

Dirk W. Hoffmann

# Einführung in die Informations- und Codierungstheorie

2. Auflage

---

## Einführung in die Informations- und Codierungstheorie

---

Dirk W. Hoffmann

# Einführung in die Informations- und Codierungstheorie

2. Auflage



Springer Vieweg

Dirk W. Hoffmann  
Fakultät für Informatik und Wirtschaftsinformatik  
Hochschule Karlsruhe – University of Applied Sciences  
Karlsruhe, Deutschland

ISBN 978-3-662-68523-5      ISBN 978-3-662-68524-2 (eBook)  
<https://doi.org/10.1007/978-3-662-68524-2>

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

© Springer-Verlag Berlin Heidelberg 2014, 2023

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

Die Wiedergabe von allgemein beschreibenden Bezeichnungen, Marken, Unternehmensnamen etc. in diesem Werk bedeutet nicht, dass diese frei durch jedermann benutzt werden dürfen. Die Berechtigung zur Benutzung unterliegt, auch ohne gesonderten Hinweis hierzu, den Regeln des Markenrechts. Die Rechte des jeweiligen Zeicheninhabers sind zu beachten.

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

Planung/Lektorat: Leonardo Milla  
Springer Vieweg ist ein Imprint der eingetragenen Gesellschaft Springer-Verlag GmbH, DE und ist ein Teil von Springer Nature.

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

Das Papier dieses Produkts ist recyclebar.

# Vorwort

---

Die Kommunikation ist von jeher ein Teil der menschlichen Kultur, und doch haben sich der Bezug und der Austausch von Informationen in den letzten hundert Jahren in einer Art und Weise verändert, die selbst kühne Visionäre nicht vorgesehen haben. Als Kinder des Informationszeitalters erachten wir flächendeckend verfügbare Radio- und Mobilfunknetze als genauso selbstverständlich wie das mobile Internet oder die satellitengestützte Verkehrsnavigation.

Lassen wir die Entwicklung der Kommunikationstechnik in ihrer zeitlichen Abfolge vor unserem geistigen Auge Revue passieren, so wird unsere Aufmerksamkeit zwangsläufig auf die Entwicklungen gelenkt, die in der Mitte des 20. Jahrhunderts stattgefunden haben. In jener Zeit verschmolz die Nachrichtentechnik, die als eine pure Ingenieurdisziplin begann, mit der damals aufkeimenden Informations- und Codierungstheorie. Es ist eine beeindruckende Symbiose von Ingenieurskunst und mathematischer Finesse, die im Ergebnis zu einer vollständigen Digitalisierung der Nachrichtenübertragung führte und den technologischen Fortschritt in ungeahnter Weise befeuerte. Es ist nicht übertrieben, wenn wir diese Symbiose als den Schlüssel bezeichnen, der uns die Tür in das Informationszeitalter öffnen ließ.

Die Gedanken, Konzepte und Methoden, die sich hinter der Informations- und Codierungstheorie verborgen, sind Inhalt dieses Buchs. In allen meinen bisher erschienenen Publikationen habe ich das Ziel verfolgt, den Stoff anwendungsorientiert und didaktisch ansprechend zu vermitteln, und auch dieses Mal möchte ich in diesem Punkt keine Ausnahme machen. Um eine enge Verzahnung zwischen Theorie und Praxis zu erreichen, wurden die theoretischen Ausführungen immer dann, wenn es mir möglich und sinnvoll erschien, durch Anwendungsbeispiele und Querbezüge ergänzt. Ferner habe ich die Lehrinhalte aller Kapitel durch zahlreiche Übungsaufgaben komplementiert, damit das Buch auch im Selbststudium eingesetzt werden kann.

Mittlerweile ist die *Einführung in die Informations- und Codierungstheorie* in der zweiten Auflage erschienen. Diese Gelegenheit möchte ich nutzen, um mich für die Hinweise und Verbesserungsmöglichkeiten zu bedanken, die ich von Leserinnen und Lesern der ersten Auflage erhalten habe. Namentlich erwähnen möchte ich Herrn Gernot Wistuba, der mich auf mehrere bisher unentdeckte Fehler aufmerksam gemacht hat.

Karlsruhe, im August 2023

Dirk W. Hoffmann

---

## Symbolwegweiser



Definition



Satz, Lemma, Korollar



Leichte Übungsaufgabe



Mittelschwere Übungsaufgabe



Schwere Übungsaufgabe

---

## Lösungen zu den Übungsaufgaben

In wenigen Schritten erhalten Sie die Lösungen zu den Übungsaufgaben:

1. Gehen Sie auf die Seite: [www.dirkwhoffmann.de/codierungstheorie](http://www.dirkwhoffmann.de/codierungstheorie)
2. Geben Sie einen der im Buch abgedruckten Webcodes ein.
3. Die Musterlösung wird als PDF-Dokument angezeigt.

# Übersicht

---

Das vorliegende Buch führt praxisnah in das Gebiet der Informations- und Codierungstheorie ein und orientiert sich dabei an den typischen Lehrinhalten, die an Hochschulen und Universitäten in den Bachelor- und Masterstudiengängen der Nachrichtentechnik, Informatik, Elektrotechnik und Informationstechnik im Hauptstudium vermittelt werden.

Kapitel 1 beginnt mit einem Rückblick auf die Geschichte der Nachrichtenübertragung. Dieser wird helfen, die in den nachfolgenden Kapiteln erarbeiteten Ergebnisse historisch einzuordnen und mit Leben zu füllen. Der geschichtlich uninteressierte Leser mag dieses Kapitel gefahrlos überspringen; es ist für das Verständnis der technischen Details nicht erforderlich.

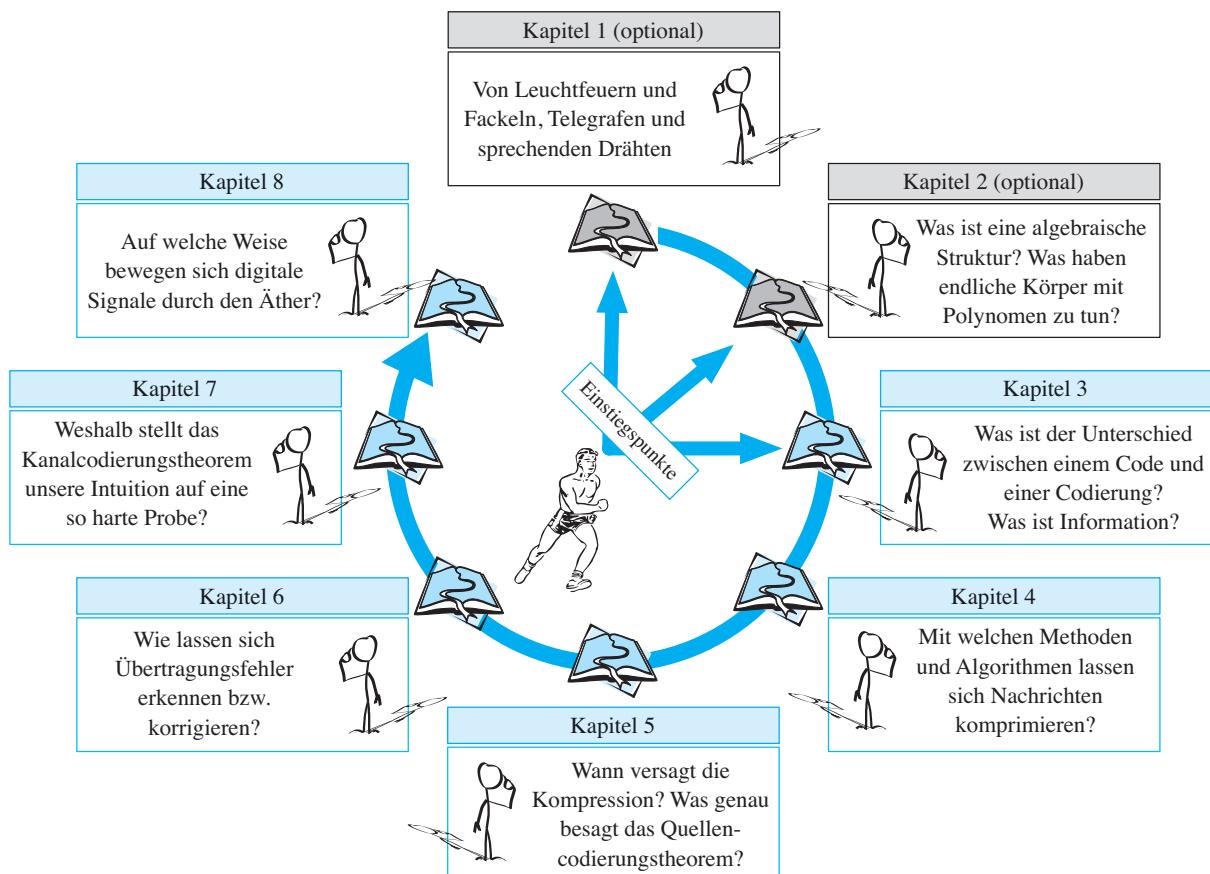
Kapitel 2 errichtet das mathematische Grundgerüst, auf dem wichtige Teile der Informations- und Codierungstheorie ruhen. Im Mittelpunkt stehen Begriffe aus der Algebra, die in den adressierten Studiengängen nicht immer zum Standardrepertoire gehören. Erwartet wird, dass der Leser mit den elementaren Begriffen und Methoden der Wahrscheinlichkeitsrechnung vertraut ist, ohne die ein tieferes Verständnis der Informations- und Codierungstheorie nicht möglich ist. „Elementar“ ist hier wörtlich gemeint: Ich habe versucht, das notwendige Wissen aus diesem Bereich auf das Nötigste zu begrenzen; die Verständlichkeit hat an den betreffenden Stellen Vorrang vor der mathematischen Stringenz.

Im dritten Kapitel stehen die Begriffe *Codierung* und *Information* im Mittelpunkt; beide werden dort ausführlich in ihren unterschiedlichen Facetten behandelt. Die Vorstellung des Informationsbegriffs orientiert sich dabei eng an der gedanklichen Linie von Claude Shannon, jenem Mathematiker, der mit seiner 1948 erschienenen Arbeit „*A mathematical theory of communication*“ die Informationstheorie ins Leben rief. Shannons Werk ist für die gesamte Kommunikationstechnik von so immenser Bedeutung, dass wir an zahlreichen Stellen darauf zu sprechen kommen werden.

Danach folgt das Buch dem klassischen Kanon, der die digitale Datenübertragung in eine *Quellencodierung*, eine *Kanalcodierung* und eine *Leitungscodierung* unterteilt. Die Quellencodierung umfasst alle

Methoden und Algorithmen, die sich mit der Kompression von Datenbeständen befassen, während die Kanalcodierung das Ziel verfolgt, eine komprimierte Nachricht so um zusätzliche Information anzureichern, dass der Empfänger Übertragungsfehler erkennen bzw. korrigieren kann. Die Leitungscodierung ist die letzte Stufe in der geschilderten Verarbeitungskette. Sie beantwortet die Frage, wie sich digitale Signale physikalisch über analoge Übertragungsmedien transportieren lassen.

