden Schritte *Substitution* und *Wertberechnung*, die bei der Interpretation (Ausführung) des gesamten vorangehenden Ausdrucks durchzuführen

sind. Auf das erste Ergibtzeichen folgt das Ergebnis der Substitution von *x* durch 3 und auf das zweite das Ergebnis der Wertberechnung. Beide Operationen gemeinsam, die Substitution und die Wertberechnung, werden **Auswertung** oder **Evaluierung** genannt, die erste der beiden Operationen, die das λ zum Verschwinden bringt, wird auch als **Lambda-Eliminierung** bezeichnet.

Wir geben ein zweites Beispiel für einen Audruck des Lambda-Kalküls.

$$(\lambda y.(\lambda x.\ +\ x\ y)2)3 \tag{8.18}$$

Der Leser wird den Ausdruck sicher richtig interpretieren. Die Eliminierung des zweiten und anschließend des ersten λ ergibt (+ 2 3).

Der entscheidende Punkt ist nun, dass nach einer schließenden Klammer ein *beliebiger Ausdruck* stehen darf. Als Beispiel diene die Lösung des Problems, an dem wir oben gescheitert waren, als wir in der Additionsfunktion (+ *x y*) die Variable *x* durch (pot *x n*) substituieren wollten und dafür eine funktional notierte Vorschrift suchten. Jetzt kennen wir die Lösung:

$$(\lambda x.\ +\ x\ y)(\text{pot}\ x\ n).$$

Damit lautet die Definition (Bildungsvorschrift) der Funktion f_1

$$\lambda y.(\lambda x.\ +\ x\ y)(\text{pot}\ x\ n))x).$$

Mit der oben benutzten Wortverbindung “beliebiger Ausdruck” ist jeder im Sinne des Lambda-Kalküls *auswertbare* Ausdruck oder auch das Ergebnis jeder Auswertung gemeint. Jeder derartige Ausdruck wird **Term** genannt. Auch Variablen- und Konstantenbezeichner sind Terme. Damit nimmt die Substitutionsvorschrift folgende allgemeine Form an:

$$(\lambda x.\ U)V \tag{8.19}$$

Darin bezeichnen *U* und *V* Terme. (8.19) schreibt vor, dass *x überall, wo es in U auftritt*, durch *V* zu substituieren ist.

Der Ausdruck (8.19) kann als 3-stellige Funktion aufgefasst werden mit den Argumenten *x*, *U* und *V*. Doch ist (8.19) nicht in Listenform notiert. Eine Listennotation wäre beispielsweise

$$(\#\ x\ U\ V), \tag{8.20}$$

worin das Doppelkreuz als Bezeichner für die **Substitutionsfunktion** verwendet ist. In der Programmierungstechnik kommen noch andere Notationsweisen zur Anwendung. In der theoretischen Literatur hat sich die ursprüngliche Lambda-Notation erhalten, wobei i.Allg. nicht von Lambda-Funktion, sondern von **Lambda-Operator**

gesprochen wird. Das entspricht dem mathematischen Sprachgebrauch, wonach ein Operator einer Funktion eine Funktion zuordnet.

Damit der Lambda-Kalkül Aussicht hat, universell zu sein, muss er über ein 36
Pendant zur Zweigeweiche verfügen, eine "*Alternativ-Funktion*" oder "*bedingte Funktion*". Wir führen sie als 3-stellige Funktion mit dem Bezeichner "if" ein:

$$(\text{if } P\ U\ V) \tag{8.21}$$

Darin bezeichnet *P* ein Prädikat und *U* und *V* Terme. Wir nennen die Funktion (8.21) **if-Funktion**. Der Ausdruck (8.21) ist folgendermaßen zu lesen: "Falls das Prädikat *P* erfüllt ist, substituiere die if-Funktion (den gesamten Klammerausdruck) durch *U*, andernfalls durch *V*". Es kommen auch andere Funktionsbezeichner zur Anwendung, in der funktionalen Programmiersprache Lisp beispielsweise "cond" (von condition). Die in Kap.20.2 benutzte Programmiersprache CommonLisp verwendet die Syntax von (8.21). Damit kann die in (8.1) definierte Funktion $f(x,n)$ als Lambda-Ausdruck notiert werden:

$$(\lambda y.(\lambda x.\ +\ x\ y)(\text{pot } x\ n))(\text{if}(\leq x\ 0)\ x\ \text{sin}x).$$

Die Frage liegt nahe, ob in (8.20) der Term *x* eine Variable sein muss oder ob auch er - in Analogie zu (8.21) - ein beliebiger Term sein darf. Wäre das erlaubt, würde der Ausdruck (# *U V W*) bedeuten, dass in dem "Kompositterm" *V* der "Bausteinterm" *U* durch den Term *W* zu substituieren ist. Das sieht der Lambda-Kalkül nicht vor, denn seine Grundidee und die Rolle des Lambda-Operators ist das Binden einer *Variablen* in einem Ausdruck. Demgegenüber wäre der Ausdruck (# *U V W*) im Rahmen eines Markovalgorithmus durchaus sinnvoll und würde die Substitution von *V* durch *W* in *U* beschreiben, die erfolgen kann, falls in der Regelliste des Algorithmus die Regel $V \rightarrow W$ vorhanden ist.

Trotz dieses Unterschiedes ist eine deutliche Analogie zwischen der "*Termmanipulation*" des Lambda-Kalüls und der "*Wortmanipulation*" des Markovalgorithmus erkennbar. Das Wort "*Manipulation*" bringt zum Ausdruck, dass Zeichenketten mittels *Substitution* in andere überführt werden. Diese Analogie animiert zu einem umfassenderen Vergleich des Lambda-Kalküls mit anderen Methoden der Funktionsdefinition. Ein auffallender *Unterschied* zum Markovalgorithmus liegt darin, dass dieser die anzuwendenden Substitutionsregeln *vorgibt*. Die Regeln sind *Bestandteil* des Algorithmus, ähnlich wie im Falle der universellen Turingmaschine, wo die anzuwendende Automatentafel Bestandteil der Anfangsbeschriftung des Bandes ist.

Demgegenüber werden die Substitutionsregeln im Lambda-Kalkül nach den Regeln des Kalküls selbst *produziert*; es werden sowohl die *zu ersetzenden*, als auch die *ersetzenden* Zeichenketten (Terme) produziert, und zwar auch wieder durch selbstproduzierte Regeln. Es fragt sich, wo dieser Selbstproduktionsprozess seinen Anfang nimmt. Die Antwort lautet: In der sehr abstrakten rekursiven Definition des Termbegriffs. Um dem Leser eine Vorstellung vom Abstraktionsniveau zu geben,

auf dem Church gedacht hat, soll die Definition des Terms ohne Kommentar angegeben werden (in Anlehnung an [Hermes 71], S.209):

Jede Variable *x* ist ein Term; *x* kommt in *x* frei vor; keine andere Variable kommt in *x* frei vor. Sind T_1 und T_2 Terme, so ist (T_1T_2) ein Term; in diesem Term kommt eine Variable *x* frei vor, wenn *x* in T_1 oder in T_2 frei vorkommt. Ist *T* ein Term und *x* eine beliebige Variable, so ist $\lambda x.$T ein Term, in dem eine Variable *y* frei vorkommt, wenn *y* in *T* frei vorkommt und von *x* verschieden ist.

Auf den ersten Blick überrascht es, dass diese Definition zusammen mit den Regeln der Term-Manipulation gemäß (8.19) und (8.21) und einigen fast selbstverständlichen weiteren Regeln (wie Reflexivität, Symmetrie und Transitivität der Gleichheitsbeziehung $T_1=T_2$) ausreicht, um sämtliche effektiv berechenbaren Funktionen zu definieren. Es ist z.B. nicht ohne weiteres zu verstehen, wie durch den Lambda-Kalkül arithmetische Funktionen erfasst werden.

Überhaupt scheint unklar zu sein, wie man von der abstrakten, auf Termmanipulation beruhenden Funktionsdefinition zu einer Definition in Form einer Abbildung $X \rightarrow Y$ gelangen kann, insbesondere einer Abbildung in der Menge der ganzen Zahlen. Hierin liegt ein auffallender Unterschied zur rekursiven Methode und zur Registermaschine (URM-Methode). In allen vorangehenden Beschreibungsmethoden war die Definition einer Funktion als Abbildung kein Problem, denn es wurden stets Werte ("ausge*wert*ete Terme" in der Sprache des Lambda-Kalküls) in Werte transformiert. Im Lambda-Kalkül ist das nicht der Fall. Ein Term ist abstrakter, intensionaler Natur.

Angesichts des hohen Abstraktionsniveaus des Lambda-Kalküls hat es den Anschein, als ließe der Kalkül eine extensionale Definition einer Funktion als eindeutige Abbildung $X \rightarrow Y$ gar nicht zu. Es gibt keine aufzählbaren Mengen von Operanden, mit denen gerechnet werden könnte, kein Alphabet, keine Zahlen. All das muss nachträglich mit Hilfe der Regeln des Kalküls deduziert werden. Wie aber lassen sich die ganzen Zahlen, von denen die Registermaschine und die rekursive Definitionsmethode ausgehen, im Lambda-Kalkül einführen, d.h. als *Funktionen ableiten*?

Church erkannte, dass die ganzen positiven Zahlen in seinem Kalkül bereits implizit existieren und zwar als Iterationszahlen. Er definierte **Iterationsfunktionen** $\underline{0}$, $\underline{1}$, $\underline{2}$, $\underline{3}$ u.s.f. Um ihre Bedeutung (interne Semantik) zu verstehen, erinnere man sich an die Definition der Iteration in Kap.8.4.5 als Wiederholung ein und derselben Substitution, darstellbar durch eine Kette identischer Operatoren und notierbar als geschachtelte Funktion $f(f(f....(x)...))$ (vgl. (8.7) und Bild 8.7a mit $f_1=f_2=f_3=f$) oder in Listennotation $(f\ (f...(f\ x)...))$. Die Iterierte irgendeiner Funktion definieren wir nun als Iterationsfunktion und bezeichnen sie mit der Iterationszahl, die wir unterstreichen, um die Zahl als Funktionsnamen kenntlich zu machen. Danach ist beispielsweise die dritte Iterierte einer Funktion mit dem Namen *f* als $\underline{3}(f)$ bzw. $(\underline{3}\ f)$ zu notieren und die n-te Iterierte als $\underline{n}(f)$ bzw. $(\underline{n}\ f)$.

Es lassen sich nun neue Funktionen, auch 2-stellige Funktionen definieren, die bei Anwendung auf Iterationsfunktionen wieder Iterationsfunktionen liefern, und

zwar so, dass sich die Bezeichner (Nominalzahlen) der resultierenden Iterationsfunktionen als Resultat arithmetischer Operationen (z.B. der Addition) mit den Bezeichnern (Nominalzahlen) derjenigen Iterationsfunktionen interpretieren lassen, auf welche die neu definierte Operation angewandt wurde. Das berechtigt dazu, die Nominalzahlen als natürliche Zahlen zu interpretieren. Auf diese Weise konnte Church die arithmetischen Funktionen im Rahmen des Lambda-Kalküls definieren. Es konnte bewiesen werden, dass die lambda-definierbaren Funktionen genau die rekursiv definierbaren Funktionen sind.

Church hat sich den hohen Abstraktionsgrad durch die Aufgabe, die er sich gestellt hat, selbst auferlegt. Die Folge ist die Relativierung der Unterscheidung zwischen Operatoren (Operationen, Funktionen), Operanden und Werten (Funktionswerten). Ein Term kann sowohl ein Operator als auch ein Operand als auch ein Wert sein. Die Generalisierung des Operator-, Operanden- und Wertbegriffs zu einem einzigen abstrakten Oberbegriff ist ein wesentlicher Grund dafür, dass die Idee des Lambda-Kalküls sich als fruchtbar für die softwaremäßige Realisierung künstlicher Intelligenz erwiesen hat. Denn auch für das menschliche Denken ist charakteristisch, dass ein Idem, mit dem es hantiert, gleichzeitig als Operator (Subjekt) und als Operand (Objekt) auftritt. Das gilt speziell für das *Ich*-Idem, denn der Mensch sieht sich selbst ständig sowohl als *aktiven* als auch als *passiven* Teil der Welt.

Mit dieser etwas spekulativen Bemerkung beenden wir die Besprechung des Lambda-Kalküls. Ausführlichere Darstellungen findet der Leser in der einschlägigen Literatur[17].

8.5* CHURCH‘sche These und Kalkültransformation

Jede der sechs behandelten Methoden der Algorithmenbeschreibung definiert eine Klasse von Funktionen. Da den Methoden sehr unterschiedliche Ideen zugrunde liegen, sollte man erwarten, dass sie unterschiedliche Funktionsklassen definieren. Überraschenderweise ist das jedoch nicht der Fall. Die Algorithmentheoretiker konnten beweisen, dass nicht nur die genannten, sondern überhaupt sämtliche vorgeschlagenen algorithmischen Systeme ein und dieselbe Klasse von Funktionen festlegen. Die Funktionen werden, unabhängig von der Definitionsmethode, **rekursive Funktionen** genannt.