## Kapitelübersicht



# Inhaltsverzeichnis

---

<b>1</b>	<b>Geschichte der Nachrichtentechnik</b>	<b>13</b>
1.1	Von Fackeln und Feuern der Antike . . . . .	15
1.1.1	Fackelpost des Agamemnon . . . . .	15
1.1.2	Synchrontelegraf des Aineias . . . . .	17
1.1.3	Fackelcode des Polybios . . . . .	18
1.2	Das mechanische Internet . . . . .	19
1.2.1	Semaphoren-Telegraf von Claude Chappe . . . . .	19
1.2.2	Klapptelegrafen . . . . .	23
1.3	Elektrische Nachrichtenübertragung . . . . .	26
1.3.1	Nadeltelegrafen . . . . .	29
1.3.2	Der Morse-Telegraf . . . . .	33
1.4	Mission Transatlantik . . . . .	40
1.5	Von der Telegrafie zur Telefonie . . . . .	45
1.5.1	Suche nach dem harmonischen Telegrafen . . . . .	46
1.5.2	Drahtlos durch den Äther . . . . .	56
1.6	Von der Röhre zum Supercomputer . . . . .	61
1.7	Informations- und Codierungstheorie . . . . .	65
1.8	Übungsaufgaben . . . . .	74
<b>2</b>	<b>Mathematische Grundlagen</b>	<b>81</b>
2.1	Motivation . . . . .	82
2.2	Modulare Arithmetik . . . . .	83
2.3	Algebraische Strukturen . . . . .	89
2.3.1	Gruppen . . . . .	89
2.3.2	Körper . . . . .	97
2.3.3	Ringe . . . . .	105
2.3.4	Ideale . . . . .	110
2.4	Endliche Körper . . . . .	111
2.4.1	Polynomringe . . . . .	111
2.4.2	Konstruktion endlicher Körper . . . . .	116
2.4.3	Schnelles Rechnen in endlichen Körpern . . . . .	123
2.5	Vektorräume . . . . .	136
2.5.1	Generatormatrizen . . . . .	147
2.5.2	Orthogonalräume . . . . .	148
2.6	Übungsaufgaben . . . . .	152
<b>3</b>	<b>Codierungen, Codes und Information</b>	<b>167</b>
3.1	Motivation . . . . .	168

3.2	Definition und Eigenschaften . . . . .	168
3.3	Längenvariable Codes . . . . .	171
3.3.1	Präfixfreie Codes . . . . .	171
3.3.2	Kraft'sche Ungleichung . . . . .	175
3.4	Blockcodes . . . . .	178
3.4.1	Zeichencodes . . . . .	181
3.4.2	Zahlencodes . . . . .	186
3.4.3	Lineare Codes . . . . .	188
3.5	Der Übertragungskanal . . . . .	198
3.5.1	Kanalkapazität . . . . .	199
3.5.2	Kapazität des Morse-Kanals . . . . .	200
3.6	Der Informationsbegriff . . . . .	203
3.7	Übungsaufgaben . . . . .	216
<b>4</b>	<b>Quellencodierung</b> . . . . .	<b>227</b>
4.1	Motivation . . . . .	228
4.2	Die Informationsquelle . . . . .	229
4.2.1	Gedächtnislose Quellen . . . . .	230
4.2.2	Markov-Quellen . . . . .	231
4.3	Datenkompression . . . . .	237
4.4	Entropiecodierungen . . . . .	242
4.4.1	Shannon-Codierung . . . . .	242
4.4.2	Fano-Codierung . . . . .	245
4.4.3	Huffman-Codierung . . . . .	247
4.4.4	Decodierung präfixfreier Codes . . . . .	251
4.5	Arithmetische Codierung . . . . .	254
4.6	ANS-Codierung . . . . .	260
4.6.1	rANS-Codierung . . . . .	261
4.6.2	uABS-Codierung . . . . .	271
4.7	Substitutionscodierungen . . . . .	274
4.7.1	Lempel-Ziv-77-Kompression . . . . .	275
4.7.2	LZSS-Kompression . . . . .	279
4.7.3	Lempel-Ziv-78-Kompression . . . . .	280
4.7.4	Lempel-Ziv-Welch-Kompression . . . . .	284
4.8	Burrows-Wheeler-Transformation . . . . .	288
4.8.1	Move-to-front-Codierung . . . . .	295
4.9	Übungsaufgaben . . . . .	297
<b>5</b>	<b>Grenzen der Quellencodierung</b> . . . . .	<b>315</b>
5.1	Motivation . . . . .	316
5.2	Entropie, Information, Redundanz . . . . .	317
5.3	Das Quellencodierungstheorem . . . . .	325
5.4	Blockweise Codierung . . . . .	330
5.5	Übungsaufgaben . . . . .	334

<b>6 Kanalcodierung</b>	<b>339</b>
6.1 Motivation . . . . .	340
6.2 Prüfziffercodes . . . . .	341
6.2.1 Erkennung von Einzelfehlern . . . . .	342
6.2.2 Erkennung von Vertauschungsfehlern . . . . .	344
6.2.3 Prüfziffercodes aus der Praxis . . . . .	346
6.3 Fehlererkennung und -korrektur . . . . .	353
6.3.1 Hamming-Distanz . . . . .	354
6.3.2 Code-Distanz . . . . .	356
6.4 Lineare Kanalcodes . . . . .	361
6.4.1 Syndromdecodierung . . . . .	361
6.4.2 Hamming-Codes . . . . .	370
6.4.3 Zyklische Codes . . . . .	377
6.4.3.1 Codierung . . . . .	382
6.4.3.2 Hardware-Implementierung . . . . .	389
6.4.4 BCH-Codes . . . . .	394
6.4.4.1 Vandermonde-Matrizen . . . . .	396
6.4.5 Reed-Solomon-Codes . . . . .	414
6.4.5.1 Codierung . . . . .	415
6.4.5.2 Decodierung . . . . .	423
6.4.5.3 Reed-Solomon-Codes unter der Lupe . . . . .	428
6.4.5.4 Cross-interleaved Reed-Solomon Code . . . . .	435
6.4.6 Hadamard-Codes . . . . .	445
6.4.7 Simplex-Codes . . . . .	456
6.4.8 Reed-Muller-Codes . . . . .	460
6.4.8.1 Reed-Muller-Codes erster Ordnung . . . . .	462
6.4.8.2 Reed-Muller-Codes höherer Ordnung . . . . .	466
6.5 Faltungscodes . . . . .	469
6.5.1 Viterbi-Algorithmus . . . . .	476
6.6 Übungsaufgaben . . . . .	482
<b>7 Grenzen der Kanalcodierung</b>	<b>509</b>
7.1 Motivation . . . . .	510
7.2 Was kostet die Fehlerkorrektur?	511
7.2.1 Singleton-Schranke . . . . .	511
7.2.2 MDS-Codes . . . . .	512
7.2.3 Perfekte Codes . . . . .	514
7.3 Golay-Codes . . . . .	522
7.3.1 Zyklischer Golay-Code . . . . .	524
7.3.2 Erweiterter Golay-Code . . . . .	529
7.3.3 Ternärer Golay-Code . . . . .	531
7.4 Restfehlerwahrscheinlichkeit . . . . .	533
7.4.1 Restfehler bei der Fehlererkennung . . . . .	533
7.4.2 Restfehler bei der Fehlerkorrektur . . . . .	537

7.5	Das Kanalcodierungstheorem . . . . .	541
7.5.1	Inhaltliche Aussage . . . . .	541
7.5.2	Beweisskizze . . . . .	545
7.6	Übungsaufgaben . . . . .	558
<b>8</b>	<b>Leitungscodierung und Modulation</b>	<b>565</b>
8.1	Motivation . . . . .	566
8.2	Leitungscodierungen . . . . .	567
8.2.1	Bitcodierungen . . . . .	569
8.2.2	Blockcodes . . . . .	574
8.2.2.1	MMS43-Codierung . . . . .	576
8.2.2.2	RLL-Codierungen . . . . .	578
8.2.3	Externe Resynchronisation . . . . .	582
8.2.3.1	Bit Stuffing . . . . .	582
8.2.3.2	Scrambler . . . . .	584
8.3	Modulationsverfahren . . . . .	585
8.3.1	Digitale Modulation . . . . .	586
8.3.2	Kombinierte Modulationsverfahren . . . . .	589
8.4	Multiplexverfahren . . . . .	590
8.4.1	Frequenzmultiplexverfahren . . . . .	590
8.4.2	Raummultiplexverfahren . . . . .	592
8.4.3	Zeitmultiplexverfahren . . . . .	593
8.4.4	Codemultiplexverfahren . . . . .	597
8.5	Spreizcodes . . . . .	602
8.5.1	OVSF-Codes . . . . .	602
8.5.2	Pseudozufallsfolgen . . . . .	609
8.5.3	Gold-Folgen . . . . .	614
8.5.4	Kasami-Folgen . . . . .	621
8.6	Übungsaufgaben . . . . .	626
	<b>Literaturverzeichnis</b>	<b>643</b>
	<b>Namensverzeichnis</b>	<b>647</b>
	<b>Sachwortverzeichnis</b>	<b>649</b>

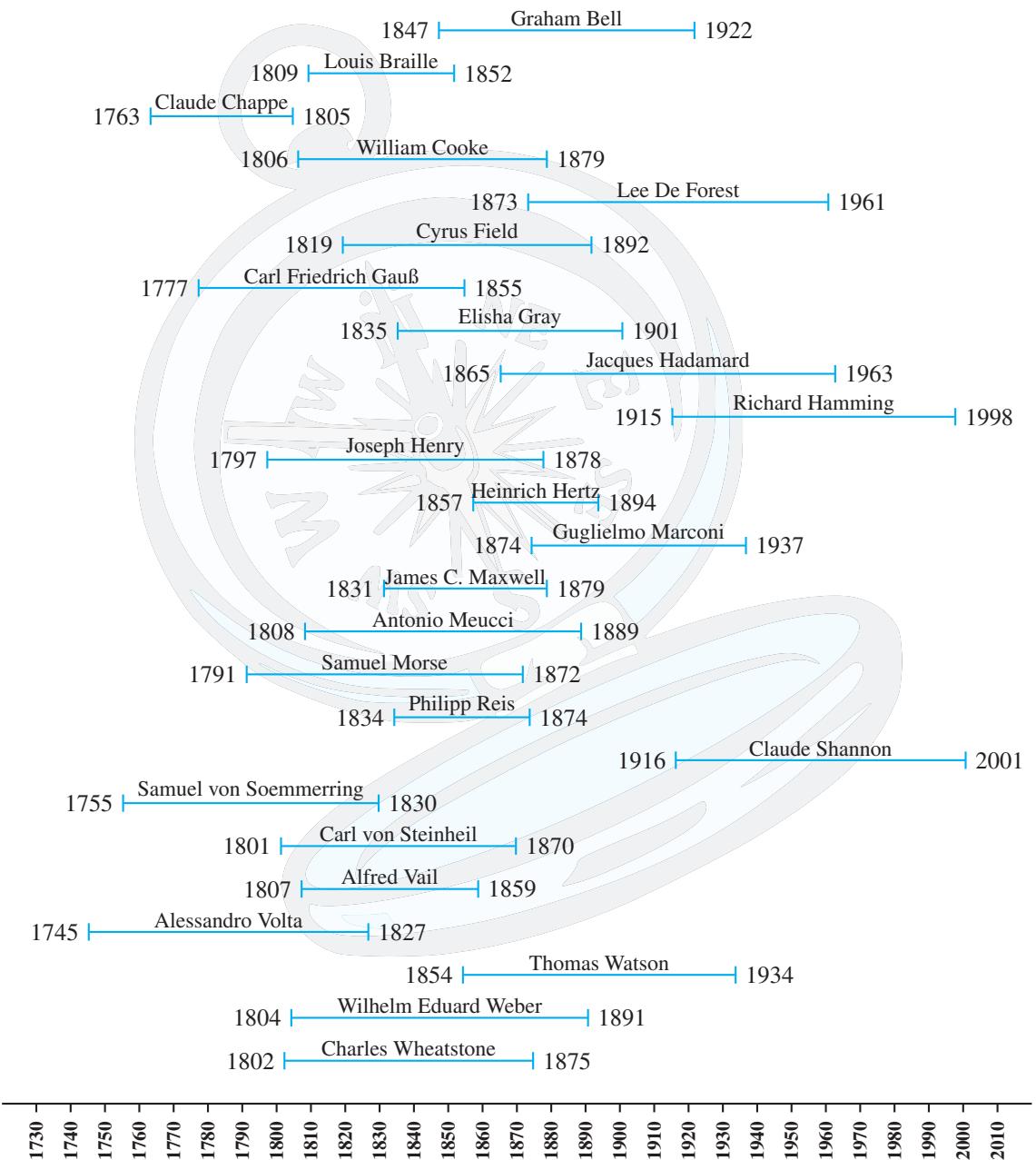
# 1 Geschichte der Nachrichtentechnik

---

In diesem Kapitel werden Sie ...

- eine Reise in die Geschichte der Nachrichtentechnik unternehmen,
- den Wandel einer ehemaligen Ingenieurdisziplin zu einer Wissenschaft begleiten,
- die Bedeutung der Informations- und Codierungstheorie verstehen.



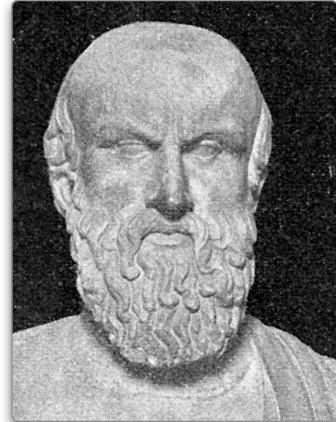


## 1.1 Von Fackeln und Feuern der Antike

### 1.1.1 Fackelpost des Agamemnon

Gebannt blickte er auf das Flackern, das seine müden Augen am Horizont erspähten. War es nur eine Illusion oder tatsächlich die Flamme, nach der er seit Jahren vergebens Ausschau hielt? Nacht für Nacht hatte er auf dem Dach des Königspalasts zu Argos zugebracht, oft vom Tau durchnässt, einem Wächterhund gleich. Geträumt hatte er schon lange nicht mehr, zu groß war seine Furcht, tief in den Schlaf zu entgleiten. Tatsächlich! Auf Arachnaions Gipfel loderte das ersehnte Feuer, das die sternenklare Nacht jetzt hell durchdrang. 10 Jahre waren nicht umsonst. Troja ist gefallen!

In etwa so beginnt das Drama *Agamemnon*, der erste Teil der *Orestie* des Dichters Aischylos (Abbildung 1.1). Ausgangspunkt der griechischen Tragödie ist die Eroberung Trojas (Abbildung 1.2). Der Mythologie zufolge hielt die Stadt einer zehnjährigen Belagerung stand, bis sie unter der Führung von Agamemnon, dem Oberbefehlshaber der griechischen Streitkräfte, endgültig eingenommen wurde. Das besagte Feuer auf dem Gipfel von Arachnaion war eines von insgesamt 8 Signalfeuern, die nacheinander entzündet wurden, um die Siegesnachricht an Agamemnons Frau Klytaimestra im königlichen Palast in Argos zu übermitteln (Abbildung 1.3).



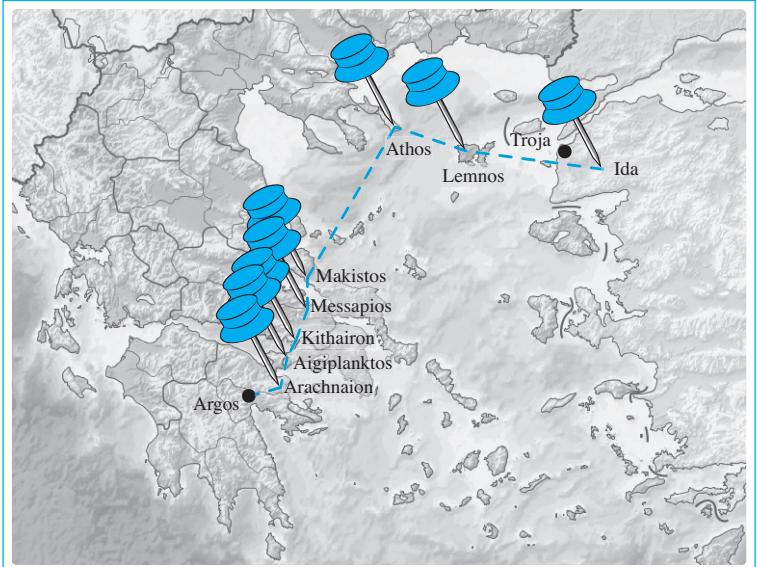
Aischylos  
(525 v. Chr. – 456 v. Chr.)

**Abb. 1.1:** Büste des Aischylos. Neben Sophokles und Euripides zählt Aischylos zu den drei großen Tragödiendichtern des antiken Griechenlands.

Hephaistos, der vom Ida hellen Strahl gesandt!  
Denn hergeschickt hat in der Feuer Wechselpost  
Ein Brand den andern. Ida selbst zum Hermesfels,  
In Lemnos; von der Insel her zum dritten nahm  
Den breiten Lichtstrahl auf des Zeus Athos gebirg.  
Hochleuchtend, daß der Wanderin Flamme mächtiger Schein  
Weithin der Meerflut Rücken überflog, ein Brand  
Der Freude, ward goldstrahlend, einer Sonne gleich,  
Zur Warte von Makistos dann das Licht gesandt.  
Die schürte weiter, säumig nicht noch unbedacht  
Vom Schlaf bewältigt, ihren Botenteil hinaus.  
Und wieder fernhin eilend gen Euripos' Flut  
Rief auf der Strahl die Wächter auf Messapios.  
Die dann entbrannten und entsandten neuen Schein,  
Der Graias Haufen Heidekraut anzündete.

Die rüstge Flamme, nicht ermüdet noch geschwächt,  
Sie eilte weithin über Asopos' Ebene,  
Gleich hellem Mondlicht, gen Kithairons Felsenstirn  
Und weckte schnell der Feuerboten Wechsel auf.  
Fernhin erkennbar neue Flamme schürte dort  
Die Wache; hoch schlug dann das hellste Feuer auf  
Und warf den Glanz weit über den Gorgopis-See.  
Auf Aigiplanktos' Scheitel treffend trieb es an,  
Des Fanales Lichtbahnen nicht zu stören; schnell geschah's;  
Sie sandten glutanschürend zu wolkenglühndem Schein  
Den mächtgen Schweif der Flamme, daß er fernhinaus  
Die weite Spiegelfläche des saronischen  
Meerbusens leuchtend überstrahlte, bis er kam  
Zu Arachnaions Gipfel nah bei unsrer Stadt.  
Von dort ergoß dies Feuer sich in dieses Schloß [...]

**Abb. 1.2:** Das Drama *Agamemnon* des griechischen Dichters Aischylos ist der erste Teil der *Orestie*, der einzigen erhaltenen Trilogie des antiken Griechenlands. Es enthält eine der ältesten Beschreibungen dessen, was wir heute als Fackelpost bezeichnen [1].



**Abb. 1.3:** Fackelpost des Agamemnon. Ausgangspunkt soll das östlich von Troja gelegenen Ida-Gebirge gewesen sein. Glauben wir der Beschreibung in der Orestie, so wurde das Signal über 7 weitere Relaisstationen bis nach Arachnaion übertragen. Von dort aus bestand eine direkte Sichtverbindung zum königlichen Palast in Argos.

Was wir in der Orestie nachlesen können, ist eine der ältesten Beschreibungen der *Fackelpost*. Es ist die primitivste Art der optischen Nachrichtenübertragung, bei der eine vorher festgelegte Botschaft durch das Abbrennen von Fackeln oder Holzstößen über längere Distanzen übermittelt wird.

Doch ist die Schilderung des Aischylos überhaupt realistisch? Wurde über den Fall Trojas tatsächlich auf dem beschriebenen Weg berichtet? Wahrscheinlich nicht. Geografisch wäre eine Übertragung auf der besagten Strecke möglich gewesen, denn alle von Aischylos beschriebenen Relaisstationen liegen tatsächlich an Orten, die über eine direkte Sichtlinie miteinander verbunden sind. Allerdings darf bezweifelt werden, dass die Griechen in Athos ein Feuer mit so hoher Lichtintensität entfachen konnten, dass es im 177 km entfernten Makistos immer noch mit bloßem Auge sichtbar war. Wir wissen heute, dass riesige Holzstöße notwendig gewesen wären, die es wohl nie gegeben hat [3]. Viel wahrscheinlicher ist, dass Aischylos seine Erzählung mit einem dramaturgischen Höhepunkt beginnen lassen wollte, der mit der Realität, wenn überhaupt, nur sehr entfernt zu tun hatte.

Dennoch ist Aischylos' Drama Agamemnon für die Nachrichtenübertragung von historischer Bedeutung. Es beweist, dass die Griechen mit dem Prinzip der Fackelpost 458 v. Chr., dem Jahr der Erstaufführung der Orestie, wohlvertraut waren.

In technischer Hinsicht markiert die Fackelpost den Beginn der modernen Nachrichtenübertragung, denn mit ihrer Hilfe war es erstmals möglich, Nachrichten über größere Distanzen mit hoher Geschwindigkeit zu kommunizieren. Im Drama Agamemnon erreichte Argos die Botschaft vom Fall Trojas in wenigen Stunden; die damals vorherrschende Nachrichtenübermittlung per Ross und Reiter hätte hingegen viele Tage benötigt. Natürlich war die Fackelpost nicht in der Lage, die herkömmlichen Übertragungswege zu ersetzen. Der Aufbau einer Signalstrecke war aufwendig und ohnehin nur unter günstigen geografischen Gegebenheiten möglich. Ihr grösster Nachteil war, dass keine differenzierten Nachrichten übertragen werden konnten. Ein Leuchtfeuer signalisierte immer nur das Eintreten eines einzigen, vorher verabredeten Ereignisses, und nicht mehr.