Auf Grund dieser Tatsache formulierte A.CHURCH die nach ihm benannte **churchsche These:** *Die Klasse der effektiv berechenbaren Funktionen ist mit der Klasse der rekursiven Funktionen identisch.* Jedes universelle algorithmische System, das entwickelt wurde oder das möglicherweise entwickelt werden wird, legt ein und dieselbe Klasse von Funktionen fest, die der rekursiven Funktionen. Nach den

17 Genannt seien [Hermes 71], [Louden 94], [Penrose 95].

Überlegungen in Kap.8.4.1 kann diese Aussage nur eine Hypothese sein. Doch hat sie sich ausnahmslos bestätigt, sodass sie als "praktisch nachgewiesen" gilt und sogar als richtige Voraussetzung in Beweisen herangezogen wird.

Die Tatsache, dass die verschiedenen Methoden die gleiche Klasse von Funktionen definieren, ist schon insofern überraschend, als sie von scheinbar ganz unterschiedlichen Objekten (Wertemengen) ausgehen. Die Operanden der Turingmaschine und des Markovalgorithmus sind Wörter über einem beliebig vorgebbaren Alphabet, die Operanden der Registermaschine und der rekursiven Funktionen sind die ganzen, nichtnegativen Zahlen, und die Operanden des Lambda-Kalküls sind abstrakte Objekte, Terme genannt.

Man könnte sich damit zufrieden geben, dass der Beweis der Identität der durch die verschiedenen Algorithmischen Systeme festgelegten Funktionsklassen von Autoritäten erbracht worden ist. So einfach wollen wir es uns jedoch nicht machen. Wir könnten den Beweis dadurch erbringen, dass wir nachweisen, dass nicht nur die Klasse der rekursiven Funktionen, sondern auch die anderen Funktionsklassen mit der Klasse der USB-Funktionen identisch sind. Im Weiteren werden wir die Identitäten zwar nicht exakt mathematisch beweisen, wir werden sie aber logisch plausibel machen.

Wir beginnen mit einer näheren Betrachtung des erwähnten Umstandes, dass in den verschiedenen algorithmischen Systemen Objekte ganz unterschiedlicher Natur die Rolle von Operanden spielen. Es kann z.B. rätselhaft erscheinen, dass die rekursiv berechenbaren Funktionen und die URM-berechenbaren Funktionen für natürliche Zahlen definiert sind, während nach der churchschen These auch Funktionen reeller Zahlen rekursive Funktionen sein müssen, wenn sie nur irgendwie berechenbar sind, z.B. durch einen Computer. Der aber hantiert ausschließlich mit Bitketten. Wie ist es dann zu verstehen, dass er rekursive Funktionen von reellen Veränderlichen berechnen kann?

37 Die Lösung des Rätsels ist nicht schwer zu erraten. Sie liegt in der Arbitrarität des Codierens (s. Kap.6.1) Die übliche Codierung reeller Zahlen ist ihre Darstellung als gebrochene Dezimalzahl, also als Kette arabischer Ziffern, die ein Komma enthält. Man kann auf das Komma als zusätzliches Zeichen verzichten, wenn ein für allemal ein fester Platz vereinbart wird, an dem das gedachte Komma zu stehen hat (sog. *Festkommadarstellung*). Wenn beispielsweise festgelegt wird, dass das Komma vor der dritten Ziffer von rechts steht, ist die reelle Zahl 3,51 durch die "ganze Zahl" 3510 darzustellen. Die Anführungsstriche sollen andeuten, dass die Interpretation der *Ziffernkette* 3510 als *ganze Zahl* im vorliegenden Kontext eine falsche semantische Belegung darstellt.

Der Platz des Kommas kann auch als Zehnerpotenz angegeben werden, z.B. $351 * 10^{-2}$. Wenn die Zehnerpotenz als Ziffernkette codiert wird, was natürlich möglich ist, ergibt sich wiederum eine "ganze Zahl" als Codewort für eine relle Zahl (sog. *Gleitkommadarstellung*). Auf die Codierung des Vorzeichens, die wir bisher unterschlagen haben, kommen wir sogleich zurück.

Eine weitere mögliche Codierungsmethode ist das *Durchnummerieren*, also das Zuweisen natürlicher (ganzer positiver) Zahlen an die reellen Zahlen durch Abzählen, m.a.W. die *Benennung* reeller Zahlen mit *Nominalzahlen*. (Nominalzahlen sind Ziffern oder Ziffernketten, die als Namen fungieren.) Das ist stets möglich, wenn davon ausgegangen wird, dass die durchzunummerierenden Mengen endliche oder abzählbar unendliche Mengen sind. Das Durchnummerieren legt folgende Methode für die Darstellung des Vorzeichens nahe. Angenommen, es sollen N positive und ebenso viele negative reelle Zahlen dargestellt werden. Es ergibt sich eine eindeutige Codierung, wenn die negativen reellen Zahlen mit den ganzen Zahlen von 1 bis N und die positiven mit den Zahlen von N+1 bis 2N benannt werden.

Um zu der üblichen rechnerinternen Zahlendarstellung zu gelangen, müssen die Ziffernketten in Bitketten umcodiert werden. Das kann z.B. dadurch erfolgen, das die Ziffernketten als Dezimalzahlen interpretiert und in Dualzahlen überführt werden, z.B. 0→0, 1→1, 2→10, 10→1010 oder durch irgendeine andere ein-eindeutige (in beiden Richtungen eindeutige) Abbildung. Die Umcodierung muss nicht zahlenweise (wortweise), sie kann auch ziffernweise (zeichenweise) erfolgen, indem jeder Ziffer eine vierstellige Bitkette zugewiesen wird, z.B. 0→0000, 1→0001 und folglich 10→00010000. Diese Methode liefert zwar längere Codeworte, hat aber ihre technischen Vorteile.

In der Rechentechnik ist eine Codewortlänge von 8 Bit üblich. Sie reicht aus, um alle **alphanumerischen** Zeichen (das sind die Ziffern und die Buchstaben) und alle **Sonderzeichen** (das sind zusätzliche Zeichen wie z.B. das Leerzeichen und Klammern) zu codieren. Eine solche Kette heißt **Byte**.

Die Codierung durch Nominalzahlen (mit anderen Worten die Abbildung in die Menge der natürlichen Zahlen) ist eine universelle Methode. Sie wird **Gödelisierung** und die codierenden Zahlen werden **Gödelnummern** genannt. Sie lässt sich auf jede abzählbare Menge anwenden, z.B. auf die Buchstaben eines beliebigen Alphabets oder auf die Wörter und Sätze jeder natürlichen oder künstlichen Sprache. Das bedeutet z.B., dass sich jedes Buch, ein Telefonbuch, eine Formelsammlung oder ein Roman, mittels natürlicher Zahlen codieren lässt. Es bedeutet aber auch, dass sich jeder Operand und die Wertetafel jeder Funktion mittels natürlicher Zahlen darstellen lässt.