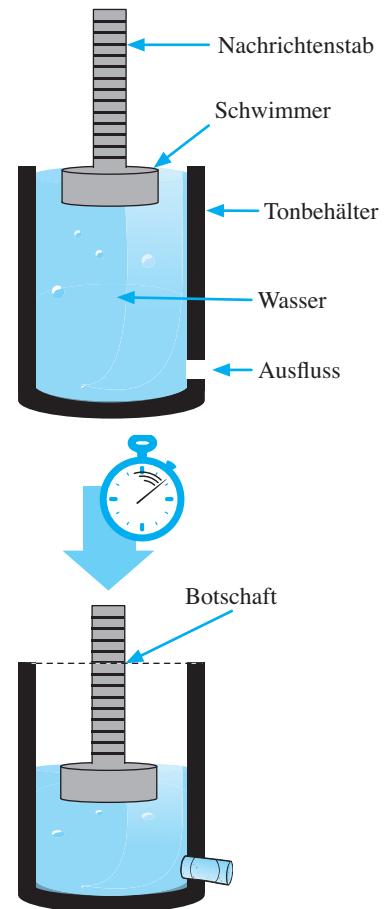
### 1.1.2 Synchrontelegraf des Aineias

Eine der ersten technischen Konstruktionen, die zur Übertragung differenzierter Nachrichten genutzt werden konnte, geht auf den Militärstrategen Aineias Taktikos zurück. Im 4. Jh. v. Chr. schlug der Grieche den Bau einer Apparatur vor, die wir heute als *Synchrontelegraf* bezeichnen (Abbildung 1.4).

Dass wir von Aineias' Erfindung heute überhaupt Kenntnis haben, verdanken wir dem griechischen Geschichtsschreiber Polybios von Megalopolis, der den Aufbau des Synchrontelegrafen im 2. Jh. v. Chr. in einer seiner Schriften erwähnt:

„[Aineias] sagt, dass sich diejenigen, die wichtige Nachrichten mithilfe von Feuerzeichen austauschen wollen, tönerne Gefäße beschaffen sollen, deren Durchmesser und Höhe auf das Genaueste gleich sein sollen, [...]. Wenn dies geschehen ist, sollen beide Gefäße sorgfältig angebohrt werden, sodass kleine Ausflüsse entstehen, durch die [Wasser] gleich schnell abfließen kann.“

Polybios von Megalopolis [4, 71]



**Abb. 1.4:** Aufbau des Synchrontelegrafen des Aineias, nach einer Erzählung des griechischen Geschichtsschreibers Polybios

Nach dieser Darstellung hatte Aineias zwei identische Tongefäße vorgesehen, die in etwa 1,5 m hoch und 0,5 m breit waren. Die Gefäße wurden mit Wasser gefüllt und besaßen am unteren Rand eine Ausflussoffnung. Ferner befand sich in beiden Gefäßen ein Schwimmer, auf dem ein Nachrichtenstab mit mehreren eingravierten Textnachrichten

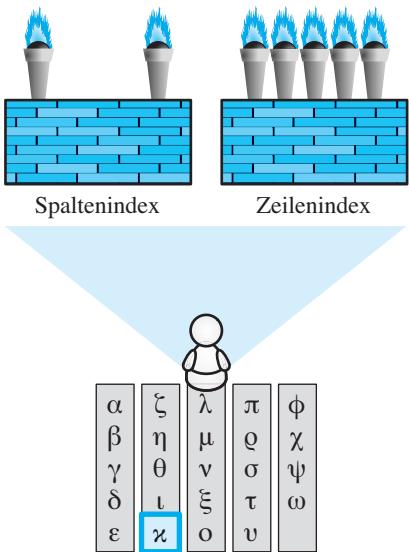


Abb. 1.5: Fackelcode des Polybios

montiert war. Eines der Gefäße behielt der Sender bei sich, das andere wurde auf der Empfängerseite oder in einem dazwischen geschalteten Relaisstützpunkt postiert. Um eine Nachricht zu übertragen, mussten beide Gefäße zunächst bis zum Rand mit Wasser gefüllt werden, anschließend signalisierte der Sender den Beginn der Übertragung mit einem Lichtsignal. Daraufhin öffnete der Empfänger die Ausflussvorrichtung und ließ das Wasser langsam ab. Sobald der Sender ein zweites Lichtsignal gab, wurde die Ausflussöffnung wieder verschlossen und die übertragene Botschaft an dem eingesunkenen Nachrichtenstab abgelesen.

Ob die vorgeschlagene Konstruktion des Aineias in dieser Form Verwendung fand, wissen wir nicht [4]. Bis heute ist keine Überlieferung bekannt, die eine praktische Anwendung beschreibt.

### 1.1.3 Fackelcode des Polybios

Auch wenn der Synchrontelegraf des Aineias zu Zeiten der alten Griechen eine trickreiche Konstruktion war, haftete ihr ein entscheidender Makel an: Die übertragbaren Nachrichten waren auf eine vorher festgelegte Auswahl beschränkt. Die Meldung unvorhergesehener Ereignisse, wie sie insbesondere zu Kriegszeiten massenweise auftraten, war auf diese Weise unmöglich.

Als Lösung schlug Polybios vor, Nachrichten nicht mehr als Ganzes, sondern buchstabenweise zu codieren (Abbildung 1.5). Auf der Senderseite wurden 10 Fackeln entzündet, von denen jeweils 5 hinter einer gemeinsamen Sichtblende positioniert waren. Jede Fackel konnte einzeln über die Blende gehoben oder hinter ihr verborgen werden. Auf der Empfängerseite wurde die Sendevorrichtung durch eine Sichtröhre beobachtet und die Anzahl der Fackeln über der linken Sichtblende als Spaltennummer und die Anzahl der Fackeln über der rechten Sichtblende als Zeilennummer einer Buchstabentafel interpretiert. Das abgebildete Zeichen wurde notiert und die gesendete Nachricht Buchstabe für Buchstabe rekonstruiert.

Was Polybios ersann, war der ersten *Zeichencode* in der Geschichte der Nachrichtenübertragung. Trotz seiner revolutionären Idee fand sein Vorschlag aber kaum praktische Verwendung. Das Fernrohr war noch lange nicht erfunden und die Abstände zwischen der Sende- und der Empfangsstation daher auf einige hundert Meter beschränkt. Die hohe Anzahl notwendiger Relaisstationen machte eine Übertragung über längere Distanzen praktisch unmöglich.



Claude Chappe wurde am 25. Dezember 1763 in Brûlon geboren, einem kleinen Dorf, 200 km südöstlich von Paris. Claude war eines von zehn Kindern einer einflussreichen Familie, die seine Zukunft im Schoße der Kirche sah. Bereits während seiner Ausbildung zum *Abbé Commendataire* zeigte Chappe reges Interesse für die Wissenschaft. In Paris entdeckte er seine Leidenschaft für die Physik und wurde kurze Zeit später Mitglied in der angesehenen *Société Philomathique*.

Die einschneidenden Ereignisse der französischen Revolution im Jahr 1789 brachten große Veränderungen in Chappes Leben. Er verlor seinen geistlichen Status und war gezwungen, nach Brûlon zurückzukehren. Dort befasste er sich, zusammen mit seinen Brüdern, intensiv mit den verschiedenen Möglichkeiten, differenzierte Nachrichten über längere Distanzen zu übertragen. Im März 1791 fand die erste öffentliche Demonstration einer ihrer Prototypen statt, ein Jahr später trat Chappe vor die gesetzgebende Nationalversammlung in Paris.

Sein Engagement zahlte sich aus. Er wurde zum *Ingénieur Télégraphe* ernannt und durch den französischen Staat finanziell unterstützt. Die nächsten Jahre waren ein Ritt auf einer

Welle des Erfolgs. Während Chappe die ersten Teststrecken errichtete, gelang es ihm, die prototypischen Entwürfe zu einem praxistauglichen Telegrafensystem weiterzuentwickeln und die Erlaubnis für den Bau weiterer Strecken zu erhalten. Die Geschichte von Claude Chappe endet tragisch, denn Ruhm und Ehre waren dem Franzosen hauptsächlich zu Beginn seines ehrgeizigen Projekts vergönnt. Mit zunehmendem Erfolg mehrten sich kritische Stimmen; Konkurrenten bemängelten seinen Entwurf als technisch minderwertig oder bezichtigten ihn des Plagiats. Selbst ehemalige Freunde standen ihm mit einem Mal als Widersacher gegenüber. Chappe ertrug die Anfeindungen nicht auf Dauer und verfiel in eine tiefe Depression. Als gebrochener Mann schied er am 23. Januar 1805 selbstbestimmt aus dem Leben – im Alter von 41 Jahren. In einer kurzen Notiz begründete Chappe seinen Freitod mit den folgenden Worten [42]:

*„Je me donne la mort pour éviter l’ennui de la vie qui m’accable; je n’ai point de reproches à me faire.“*

*„I give myself to death to avoid life’s worries that weigh me down; I’ll have no reproaches to make myself.“*



Claude Chappe  
(1763 – 1805)

## 1.2 Das mechanische Internet

### 1.2.1 Semaphoren-Telegraf von Claude Chappe

*Liberté, égalité, fraternité.* Mit dem Sturm auf die Bastille begann am 14. Juli 1789 die französische Revolution, und mit ihr eine der größten Umbruchphasen in der europäischen Geschichte. Chaotische Zustände beherrschten das Land und die sich abzeichnenden Kriege mit den europäischen Nachbarn verschärften die Situation zusehends. Die Zeit der französischen Revolution war der geeignete Nährboden für ein Vorhaben, das man noch wenige Jahre zuvor als bedeutungslos zurückgewiesen hätte: die Errichtung eines landesweiten optischen Kommunikationsnetzes. In Anbetracht der vielschichtigen Bedrohungen, denen Frankreich jener Tage gegenüberstand, war die Fähigkeit, differenzierte Nachrichten über große Distanzen in kurzer Zeit zu übertragen, mehr als pure Spielerei. Sie war zu einer Notwendigkeit geworden.

Der Weg für die Errichtung eines landesweiten Kommunikationsnetzes wurde in Paris geebnet. Am Abend des 24. März 1792 trat ein Mann

**Abb. 1.6:** Mit Claude Chappe beginnt die Ära der flächendeckenden Telekommunikation.

vor die Nationalversammlung, um den Inhalt eines Antrags vorzustellen, den er zwei Tage zuvor in Schriftform eingereicht hatte. Der Redner war der erst 28 Jahre alte Claude Chappe (Abbildung 1.6). Der junge Franzose war davon überzeugt, mit seiner Erfindung, die er als *tachygraphe* (dt. *Tachygraf*, *Schnellschreiber*) bezeichnete, ein Kommunikationsnetz errichten zu können, mit dem sich differenzierte Nachrichten in Windeseile über große Distanzen übertragen ließen (Abbildung 1.7).

Sein Antrag wurde mit Interesse zur Kenntnis genommen und zur Prüfung an ein Komitee übergeben. Die Entscheidung fiel positiv aus, und im April 1793 wurde Chappe beauftragt, eine ca. 30 km lange Teststrecke zu errichten. Die Strecke begann in Ménilmontant, im Nordosten von Paris. Von dort aus wurden die Signale auf die 15 km nördlich gelegenen Höhen von Ecouen gesendet und dann weiter in die 11 km entfernte Ortschaft Saint-Martin-du-Tertre [42].

Der Tachygraf bestand die praktische Erprobung, und Chappe erhielt den Auftrag für den Bau einer Kommunikationslinie zwischen Paris und dem 210 km entfernten Lille. Im Juli 1794 waren die insgesamt 23 Relaisstationen fertiggestellt und gingen kurze Zeit später in Betrieb. Offiziell wurde der Tachygraf ab jetzt als *Télégraphe* bezeichnet (dt. *Telegraf*, *Fernschreiber*).

Eine einzelne Telegrafenstation bestand aus einem Mast, an dessen oberen Ende der *Regulator* befestigt war (Abbildung 1.8 rechts). Hierbei handelte es sich um einen ca. 4,5 m langen Balken, der sich durch am Boden befindliche Kurbeln in vier verschiedene Stellungen drehen ließ (vertikal, diagonal fallend, horizontal, diagonal steigend). An jedem Ende befand sich ein weiterer ca. 2 m langer Balken. Diese *Indikatoren* konnten unabhängig von der Position des Regulators in 7 verschiedenen Winkelpositionen gebracht werden ( $0^\circ$ ,  $45^\circ$ ,  $90^\circ$ ,  $135^\circ$ ,  $225^\circ$ ,  $270^\circ$ ,  $315^\circ$ ). Da der Regulator und die Indikatoren unterschiedlich voneinander bewegt werden konnten, war es theoretisch möglich,  $4 \cdot 7 \cdot 7 = 196$  verschiedene Figuren darzustellen (Abbildung 1.8 links).

Die erste ereignisreiche Nachricht wurde am 15. August 1794 von Lille nach Paris telegrafiert, um die Rückeroberung der 200 km nördlich von Paris gelegenen Stadt Le Quesnoy zu melden. Für die Franzosen war dies der endgültige Beweis für den praktischen Nutzen der noch jungen Technik, und entsprechend schnell fiel die Entscheidung, weitere Kommunikationslinien zu errichten.

Noch im selben Jahr begann der Bau der Strecke Paris – Strasbourg. 1798 nahm sie den Betrieb auf, genauso wie die in nur 7 Monaten fertiggestellte Linie Paris – Brest.

Monsieur le Président,

*Je viens offrir à l'Assemblée Nationale l'hommage d'une découverte que je crois utile à la chose publique. Cette découverte suite de plusieurs années de travail et d'expérience présente un moyen facile de communiquer rapidement à de grandes distances tout ce qui peut être l'objet d'une correspondance. Le récit d'un fait ou d'un événement quelconque peut être transmis, la nuit ainsi que le jour, à plus de 40 mille dans moins de 46 minutes. Cette transmission s'opéroeroit d'une manière presqu'aussi rapide à une distance beaucoup plus grande le temps employé pour la communication n'augmentant point en raison proportionnelle des espaces.*

*Je puis, en 20 minutes, transmettre à la distance de 8 ou 10 mille la série de phrases que voici ou toute autre équivalente «Lukner s'est porté vers Mons pour faire le siège de cette ville. Bender s'est avancé pour la défendre. les deux généraux sont en présence. on livrera demain bataille». Ces mêmes phrases seroient communiquées en 24 minutes à une distance double de la première. En 33 minutes, elles parviendroient à 50 mille. La transmission à une distance de 100 mille ne nécessiteroit que 12 minutes de plus.*

*Parmi la multitude d'applications utiles, dont cette découverte est susceptible, il en est une qui, dans les circonstances présentes, est de la plus haute importance. Elle offre un moyen certain d'établir une correspondance telle que le Corps Législatif puisse faire parvenir ses ordres à nos frontières et en recevoir la réponse pendant la durée d'une même séance.*

*Ce n'est point sur une simple théorie que je fonde ces assertions. Plusieurs expériences tentées à la distance de dix mille, dans le département de la Sarthe, et suivies du succès, sont pour moi de sûrs garants de la réussite. Les procès-verbaux ci-joints, dressés par deux municipalités, en présence d'une foule de témoins, en attestent l'authenticité. L'obstacle qui me sera le plus difficile à vaincre sera l'esprit de prévention avec lequel on accueille ordinairement les faiseurs de projets. Je n'aurois jamais pu m'élever au-dessus de la crainte de leur être assimilé, si je n'avois été soutenu par la persuasion où je suis que tout citoyen français doit, dans ce moment plus que jamais, à son pays le tribut de tout ce qu'il croit lui être utile.*

*Je demande donc, Messieurs, que l'Assemblée Nationale renvoie à l'un de ses comités l'examen des objets que j'ai l'honneur de vous annoncer, afin qu'il nomme des commissaires pour en constater les effets, par une expérience qui sera d'autant plus facile à faire qu'en l'exécutant pour un distance de 8 à 10 mille, on sera à portée de se convaincre qu'elle peut s'appliquer à tous les espaces. Je la ferai au surplus à toutes les distances que l'on voudra m'indiquer, et je ne demande, en cas de réussite, qu'à être indemnisé des frais qu'elle aura occasionnés.*

Mr. President,

*I have come to offer to the National Assembly the tribute of a discovery that I believe to be useful to the public cause. This discovery provides a simple method for rapidly communicating over great distances, anything that could be the subject of a correspondence. The report of an event or an occurrence could be transmitted, by night or by day, over more than 40 miles in under 46 minutes. This transmission takes place almost as rapidly over a much larger distance (the time required for the communication does not increase proportionally with the distance).*

*I can, in 20 minutes, transmit over a distance of 8 to 10 miles, the following, or any other similar phrase: «Lukner has left for Mons to besiege that city. Bender is advancing for its defense. The two generals are present. Tomorrow the battle will start.» These same phrases are communicated in 24 minutes over a distance twice that of before; in 33 minutes they cover 50 miles. The transmission over a distance of 100 miles requires just 12 minutes more.*

*Among the many useful applications for which this discovery can be used, there is one that, under the present circumstances, is of the greatest importance. It offers a reliable way of establishing a correspondence by which the legislative branch of the government could send its orders to our frontiers, and receive a response from there while still in session.*

*My assertions are not just based on a simple theory. Many successful experiments, held at a distance of 10 miles, in the Sarthe department, are for me a certain guarantee that this can be accomplished. The attached affidavits, drawn up at two municipalities, in the presence of a range of witnesses, attest to its authenticity. The obstacle that seems to me to be the most difficult to overcome is the popular suspicion that usually confronts those who pursue projects such as these. I could never have escaped from the fear that has overtaken them, if I was not sustained by the conviction that I should, as every French citizen, today more than ever, contribute to his country what he can.*

*I ask, Sirs, that the Assembly submit to one of its committees the examination of this project that I have the honor to announce to you, so that they can appoint delegates to observe the results of an experiment readily performed at a distance of 8 to 10 miles, and convince themselves that the same can be accomplished at any distance. I will perform this experiment, and in addition, at any distance that is requested, and I ask only, in case of success, to be reimbursed for the expenses that are made.*

Abb. 1.7: Chappes Einreichung aus dem Jahr 1792, links im Original [46] und rechts in der Übersetzung aus [42]

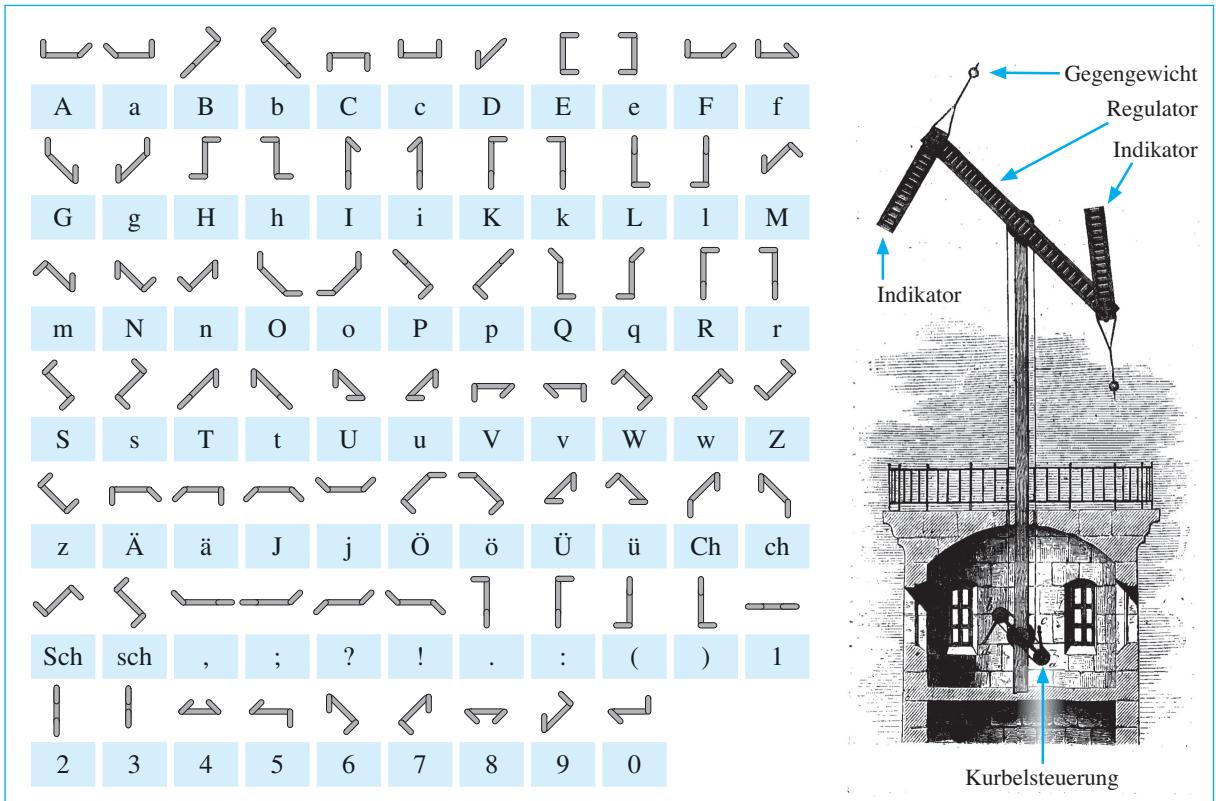


Abb. 1.8: Auszug aus dem preußischen Semaphoren-Alphabet (nach [45])

1803 wurde die Linie Paris – Lille nach Brüssel und Boulogne erweitert und 1804 das südöstlich von Paris gelegene Dijon und Lyon erschlossen (Abbildung 1.9 links).

In den Folgejahren wurde das Netz auf die südlichen Landesteile ausgedehnt und nach Norden und Osten über die Grenzen Frankreichs hinaus erweitert (Abbildung 1.9 rechts). Um 1845 erstreckte sich die nördliche Telegrafenlinie bereits nach Amsterdam, und im Osten waren die italienischen Städte Turin, Mailand und Venedig angebunden. Über Europa verbreitete sich ein zunehmend dichter werdendes Netzwerk, das die Übertragung über immer längere Entfernungen gestattete. Das mechanische Internet nahm Gestalt an.

All dies hatte Claude Chappe nicht mehr erlebt. Im Jahr 1845 jährte sich sein früher Freitod bereits zum vierzigsten Mal.