Der letzte Satz wirft eine interessante Frage auf. Bedeutet die Umcodierbarkeit der Wertetafeln von Funktionen (Operatoren), dass die Funktionen (Operatoren) selber sich beliebig *umcodieren* lassen, z.B. mittels Nummerierung? Die Frage ist hinsichtlich der Operations*namen* offensichtlich zu bejahen, denn Namen sind arbiträr (Schall und Rauch). Aber was bedeutet es für die Operations*vorschriften*? Ein Beispiel soll die Sachlage illustrieren.

Viele Leser werden in der Schule gelernt haben, dass durch Logarithmieren eine Multiplikation in eine Addition, eine Division in eine Subtraktion und das Potenzieren in eine Multiplikation überführt wird. Es sei daran erinnert, dass der Dezimallog-

arithmus einer Zahl x, notiert als lg$x$, diejenige Zahl y ist, für die $10^y=x$ gilt. Daraus folgt beispielsweise, dass x_1*x_2 durch Logarithmieren in lgx_1+lgx_2 überführt wird:

$$\lg(x_1*x_2) = \lg x_1 + \lg x_2. \tag{8.22}$$

Das Resultat der Multiplikation ergibt sich durch Antilogarithmieren (die dem Logarithmieren inverse Operation) des Resultats der Addition.

Der Übergang zu Logarithmen führt zu einer ganz ähnlichen "Umcodierung" der Operanden, wie der Übergang von Festkommazahlen zu Gleitkommazahlen mittels Zehnerpotenzen, denn diese sind - neben den Mantissen - Bestandteil der Logarithmen. (Das Wort "Umcodierung" ist durchaus passend, denn beide "Codes" bezeichnen ein und dieselbe reelle Zahl.) Für die Umcodierung existiert eine *Vorschrift*, sodass der neue Code *berechnet* werden kann. Die Wahl der Umcodierungsvorschrift der Operanden (z.B. deren Logarithmieren) ist an sich beliebig. Doch ist mit der Wahl der Umcodierungsvorschrift für die Operanden gleichzeitig die neue Operationsvorschrift "ausgewählt", die sich hinter dem "umcodierten" Operationssymbol verbirgt. Operanden und Operatoren werden also gemeinsam umcodiert. Darum notieren wir die Umcodierung als mathematische Abbildung gemäß (8.23a). Das Ergebnis der Umcodierung ist durch einen Apostroph gekennzeichnet. Im Falle des Logarithmierens ist z.B. $100'$ identisch mit 2 (denn es gilt $10^2=100$) und $*'$ ist identisch mit +.

$$(Od,Op) \rightarrow (Od',Op') \tag{8.23a}$$

$$(X,Y,F) \rightarrow (X', Y', F') \tag{8.23b}$$

$$(Od,Alg) \rightarrow (Od',Alg') \tag{8.23c}$$

Auf Grund der vorausgesetzten Identität der Begriffe Operation und Funktion[18] kann (8.23a) in (8.23b) umgeschrieben werden. Darin bezeichnet F eine Menge von Funktionen und X bzw. Y die Menge der Argument- bzw. Funktionswerte. Die Unterscheidung zwischen Argument- und Funktionswerten in (8.23b) ist im Grunde überflüssig und eine Konzession an die gewohnte Notationsweise.

In (8.23c) sind die Funktionen durch die Algorithmen ersetzt, nach denen die Funktionen berechnet werden. Die Abbildung (8.23c) kann entweder als Übergang von einem algorithmischen System in ein anderes oder als **Übersetzung** aus einer algorithmischen Sprache in eine andere aufgefasst werden. Mit *Alg* ist die Menge aller Agorithmen bezeichnet, die in dem betreffenden algorithmischen System, d.h. in der algorithmischen Sprache des Systems artikuliert werden können.

Inhaltlich sind die Zusammenfassungen (*Od*,*Op*), (*F*,*X*,*Y*) und (*Od*,*Alg*) einander äquivalent, und die drei Abbildungen in (8.23) bezeichnen im Grunde ein und dieselbe Zuordnungsoperation. Wir nennen sie **Kalkültransformation**. Die Be-

18 Es sei daran erinnert, dass nur *eindeutige* Operationen und *realisierbare* Funktionen betrachtet werden, sodass die Worte *Operation* und *Funktion* als Synonyme verwendet werden dürfen.

zeichnung ist gerechtfertig, denn jeder der 6 Klammerausdrücke in (8.23) stellt eine Menge von Transformationsregeln (z.B. Op) und von Ausdrücken der jeweiligen formalen Sprache (z.B. Od) dar. Die Pfeile stellen also Transformationen zwischen Kalkülen dar. Wir ziehen das Wort *Transformation* den Wörtern *Umcodierung* und *Übersetzung* vor, weil es in diesem Zusammenhang dem üblichen Sprachgebrauch besser entspricht.

Aus der inhaltlichen Äquivalenz der drei Abbildungen in (8.23), bzw. der in ihnen zusammengefassten Einzelzuordnungen, folgt die im ersten Augenblick sicher überraschende Feststellung, dass die Wörter bzw. Wortverbindungen *Kalkül, algorithmisches System* und *algorithmische Sprache* Synonyme sind. Tatsächlich haben sie auf einer Abstraktionsebene, an die man nicht unbedingt gewöhnt ist, ein und dieselbe Bedeutung. Auf dieser Abstraktionsebene ist auch das Wort Algebra ein Synonym mit den drei genannten Bezeichnungen. Ein Blick in die Literatur zeigt, dass die Zusammenfassung (*Od*,*Op*) als **Algebra**, zuweilen auch als **algebraische Struktur** bezeichnet wird.

Da die Bezeichnungen “Kalkül” und “Algorithmisches System” auf einem ausreichend hohen Abstraktionsniveau Synonyme sind, können statt der bisher benutzten Namen für die 6 behandelten algorithmischen Systeme mit gleichem Recht auch die folgenden Namen verwendet werden: Turing-*Kakül*, URM-*Kalkül*, Markov-*Kalkül*, rekursiver *Kalkül*, USB-*Kalkül* und Lambda-*Algorithmus*.

Historisch betrachtet haben sich die Begriffe Kalkül, Algebra, algorithmische 38
Sprache und algorithmisches System bei der Untersuchung recht unterschiedlicher Fragestellungen herausgebildet und hatten infolgedessen zunächst unterschiedliche Bedeutungen. Doch diese näherten sich im Laufe eines langen Abstraktionsprozesses einander an, bis sie schließlich in einen einzigen Begriff zusammenflossen. Dieser Prozess der begrifflichen Annäherung soll **begriffliche Konvergenz** genannt werden.

Der Prozess der begrifflichen Konvergenz ist charakteristisch für das menschliche Denken überhaupt. Darum sprechen wir vom **Konvergenzprinzip des Denkens**. Es besteht darin, dass unterschiedliche Denkobjekte auf einer ausreichend hohen Abstraktionsebene zusammenfließen und miteinander identisch werden können. Wir werden dem Konvergenzprinzip im Weiteren wiederholt begegnen. Es wirkt im Sinne semantischer Verdichtung.

Die lange Entstehungsgeschichte der inhaltlich nahestehenden Begriffe Algebra, Kalkül, algorithmische Sprache und algorithmisches System hat zur Folge, dass jeder der Begriffe seinen speziellen Kontext hat, in dem er vorwiegend verwendet wird, und seine speziellen semantischen Nuancen. Das hat seinerseits zur Folge, dass nicht selten über ein und dasselbe Problem in unterschiedlichen Terminologien, in unterschiedlichen “Sprachen” gesprochen wird. Um die Beziehungen zwischen den verwendeten Terminologien deutlicher zu erkennen, vergegenwärtigen wir uns noch einmal einige Definitionen. Dabei werden sich interessante und für unser Vorhaben, einen Computer zu bauen, wichtige Schlussfolgerungen ergeben.

In Kap.5.4. war der Kalkülbegriff folgendermaßen verbal bestimmt worden: *Ein **Kalkül** ist ein System von Regeln für das Deduzieren* (Schlussfolgern, Berechnen); *es besteht aus Syntaxregeln und Deduktionsregeln*. Formal wird ein Kalkül in zwei Schritten definiert. Im ersten Schritt wird eine *formale Sprache* mittels Syntaxregeln festgelegt. Die Syntaxregeln geben an, welche Zeichen - elementare Zeichen (Alphabetzeichen) und Kompositzeichen (Zeichenketten, Wörter, Sätze) - erlaubt sind (zur Sprache gehören). In einem zweiten Schritt werden den zunächst völlig sinnfreien Zeichen formale Bedeutungen (Semantik) zugeordnet. Dieser Schritt heißt **formale Interpretation**[19]. Durch Interpretation werden die Zeichen zu **Bezeichnern** von Operanden, Operatoren oder Komponierungsmitteln. Sätze der formalen Sprache werden zu Deduktionsregeln (Operationsvorschriften).

Durch die formale Interpretation wird die formale Sprache zur **Kalkülsprache**, d.h. zu derjenigen Sprache, in der jeder komponierbare Operator beschrieben werden kann, m.a.W., in der die Vorschrift zur Ausführung jeder im Rahmen des Kalküls komponierbaren Operation artikuliert werden kann. Im gängigen Sprachgebrauch der Algorithmentheorie heißt eine Kalkülsprache *algorithmische* Sprache, wenn sie die Reihenfolge der Ausführungen der Bausteinoperationen eindeutig festlegt. Diese Voraussetzung wird mehr und mehr aufgegeben. Darum ist es gerechtfertigt, die Bezeichnungen Kalkülsprache und algorithmische Sprache als Synonyme zu verwenden und zu sagen: *Eine **formale** Sprache wird durch formale Interpretation zu einer **algorithmischen** Sprache*. Die Semantik einer formal interpretierten Kalkülsprache (die "formalen Bedeutungen" der mit ihrer Hilfe artikulierbaren Wörter) war in Kap.5.4 als *formale Semantik* bezeichnet worden.

In Kap.8.4.1 hatten wir ein algorithmisches System (eine algorithmische Sprache) *universell* genannt, wenn in ihm (in ihr) für jede effektiv berechenbare Funktion ein Berechnungalgorithmus artikuliert werden kann. Gemäß der churchschen These darf in dieser Definition "jede effektiv berechenbare Funktion" durch "jede rekursive Funktion" ersetzt werden. In dem gleichen Sinne sprechen wir auch von **universellem Kalkül**. Die in Kap.8.4 betrachteten algorithmischen Systeme sind Beispiele für universelle Kalküle.

Eine Kalkültransformation, also eine Abbildung (8.23a), muss folgende Bedingung erfüllen: *Das Resultat der Ausführung einer transformierten Operation (bzw. das Resultat der Berechnung einer transformierten Funktion) muss die Transformierte des Resultats der Ausführung der ursprünglichen Operation (bzw. der Berechnung der ursprünglichen Funktion) sein*. In der üblichen funktionalen Präfixnotation nimmt dieser Satz folgende Form an:

$$(f(x))' = f'(x'). \tag{8.24a}$$

19 In Kap.5.4 war ein zweiter Interpretationsschritt hinzugefügt worden. In ihm werden die Objekte eines Kalküls als Objekte der Realität interpretiert, wodurch die *formale* Semantik mit einer *externen* Semantik verbunden wird. Die externe Interpretation wird hier außer Acht gelassen.

Sämtliche Zuordnungen, die in der Abbildung (8.23b) zusammengefasst sind, müssen (8.24a) erfüllen. Darin kann *x* ein Tupel sein. Wenn es z.B. ein Paar ist, notieren wir

$$(f(x_1,x_{2)})' = f'(x_1', x_2') \tag{8.24b}$$

Diese Bedingung ist im Falle des Logarithmierens erfüllt. Verwendet man statt des Funktionssymbols das Operatorsymbol und wendet die Infixnotation an, geht (8.24b) in

$$(x_1\ op\ x_2)' = x_1'\ op'\ x_2' \tag{8.24c}$$

über. Durch Vergleich erkennt man, dass die Bedingung (8.24c) die Verallgemeinerung von (8.22) ist.

Da in jedem universellen algorithmischen System Funktionen ein und derselben Klasse berechnet werden, nämlich rekursive Funktionen und *nur* diese, kann (8.24a) beim Übergang von einem algorithmischen System in ein anderes für beliebige Funktionen durch geeignete Umcodierung erfüllt werden. Zwischen den beiden Systemen existiert also eine Abbildung (8.23b).

Diese Schlussfolgerung kann wegen der inhaltlichen Äquivalenz der Abbildungen in (8.23) auch folgendermaßen formuliert werden: *Universelle Kalküle lassen sich ineinander transformieren*. Da auch in nichtuniversellen Kalkülen gemäß der These von Church nur rekursive Funktionen berechnet werden können, wenn auch nicht sämtliche, ergibt sich der **Kalkültransformationssatz**: *Jeder Kalkül lässt sich in* 39
einen universellen Kalkül transformieren. Dieser Satz wird beim Entwurf eines universellen Rechners eine wichtige Rolle spielen.