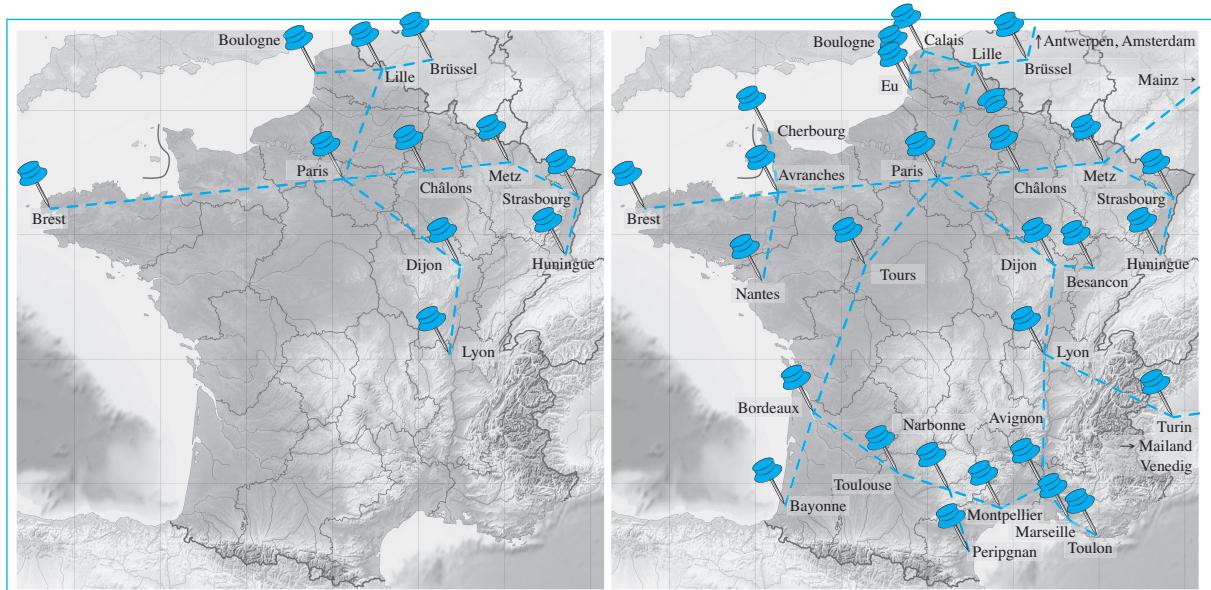


Abb. 1.9: Das Semaphoren-Netzwerk um 1805 (links) und 1845 (rechts) [42]

## 1.2.2 Klappentelegrafen

Zur selben Zeit, als der Chappe'sche Semaphoren-Telegraf das europäische Festland überzog, wurde auf der britischen Insel intensiv an Alternativen gearbeitet. Experimente wurden mit sogenannten *Shutter-Telegrafen* durchgeführt, die ein zu übertragendes Zeichen mithilfe von Klappen codierten. Jede einzelne Klappe konnte über einen Seilzug individuell bewegt und in zwei verschiedenen Positionen arretiert werden (sichtbar, nicht sichtbar). Shutter-Telegrafen arbeiteten damit nach dem gleichen Prinzip wie moderne Computer heute. Symbole wurden im Binärsystem dargestellt und nicht, wie beim Chappe'schen Semaphoren-Telegraf, mithilfe geometrischer Figuren.

Zwei Varianten des Shutter-Telegrafen wurden durch den Briten John Gamble erprobt (Abbildung 1.10). Sein erster Entwurf sah fünf Klappen vor, die senkrecht übereinander angeordnet waren. Damit waren 32 verschiedene Kombinationen darstellbar, allerdings hatte die Anordnung den Nachteil, dass die drei mittleren Klappen aus größerer Entfernung nicht mehr deutlich voneinander unterschieden werden konnten. Als Abhilfe konstruierte Gamble einen Shutter-Telegrafen mit nur noch 4 Klappen. Die Klappenstellungen waren jetzt über größere Distanzen sichtbar, dafür waren die verbliebenen 16 Kombinationen zu

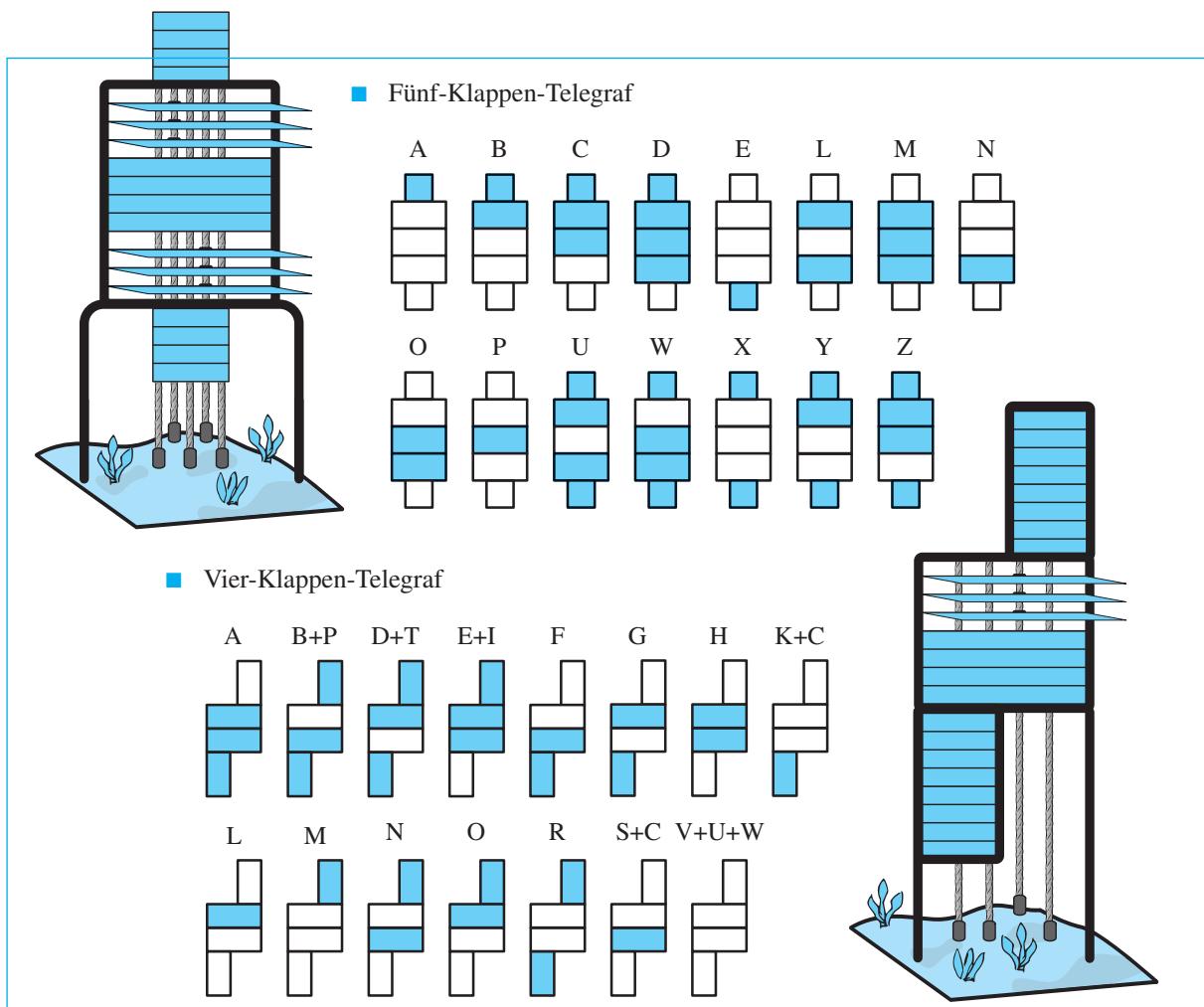


Abb. 1.10: Optische Telegrafen nach den Entwürfen von John Gamble aus dem Jahr 1795

wenige, um das komplette Alphabet zu codieren. Um trotzdem gewöhnliche Nachrichten übertragen zu können, nutzte Gamble aus, dass Texte auch dann noch korrekt verstanden werden, wenn ein Buchstabe durch einen ähnlich klingenden ausgetauscht wird. Indem er phonetisch verwandte Buchstaben mit dem gleichen Codewort belegte, konnte er mit den 16 Bitmustern einen praktikablen Code konstruieren.

Im Jahr 1795 wurde der Bau einer Telegrafenlinie zwischen London und der nördlich von Dover liegenden Hafenstadt Deal beschlossen. Zur Überraschung vieler wurde hierfür keiner von Gambles Entwürfen ge-

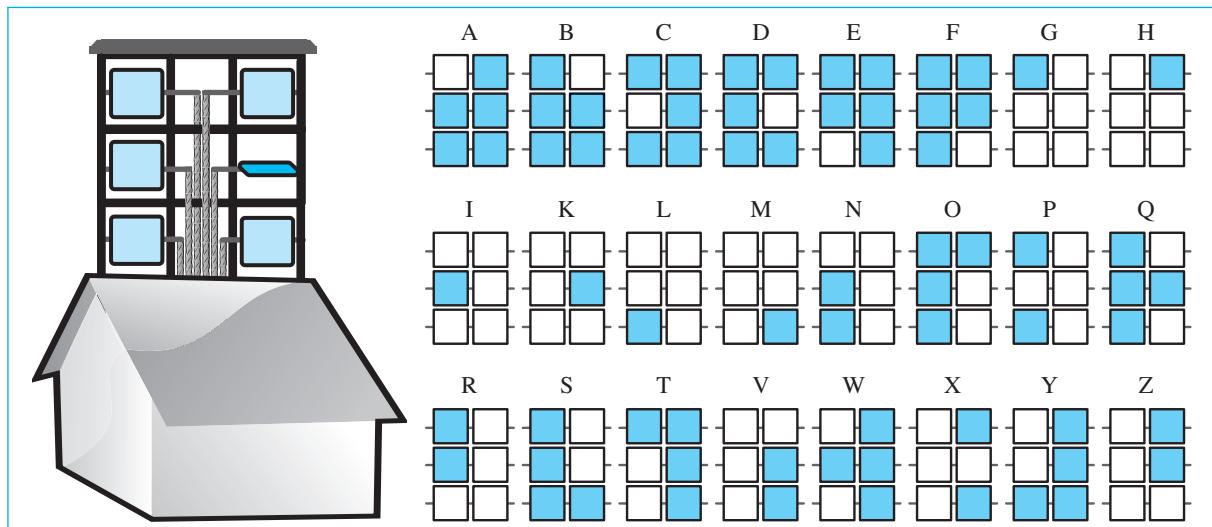


Abb. 1.11: Optischer Telegraf nach Lord Murray

wählt. Stattdessen entschied man sich, die Telegrafenstationen nach den Plänen des anglikanischen Bischofs Lord George Murray zu errichten. Anders als der Gamble-Telegraf nutzte der Murray-Telegraf 6 Klappen zur Codierung der Symbole, die in zwei Spalten mit jeweils drei Klappen angeordnet waren (Abbildung 1.11). Der Murray-Telegraf konnte sich in der Praxis bewähren, und ein Jahr später wurde eine Kommunikationsstrecke von London nach Portsmouth errichtet. 1805 wurde sie nach Westen in die Stadt Plymouth und 1808 nach Osten in die Stadt Yarmouth verlängert.