Das Kapitel soll mit einer Bemerkung zur Entscheidbarkeit von Kalkülen beendet werden. Gödel hat die Frage untersucht, ob sich ein allgemeines Verfahren angeben 40
lässt, nach dem für jeden Satz eines Kalküls, z.B. der Arithmetik oder des Prädikatenkalküls, entschieden werden kann, ob der betreffende Satz wahr ist oder nicht (Problem der **formalen Entscheidbarkeit** mathematischer Systeme). Dabei hat er den Objekten, mit denen ein Kalkül hantiert (z.B. den arithmetischen Operationen), Nominalzahlen zugeordnet, er hat sie also durch natürliche Zahlen *codiert*. Dafür ist z.B. die oben erwähnte Methode der Durchnummerierung geeignet, vorausgesetzt die Eindeutigkeit ist gewährleitstet. Gödel hat sich für die Bestimmung der Codezahlen eine spezielle Vorschrift ausgedacht, die allen Forderungen der Eindeutigkeit genügt. Mit den Codezahlen kann man *rechnen* und so einen nichtarithmetischen Kalkül *arithmetisieren*. Auf diese Weise hat Gödel die Unentscheidbarkeit verschiedener Kalküle bewiesen, nachdem er die Unentscheidbarkeit der Arithmetik bewiesen hatte. Der *Unvollständigkeitssatz*, auf den in Kap.6.2 Bezug genommen wurde, ist wohl das berühmteste Resultat der gödelschen Arbeiten[20]. Wir sind wiederholt

20 Siehe z.B. [Gödel 31],[Hermes 71],[Hofstadter 85],[Schöning 95].

auf das Problem der Unentscheidbarkeit gestoßen, allerdings in anderen Zusammenhängen, in Kap.6.2 im Zusammenhang mit widersprüchlichen Zirkularitäten [6.1] und in Kap.8.3 im Zusammenhang mit Wahrsageprädikaten [21] und mit dem Halteproblem [22].

8.6 Vier Grundideen des elektronischen Rechnens

Wir wollen nun versuchen, aus den teilweise recht abstrakten Gedankengängen und Begriffsbildungen des Teils 1, insbesondere des Kapitels 8, konkrete Hinweise für die Teile 2 und 3 herauszulesen und *Richtlinien* für den Entwurf des "universellen Computers" und für die Realisierung künstlicher Intelligenz abzuleiten. An den
41 Anfang stellen wir eine sehr weitgesteckte, aber unscharfe Zielbestimmung. *Es soll ein Gerät entworfen werden, das in der Lage ist, jedes* ***folgerichtige Denken des gesunden Menschenverstandes*** *zu simulieren, sodass es als "Denkassistent" dienen kann*. Dies ist das "**Fernziel der KI-Forschung**".

Folgerichtiges Denken ist immer ein Deduzieren, also ein formales oder nichtformales Ableiten, es ist immer ein *regelbasiertes* Denken und verläuft im Rahmen und nach den Denkregeln irgendeines geeigneten *speziellen Denkkalküls*. Wenn der Computer jedes folgerichtige Denken simulieren soll, müssen sämtliche speziellen Denkkalküle realisiert werden. Angesichts der unterschiedlichen Ausprägungen menschlichen Denkens, m.a.W. angesichts der großen Zahl möglicher spezieller Denkkalküle ist es unmöglich, sie alle hardwaremäßig zu realisieren. Den Ausweg weist der *Kalkül-Transformationssatz*, nach dem sich jeder spezielle Kalkül in einen universellen Kalkül transformieren lässt. Daraus ergegben sich zwei Richtlinien für den Entwurf und den Bau von Computern.

Erste Richtlinie. Um einen Computer zu bauen, der beliebige *Rechnungen* ausführen kann, implementiere man einen *universellen* Kalkül und für jede spezielle formale Kalkülsprache (für jede spezielle "mathematische" Sprache) ein Übersetzerprogramm, das die spezielle Sprache in die Sprache des universellen Kalküls übersetzt.

Zweite Richtlinie. Um menschliches *Denken* zu simulieren, *kalkülisiere* man es und verfahre anschließend gemäß der ersten Richtlinie. Das *Kalkülisieren* besteht in erster Linie im Herausfinden und Formulieren derjenigen Regeln, nach welchen das Schlussfolgern stattfindet. Ob das immer möglich ist, bleibt im Augenblick dahingestellt. Auf eine schwerwiegende Konsequenz soll schon jetzt hingewiesen werden. Die zweite Richtlinie bindet die KI an das Kalkülisieren und damit an die Mathematik, denn das Erfinden von und das Hantieren mit Kalkülen ist Gegenstand der Mathematik, wie wir bereits in Kap.5.4 [5.13] festgestellt hatten. Da die KI selber keine mathematischen Methoden erfindet, kann sie sich nur in den Fußstapfen der Mathematik entwickeln. Nichtsdestoweniger leistet die KI (der Computer) einen

selbständigen Beitrag zur mathematischen Modellierung der Welt - ein überraschendes und faszinierendes Ergebnis der kulturellen Evolution (siehe Kap. 21.3 [21.3]).

Wenn es die beiden Richtlinien schon zu Beginn der Rechentechnik gegeben hätte, wäre die Entwicklung sicherlich schneller verlaufen. Heute lassen sie sich aus einer Analyse des "state of the art" ablesen. Für uns werden sie als Wegweiser dienen, wenn wir uns daranmachen, in Teil 2 den Computer und in Teil 3 die Methoden der KI nachzuerfinden. Verfolgt man aus heutiger Sicht den verschlungenen Weg, den die Entwicklung der elektronischen Rechentechnik gegangen ist, heben sich aus den unzähligen Ideen, die den Weg gebahnt haben, vier besonders fruchtbare und folgenreiche Ideen heraus. Wir nennen sie die **vier Grundideen des elektronischen Rechnens**.

- **Die erste Grundidee** ist die Wahl des booleschen Kalküls (der boolesche 42
  Algebra) als hardwaremäßig zu realisierenden Kalkül (Kap.9.2).
- **Die zweite Grundidee** ist die Realisierung der elementaren booleschen Operatoren mitHilfe elektrischer Schalter (Kap.10.1).
- **Die dritte Grundidee** ist die Programmierbarkeit von Bitkettenoperatoren (Kap.13.1).
- **Die vierte Grundidee** ist die Automatisierung des Übersetzens algorithmischer Sprachen in die Maschinensprache eines Computers (Kap.16.4).

Der Begriff der Maschinensprache wird in Kap.13.5.2 eingeführt. Wir geben hier schon eine vorläufige Definition. *Ein Programm, das ein Computer unmittelbar, d.h. ohne vorherige Übersetzung interpretieren (verstehen und ausführen) kann, heißt* ***Maschinenprogramm*** *des Computers. Eine Sprache speziell für die Artikulierung von Maschinenprogrammen heißt* ***Maschinensprache*** *des betreffenden Computers. Die Gesamtheit der syntaktischen Regeln einer Maschinensprache und der semantischen Regeln, nach denen ein Computer seine Maschinensprache interpretiert, bilden ein Kalkül. Wir nennen ihn* ***Maschinenkalkül***.

Ein Maschinenprogramm wird dadurch interpretiert (ausgeführt), dass den Operanden *innere Zustände* des ausführenden Computers als *codierende Zustände* und den Operatoren *Zustandsänderungen* zugeordnet werden. Innere Zustände sind stabile elektromagnetische Zustände elektronischer Bauelemente. Die Interpretation eines Programms, d.h. die ihm zugeordnete Semantik, *ist die Wirkung* des Programms im Computer. Allgemein hatten wir die Wirkung, die eine Zeichenkette in einem informationellen System auslöst, *interne Semantik* genannt (Kap.5.4 [5.7]). Ist der Träger ein Computer, sprechen wir von *maschineninterner Semantik* oder kurz **Maschinensemantik** oder **Computersemantik**.

Wenn ein Computer über ein Programm verfügt, das die Übersetzung einer Sprache in die eigene Maschinensprache ausführt, sagt man, dass die Sprache auf dem Computer **implementiert** ist und dass der Computer die Sprache "versteht". Eine Sprache wird durch ihre Implementierung zu einer **Programmiersprache** des Computers. In ihr können Aufträge an den Computer formuliert werden.

Wenn beispielsweise die Sprache der Arithmetik implementiert ist, kann der Computer arithmetische Ausdrücke auswerten. Wenn die Formelsprache für chemische Verbindungen implementiert ist und auch die Regeln, nach denen Ausdrücke der Sprache transformiert werden (d.h. chemische Prozesse stattfinden), kann der Computer aus den Eigenschaften von Atomen mögliche Strukturformeln von Molekülen und aus diesen mögliche oder hypothetische chemische Reaktionen herleiten, wofür Kapitel 16.3 ein Beispiel bringt. Dort wird die chemische Formelsprache als Kalkülsprache, d.h. als algorithmische Sprache, verwendet. Das kann Verwunderung hervorrufen, wenn unter chemischer Formelsprache die "Umgangsfachsprache" der Chemiker verstanden wird. Diese muss erst zu einer algorithmischen Sprache "hochstilisiert", d.h. sie muss kalkülisiert werden, bevor sie implementiert werden kann.

Dem Problem der Mathematisierung natürlicher Sprachen haben sich Mathematiker vieler Jahrhunderte gewidmet (siehe Kap.11.1). Vor ihm stehen heute die Informatiker, insbesondere die "KI-Forscher". Eine sehr allgemeine Lösung stellt aus mathematischer Sicht der *Prädikatenkalkül* und aus programmierungstechnischer Sicht die *logische Programmierung* dar. Das Kernproblem liegt in der Anbindung der externen Semantik natürlichsprachlicher Ausdrücke an die interne Semantik maschinensprachlicher Ausdrücke. Dies ist eines der zentralen Probleme der künstlichen Intelligenz. Wir hatten es in Kap.5.4 das *technische Semantikproblem* genannt (vgl.Kap.15.5).

Wenn ein Mensch dem Computer einen Auftrag oder eine Frage stellt, die er selber mit derjenigen Semantik belegt, in der er denkt (wir nennen sie **Nutzersemantik**; sie kann extern oder formal sein), dann muss die in die Maschinensprache übersetzte Frage im Computer genau diejenige Informationsverarbeitung *bewirken*, die der Nutzer bewirken wollte, und der Nutzer muss die Ausgabe (die Antwort) des
43 Computers interpretieren, d.h. ihr Nutzersemantik zuordnen können. Dabei muss sich die "*zu erwartende*" Antwort ergeben, also diejenige, die der Nutzer erhalten würde, wenn er sie selber ableiteten würden ohne dabei Fehler zumachen. In diesem Sinne sprechen wir von **Erhaltung der Nutzersemantik**.

Es mag zweifelhaft erscheinen, dass es überhaupt möglich ist, die Nutzersemantik zu erhalten. Denn für den Computer existiert sie nicht. Die Zeichenketten, die der Computer verarbeitet, besitzen für ihn weder formale noch externe Semantik, die sich auf die Außenwelt, die Welt des Nutzers bezieht. Der Computer "denkt" nicht in der Semantik, in welcher der Nutzer denkt. Insofern tragen die Zeichenketten, die der Computer empfängt, verarbeitet und ausgibt, nicht diejenige Bedeutung, die sie für den Nutzer tragen. Dies ist der Inhalt des *Bedeutungsprinzips*, das im Vorwort formuliert wurde. Nichtsdestoweniger muss die Bedeutung (die Nutzersemantik) erhalten bleiben.

Wir wollen das Problem der Semantikerhaltung etwas genauer analysieren. Dazu nehmen wir an, dass der Nutzer ein Physiker ist, der die Welt mit seinen Methoden modelliert und dabei auch den Computer nutzt. Der Modellierungsprozess, also das Ableiten neuer Aussagen über die Natur, erfolgt grob gesagt in 6 Schritten:

1. Messen,
2. Interpretation eines Kalküls durch die Messergebnisse,
3. Rechnen,
4. Programmieren,
5. Computerlauf,
6. Interpretation der Ausgaben des Computers.

Im zweiten Schritt wird externe Semantik an die formale Semantik eines Kalküls, sehr oft an die formale Semantik von Differenzialgleichungen angebunden. Diese muss bis zur Computerausgabe erhalten bleiben. Das bedeutet, dass in den Schritten 3 bis 5 keine formalen Fehler auftreten dürfen; der Physiker darf sich nicht verrechnen; mittels Kalkültransformationen muss die formale Semantik der Formeln (der Sprache des Physikers) in die formale Semantik der verwendeten Programmiersprache und weiter in die formale Semantik der Maschinensprache und in die interne Semantik des Computers fehlerfrei überführt werden; und schließlich darf der Computer sich nicht verrechnen. Im letzten Schritt wird an die Computerausgaben externe Semantik angebunden.

Für die Semantikerhaltung in den Schritten 1, 2, 3 und 6 ist der Physiker (allgemein der Computernutzer) verantwortlich. Für die Erhaltung der formalen Semantik in den Schritten 4 und 5 sind die Computerbauer, die Sprachentwickler und die Programmierer verantwortlich, die das Nutzerprogramm und das Übersetzerprogramm schreiben. In Kapitel 15 wird auf die Semantikerhaltung beim Übergang von mathematischen Formeln zur Maschinensprache und in Kapitel 13 beim Übergang von der Maschinensprache zu den Prozessen im Computer, d.h. zur Computersemantik eingengangen. Dabei müssen sprachlichen Ausdrücken elektronische Bauelemente bzw. deren Zustände zugeordnet werden, m.a.W. Softwareelemente müssen als Hardwareelemente *interpretiert* werden.