Im Gegensatz zum Semaphoren-Telegrafen war dem Shutter-Telegrafen nur ein kurzes Leben beschert. Die englischen Telegrafenlinien wurden ausschließlich zur Übertragung militärischer Nachrichten genutzt und nach dem 1814 geschlossenen Pariser Frieden und der Verbannung Napoleons von vielen als bedeutungslos erachtet. In der Folge wurden die Telegrafenlinien außer Betrieb genommen und die Stationen ihrem Schicksal überlassen. Nach der Rückkehr Napoleons war es dann zu spät: Die Stationen waren bereits so verwildert, dass man sich gegen eine Instandsetzung entschied. Stattdessen wurde eine neue Linie errichtet, die dem Chappe'schen Semaphoren-Netzwerk sehr ähnlich war. An einem senkrechten Mast waren drei Indikatoren angebracht, die unabhängig voneinander in verschiedene Winkelstellungen gebracht werden konnten. Auf einen Regulator, wie er beim Chappe-Telegrafen zum Einsatz kam, wurde verzichtet.



William Gilbert  
(1544 – 1603)



Titelbild der zweiten Auflage von  
*De Magnete* aus dem Jahr 1628

**Abb. 1.12:** Die im Jahr 1600 in Latein verfasste Schrift *De Magnete* ist das Hauptwerk von William Gilbert. Sie war die erste Abhandlung, die sich systematisch mit den Phänomenen des Magnetismus beschäftigte, und wird, wegen ihrer wissenschaftlichen Strenge, von vielen als ein wegbereitendes Werk für die moderne Physik und Astronomie angesehen.

Heute ist kaum noch etwas von den optischen Telegrafennetzwerken zu sehen, die einst weite Teile Europas überzogen. Von ein paar Erinnerungsstätten abgesehen, sind die unzähligen im Laufe der Zeit errichteten Relaisstationen aus dem Landschaftsbild verschwunden. Es sind Ortsbezeichnungen wie *Telegraph hill*, die als letzte verbliebene Zeugen still an diese weitestgehend vergessene Ära der Nachrichtentechnik erinnern.

### 1.3 Elektrische Nachrichtenübertragung

Viele Naturphänomene, die auf den Wirkungsmechanismen der Elektrizität beruhen, sind der Menschheit schon lange bekannt. So machte der griechische Philosoph Thales von Milet bereits 550 v. Chr. die Entdeckung, dass Bernstein<sup>1</sup> leichte Gegenstände anzuziehen vermag, wenn er zuvor mit einem Tuch abgerieben wurde [65].

Die systematische Untersuchung elektrischer und magnetischer Phänomene begann im 16. Jahrhundert durch den englischen Arzt William Gilbert [34] (Abbildung 1.12), und im 17. Jahrhundert wurden die ersten Maschinen gebaut, die durch das Prinzip der Reibungselektrizität hohe Spannungen erzeugten. Allerdings war man noch weit davon entfernt, die physikalischen Zusammenhänge zu verstehen, die sich hinter den kuriosen Erscheinungen verbargen.

Betrachten wir die Entwicklung der Nachrichtenübertragung aus der Ferne, so erscheint die Ablösung der optischen Telegrafie durch die elektrische als eine Entwicklung des gewöhnlichen Fortschritts. In Wirklichkeit gehen die Wurzeln der elektrischen Telegrafie jedoch auf eine Zeit zurück, in der die erste optische Semaphoren-Station noch gar nicht errichtet war. Bereits im Jahr 1753 publizierte das *Scots Magazine* in seiner Februar-Ausgabe den Brief eines gewissen C. M., in dem die nahezu vollständige Beschreibung eines elektrischen Telegrafen nachzulesen ist. Der Brief beginnt mit den folgenden Worten:

„Sir, – It is well known to all who are conversant in electrical experiments, that the electric power may be propagated along a small wire, from one place to another, without being sensibly abated by the length of its progress. Let, then, a set of wires, equal in number to the letters of the alphabet, be extended horizontally between two given places, parallel to one another, and each of them about an inch, distant from that next to it. At every twenty

<sup>1</sup>Das griechische Wort für Bernstein ist ήλεκτρον, gesprochen elektron.

*yards' end let them be fixed in glass or jewelers' cement to some firm body, both to prevent them from touching the earth or any other non-electric, and from breaking by their own gravity. Let the electric gun-barrel be placed at right angles with the extremities of the wires, and about one inch below them. Also let the wires be fixed in a solid piece of glass at six inches from the end, and let that part of them which reaches from the glass to the machine have sufficient spring and stiffness to recover its situation after having been brought in contact with the barrel. Close by the supporting glass let a ball be suspended from every wire; and about a sixth or an eighth of an inch below the balls place the letters of the alphabet, marked on bits of paper, or any other substance that may be light enough to rise to the electrified ball, and at the same time let it be so continued that each of them may reassume its proper place when dropped.“*



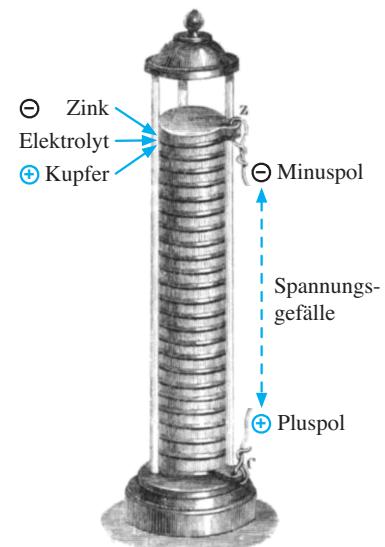
Alessandro Volta  
(1745 – 1827)

Bis heute ist nicht zweifelsfrei geklärt, wer sich hinter den Initialen C. M. verbirgt [42].

Die Vorteile eines elektrisch arbeitenden Kommunikationsnetzes liegen auf der Hand. Während z. B. das Chappe'sche Semaphoren-Netzwerk nur am Tage bei guten Sichtverhältnissen zuverlässig betrieben werden konnte, ist ein elektrisches Kommunikationsnetz rund um die Uhr verfügbar, bei guten Witterungsbedingungen genauso wie bei schlechten. Ferner benötigt ein elektrisches Netzwerk weniger Relaisstationen und lässt sich daher kostengünstiger und weniger fehleranfällig betreiben.

Den verlockenden Vorteilen zum Trotz war der elektrischen Telegrafie kein schneller Erfolg vergönnt, da gegen Ende des 18. Jahrhunderts die technologische Entwicklung noch nicht weit genug gediehen war, um die Vorzüge der Elektrizität praktisch zu nutzen. Erst mit der Volta'schen Säule, einer Erfindung des italienischen Physikers Alessandro Volta aus dem Jahr 1800, war man in der Lage, kontinuierliche Stromflüsse zu erzeugen (Abbildung 1.13).

Einer der ersten, der sich die Volta'sche Säule für die elektrische Telegrafie zu Nutze machte, war der deutsche Anatomieprofessor Samuel Thomas von Soemmerring. Im Jahr 1809 demonstrierte er vor den Mitgliedern der Königlichen Akademie der Wissenschaften in München, wie sich Textnachrichten auf elektrischem Weg übertragen lassen (Abbildung 1.14). Seine Apparatur bestand aus insgesamt 35 separaten Leitungen: eine für jeden Buchstaben des Alphabets und eine für jede Dezimalziffer<sup>2</sup>. Auf der Empfängerseite waren die Stromleitungen mit Elektroden verbunden, die in ein Wasserbad eingelassen waren.

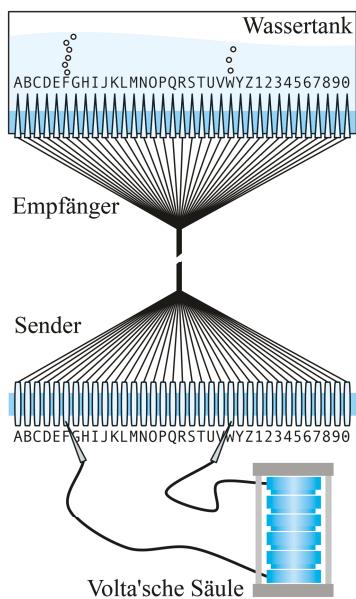


**Abb. 1.13:** Die Volta'sche Säule ist der Vorläufer der modernen Batterie. Der Italiener Alessandro Volta konstruierte sie aus mehreren in Reihe geschalteten galvanischen Zellen, die aus einer Zinkplatte, einer Kupferplatte und einer dazwischen liegenden Elektrolytschicht bestanden. Dank dieser revolutionären Erfindung war es ab dem Jahr 1800 möglich, einen kontinuierlichen Stromfluss zu erzeugen.

<sup>2</sup>Tatsächlich hielt der Soemmerring-Telegraf nur für 25 der 26 Buchstaben eine Leitung vor. Der Buchstabe 'X' konnte nicht übertragen werden.



Samuel Thomas von Soemmerring  
(1755 – 1830)



**Abb. 1.14:** Der Soemmerring-Telegraf gehörte zu den ersten Apparaturen, die Nachrichten mithilfe des elektrischen Stroms übertrugen. Um die ankommenden Zeichen sichtbar zu machen, bediente sich Soemmerring des Prinzips der Elektrolyse. Aufsteigende Bläschen verrieten, über welche Elektroden der Stromkreis geschlossen wurde.

Mit dem Soemmerring-Telegrafen konnten Zeichen paarweise übertragen werden, indem auf der Senderseite zwei Leitungen mit den beiden Polen der Volta'schen Säule verbunden wurden. Eine der beiden fungierte als Hinleiter und der andere als Rückleiter. Das Schließen des Stromkreises hatte zur Folge, dass über den betreffenden Elektroden auf der Empfängerseite Bläschen aufstiegen. Die zugehörigen Symbole wurden notiert und die Nachricht Buchstabenpaar für Buchstabenpaar rekonstruiert. Später hatte Soemmerring seinen Telegrafen weiter verbessert. Er stattete ihn mit einem gemeinsamen Rückleiter aus, sodass Buchstaben nicht nur paarweise, sondern auch einzeln übertragen werden konnten. Ferner montierte er eine spezielle Kippvorrichtung, die durch das Senden der Buchstabenkombination 'BC' aktiviert wurde und mit einer Klingel verbunden war. Gedacht war die Klingelvorrichtung, um den Beginn einer Nachrichtenübertragung zu signalisieren.

Der Soemmerring-Telegraf ist eine trickreiche Konstruktion und verdeutlicht auch gerade deshalb ein großes Problem seiner Zeit. Mit der Volta'schen Säule war es zwar möglich, einen kontinuierlichen Stromfluss zu erzeugen, allerdings fehlten die technischen Mittel, diesen auf einfache Weise sichtbar zu machen. Die Glühlampe war noch nicht erfunden, und so sah Soemmerring zur damaligen Zeit keine andere Möglichkeit, als sich des chemischen Prinzips der Elektrolyse zu bedienen. Für eine praktische Nutzung war sein Telegraf hierdurch viel zu langsam, und es vergingen mehr als 10 Jahre, bis eine Reihe epochaler Entdeckungen den Traum von einem elektrischen Telegrafen tatsächlich in greifbare Nähe rücken ließ.