Viel Sorge und Mühe kann die Erhaltung der formalen Semantik beim Programmieren bereiten. Zur Sicherung der Semantikerhaltung kann ein altes, für andere Ziele entwickeltes Mittel herangezogen werden, die *Axiomatisierung*. Sie dient der Festlegung eines Kalküls durch ein sogenanntes Axiomensystem. Dieser Begriff war in Kap.5.4 eingeführt worden. Die Definition soll noch einmal wiederholt werden, wobei das Wort "*Regel*" durch "*Aussage*" ersetzt wird. *Das **Axiomensystem** eines Kalküls ist eine Menge von Aussagen, die voneinander unabhängig sind (d.h. keine Aussage des Axiomensystems ist aus den anderen ableitbar), die sich gegenseitig nicht widersprechen und aus denen sich sämtliche wahren Aussagen des Kalküls ableiten lassen.*

Das Aufstellen eines Axiomensystems für die Aussagen eines Wissensbereiches oder einer Sprache wird **Axiomatisierung** genannt. In Kap.5.4 war die Axiomatisierung der Geometrie durch EUKLID und der Mechanik durch NEWTON erwähnt worden. Wir werden auf die axiomatische Methode nicht näher eingehen und beschränken uns auf folgende Bemerkungen.

Eine Sprache heißt axiomatisiert, wenn ein Axiomensystem existiert, aus dem sämtliche Ausdrücke der Sprache und ihre formale Semantik abgeleitet werden können. Das bedeutet, dass nicht nur die Syntax, sondern auch die Semantik der Sprache formal definiert ist. Durch Axiomatisierung wird das Problem der Semantikerhaltung beim Übersetzen von Sprachen bzw. beim Transformieren von Kalkülen nicht aus der Welt geschafft, sondern auf das Transformieren (Übersetzen) zwischen Axiomensystemen reduziert, was erheblich übersichtlicher und infolgedessen weniger fehleranfällig ist.

In Teil 3 werden wir uns überlegen, ob bzw. wie das Fernziel der KI-Forschung erreicht werden kann. Dabei werden wir ständig auf die Begriffe, Methoden und Schlussfolgerungen dieses Kapitels zurückgreifen. Wir werden uns Methoden ausdenken, wie bestimmte Arten des Schlussfolgerns simuliert werden können, und uns so dem Fernziel nähern. Doch bleibt am Ende unseres Weges die Frage, ob das Fernziel erreichbar ist, offen. Die Etappen des Weges wurden durch obige Zuordnung der Grundideen des elektronischen Rechnens zu den einzelnen Kapiteln bereits angedeutet. Die Beziehungen zwischen dem Kapitel 8 und den folgenden Kapiteln sollen noch etwas detaillierter aufgezeigt werden.

Die Formulierung der vier Grundideen und die Angabe der Kapitel, in denen sie realisiert werden, hat beim Leser vielleicht das Verständnis dafür geweckt oder vertieft, warum in Kap.8.1 ein neues algorithmisches System, der USB-Kalkül eingeführt worden ist, obwohl es in theoretischer Hinsicht nichts Neues bringt. Die USB-Methode wird uns helfen, den universellen Rechner Schritt für Schritt durch hierarchisches Komponieren zu konzipieren. Dafür ist die Methode der uniformen Systembeschreibung von ihrer Idee her prädestiniert, denn wir haben sie ausdrücklich mit dem Ziel entwickelt, beliebige hierarchisch strukturierte Systeme adäquat beschreiben zu können.

Die USB-Methode entspricht der besonderen Herangehensweise dieses Buches an die Probleme der Berechnung und der Berechenbarkeit von Funktionen. Im Unterschied zur algorithmentheoretischen Fragestellung suchen wir - es sei noch einmal betont - *nicht* nach einem *algorithmischen*, sondern nach einem *technischen* System, nach einem *Gerät*. Für seine Beschreibung ist die USB-Methode eingeführt worden. Auch Turing hat nach einem Gerät gesucht, aber mit der Nebenbedingung maximaler Einfachheit und ohne das Ziel, das Gerät zu bauen. Wir werden in Kap.9.1 die Erfüllung einer anderen Bedingung fordern: die *binär-statische Codierbarkeit*. Unter dieser Bedingung wird das Problem der Berechenbarkeit in einem neuen Lichte erscheinen.

Bei der Konzeption des gesuchten universellen technischen informationellen Systems werden wir systematisch nach der USB-Methode vorgehen. Es ist also nicht verwunderlich, wenn uns die Begriffe der USB-Methode im Weiteren ständig begleiten werden. Aber auch die anderen algorithmischen Systeme werden wir nicht aus dem Auge verlieren. Die Sprachelemente verschiedener Programmiersprachen

werden uns an das eine oder andere algorithmische System, an den einen oder anderen Kalkül erinnern.

Als Abschluss unserer Überlegungen über das Codieren und das Transformieren (Umcodieren) von Kalkülen sei eine kleine philosophische Spekulation erlaubt. Letzten Endes ist es die Arbitrarität des Codierens, der wir die Universalität des Computers verdanken. Die Übertragung dieses Umstandes auf das Codieren durch den Menschen, das dem Denken und Sprechen zugrunde liegt, provoziert die Behauptung: *Die Arbitrarität des Codierens macht das Modellieren der Welt und speziell das mathematische (semantisch objektivierte) Modellieren möglich*. Aus der Sicht der Kapitel 1 und 5.1 müsste man eher - etwas paradox - sagen: *Die Arbitrarität des Codierens hat sich entwickelt, um die Welt modellierbar und sogar mathematisch modellierbar zu machen* (im Sinne einer teleonomischen Interpretation der Evolution).

Bevor wir das “theoretische” Kapitel 8 abschließen, sei noch einmal wiederholt, dass sein Inhalt *keine* Einführung in die mathematische Theorie der Algorithmen und der Berechenbarkeit darstellt. Die angestellten Überlegungen erheben nicht den Anspruch mathematischer Exaktheit. Sie sind eher Plausibilitätsbetrachtungen. Um die Methode der uniformen Systembeschreibung streng mathematisch auszuformulieren, muss weitere Arbeit geleistet werden, die über die Zielstellung dieses Buches, den Computer nachzuerfinden, hinausgeht. Der Verwirklichung dieser Zielstellung wenden wir uns nun zu. Der Weg, den wir gehen werden, ist abgesteckt.