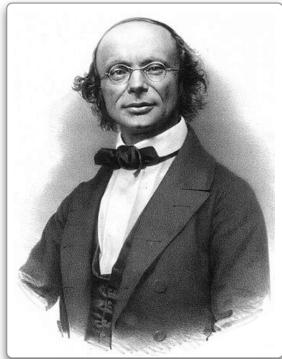
Den Anfang machte der Däne Hans Christian Ørsted im Jahr 1820. Er beobachte, dass sich Magnettadeln durch elektrische Ströme ausrichten lassen, und hatte damit eine einfache Methode gefunden, um fließende Ströme sichtbar zu machen. In kurzen Abständen folgten weitere Entdeckungen. Der Franzose André-Marie Ampère wiederholte die Versuche von Ørsted und schaffte es, die beobachteten Phänomene durch eine mathematische Theorie zu erklären. 1824 gelang dem englischen Physiker William Sturgeon der Bau des ersten Elektromagneten, und ein Jahr später formulierte Georg Simon Ohm den elementaren Zusammenhang zwischen Stromstärke, Spannung und Widerstand, den wir heute als das *Ohm'sche Gesetz* bezeichnen.

Die wachsende Fähigkeit, elektrische und magnetische Phänomene zu verstehen, zu messen und zu kontrollieren, führten zu massiven Veränderungen in nahezu allen Bereichen der Wissenschaft und Technik. Auch die Vorstöße, die neu gewonnenen Erkenntnisse auf dem Gebiet der Nachrichtenübertragung praktisch zu verwerten, sollten als bald Früchte tragen.

a	↖	l	↗↖↖	v	↖↗↖↖↖
b	↗	m	↗↖↖↗	w	↖↗↖↖↖↗
c	↖↖	n	↗↗↖	x	↗↖↖↖↖
d	↖↗	o	↗↗↗	y	↗↖↖↖↖↗
e	↗↖	p	↖↖↖↖	z	↗↖↖↖↖↗
f	↗↗	q	↖↖↖↗		
g	↖↖↖	r	↖↖↗↖		
h	↖↖↗	s	↖↖↗↗		
i	↖↗↖	t	↖↖↖↖↖		
k	↖↗↗	u	↖↖↖↖↖		



Carl Friedrich Gauß  
(1777 – 1855)



Wilhelm Eduard Weber  
(1804 – 1891)

Abb. 1.15: Telegrafencode von Gauß und Weber

### 1.3.1 Nadeltelegrafen

Zu den ersten, die sich der neuen Erkenntnisse auf dem Gebiet des Elektromagnetismus bedienten, gehörten der deutsche Mathematiker Carl Friedrich Gauß und der Physiker Wilhelm Eduard Weber. Als Sender benutzten die beiden Professoren einen Magnetstab, der in einer Spule eingebettet war und bei jeder Bewegung einen elektrischen Impuls erzeugte. Durch ein Kabel wurden die Impulse an den Empfänger übertragen, wo sie einen beweglichen Spiegel leicht nach links oder rechts drehten. Die ausgelöste Bewegung war zwar minimal, mit einem in fünf Meter Entfernung platzierten Teleskop aber gut zu beobachten.

Genau wie im Falle der optischen Telegrafie sahen Gauß und Weber vor, Textnachrichten zeichenweise zu übertragen. Für diesen Zweck ersannen sie einen eigenen Binärcode, der jedes Zeichen durch eine Folge von linken und rechten Spiegelbewegungen darstellte (Abbildung 1.15).

Gauß und Weber nahmen ihre Apparatur zunächst im Labor in Betrieb. Später verlegten sie ein ca. 1,2 km langes Kupferkabel über den Dächern von Göttingen, um die Übertragung über längere Distanzen zu testen. Ausgangspunkt war das physikalische Kabinett Webers. Von dort verlief das Kabel zum Turm der Johanniskirche über das Accouhierhaus bis hinein in das Arbeitszimmer von Gauß in der Göttinger Sternwarte.



William Fothergill Cooke  
(1806 – 1879)

**Abb. 1.16:** Zusammen mit Charles Wheatstone baute William Fothergill Cooke einen der ersten praxistauglichen elektrischen Telegrafen.

Was Gauß und Weber erfanden, war nichts Geringeres als der erste elektromagnetische Telegraf der Welt. Eine große Zukunft war diesem jedoch nicht besichert. Die beiden Professoren hatten ihre Versuchsanordnung aus wissenschaftlichem Interesse gebaut und waren an keiner kommerziellen Nutzung interessiert.

Ein anderer, der das Prinzip des Elektromagnetismus für die Übertragung von Nachrichten einzusetzen versuchte, war Paul Ludwig Schilling von Cannstatt. In den Jahren 1809 – 1811 assistierte er Soemmerring beim Bau des elektrolytischen Telegrafen und führte die Forschungsarbeiten später eigenständig fort. Nach mehreren gescheiterten Versuchen, die Soemmerring'sche Apparatur zu verbessern, verwarf er das Prinzip der Elektrolyse und wandte sich der Entwicklung elektromagnetischer Telegrafen zu. In dieser Zeit entstanden mehrere Prototypen, die ein übertragenes Zeichen über bewegliche Magnettadeln codierten. Der bekannteste seiner Entwürfe war ein Fünf-Nadel-Telegraf, den er am 23. September 1835 auf der Jahresversammlung der Gesellschaft Deutscher Naturforscher und Ärzte in Bonn vorführte. Einer der Anwesenden war der Physikprofessor Georg Wilhelm Munke. Von dem Nadeltelegrafen tief beeindruckt, ließ er mehrere Kopien anfertigen, und verwendete diese fortan als Demonstrationsobjekte in der Lehre. Schilling von Cannstatt konnte die großen Pläne, die er mit seinem Telegrafen damals im Sinn hatte, nicht mehr realisieren. Nach seinem frühen Tod im Jahr 1837 wurden alle in Planung befindlichen Projekte gestoppt.

Trotzdem sollte seine Erfindung die weitere Entwicklung der Nachrichtenübertragung nachhaltig beeinflussen. Munkes Vorlesung wurde von vielen angehenden Wissenschaftlern besucht und der Nadeltelegraf auf diese Weise vor dem Vergessen bewahrt. Zu den Hörern seiner Vorlesung gehörte auch der Brite William Fothergill Cooke (Abbildung 1.16). Er erkannte das große Potenzial des Nadeltelegrafen und begann umgehend mit der Entwicklung eigener Prototypen. Bereits nach drei Wochen war das erste Ergebnis greifbar: Cooke hatte einen Telegrafen mit drei Nadeln konstruiert, die über 6 elektrische Leitungen unabhängig voneinander ausgerichtet werden konnten. Die im Labor durchgeführten Versuche verliefen vielversprechend, doch die eigentliche Bewährungsprobe stand noch aus: die Übertragung von Nachrichten über große Distanzen.

Anders als heute waren lange elektrische Leiter damals schwer zu beschaffen und so musste Cooke diesen entscheidenden Test bis zu seiner Rückkehr nach England aufschieben. Dort stand ihm ein 1,6 km langes Kabel zur Verfügung, mit dem er seine Experimente wiederholte. Die Ergebnisse waren erschütternd: Während seine Apparatur über kurze



Joseph Henry wurde am 17. Dezember 1797 in Albany, im Bundesstaat New York, geboren. Aufgewachsen ist Henry in ärmlichen Verhältnissen,

und bereits in jungen Jahren musste er den Tod seines Vaters verkraften. Im Alter von 13 machte er eine Ausbildung als Uhrmacher und Silberschmied. Seine Leidenschaft für die Wissenschaft entdeckte er mit 16, als er durch Zufall das Buch *Lectures on Experimental Philosophy* im Bücherregal der Kirche erspähte. 1919 besuchte er die *Albany Academy* und sicherte sich mit verschiedenen Lehrtätigkeiten seine finanzielle Existenz.

Henrys außerordentliche Begabung ebnete ihm den Weg für eine makellose wissenschaftliche Karriere. 1826 wurde er Professor für Mathematik und Naturphilosophie an der Albany Academy, wo er einige seiner wichtigsten Entdeckungen machte. Unzählige akribisch durchgeführte Experimente halfen ihm, das komplizierte Wechselspiel zwischen elektrischen und magnetischen Kräften zu verstehen und für den Bau leistungsfähiger Elektromagneten nutzbar zu machen. 1832 berief ihn das Princeton College für seine wegbereitenden Entdeckungen zum Professor der Naturwissenschaften.

1846 wurde er der erste Sekretär der Smithsonian Institution in Washington, D.C., und noch heute ist sein Name eng mit dieser altehrwürdigen Forschungs- und Bildungseinrichtung verbunden.

Die Geschichte der Nachrichtenübertragung ist auf vielfältige Weise mit dem Lebensweg von Joseph Henry verwoben. Zum einen zeichnet Henry für einen großen Teil der Grundlagenforschung verantwortlich, ohne die kein elektrischer Telegraf jemals hätte gebaut werden können. Zum anderen trat er gleich mehrmals als Ideenvermittler auf. Er war es, der Cooke und Morse das Problem der langen Leitung lösen ließ, und er war es auch, der etliche Jahre später Graham Bell mit den Entwürfen von Philipp Reis vertraut machte und damit wichtige Impulse für den Bau des ersten Telefons lieferte.

Joseph Henry starb in Washington, D.C., am 13. Mai 1878, im Alter von 80 Jahren. Bereits zu Lebzeiten war er ein gefeierter Wissenschaftler, doch die größte Auszeichnung wurde ihm posthum durch den *International Electrical Congress* im Jahr 1893 verliehen. Seit diesem Jahr messen wir die Induktivität eines stromdurchflossenen Leiters in der Einheit 'H' für Henry.

Distanzen nahezu perfekt arbeitete, schlugen alle Versuche, Nachrichten über längere Entfernungen zu übertragen, fehl. Cooke bemühte sich verzweifelt, seine Apparatur zu verbessern, doch es gelang ihm nicht, das Problem der langen Leitung zu lösen. Nach zahllosen gescheiterten Versuch war sein Traum, Nachrichten über große Entfernungen elektrisch zu telegrafieren, in weite Ferne gerückt.

Cooke ahnte nicht, dass man für das Problem der langen Leitung auf der anderen Seite des Atlantiks bereits seit längerem eine Lösung in Händen hielt. Im Jahr 1829 hatte der amerikanische Physiker Joseph Henry (Abbildung 1.17) entdeckt, dass die Übertragung immer dann gelang, wenn anstelle einer einzigen, großen Batterie, mehrere kleine, in Reihe geschalteten Batterien verwendet werden. Cooke erfuhr von diesen Ergebnissen indirekt, bei einem Treffen mit dem britischen Physiker Charles Wheatstone (Abbildung 1.18). Der Physikprofessor hatte die fachliche Expertise, die Cooke so dringend benötigte, doch an einer kommerziellen Vermarktung war er zunächst nicht interessiert. In der Folge entstand eine Partnerschaft, die menschlich schwierig, aber ökonomisch erfolgreich war.

Abbildung 1.19 zeigt eines ihrer berühmtesten Ergebnisse, den *five needle telegraph*. Den wichtigsten Teil der Konstruktion bildete ein rau-



Joseph Henry (1797 – 1878)

**Abb. 1.17:** Dem Amerikaner Joseph Henry verdanken wir wertvolle Erkenntnisse auf dem Gebiet des Elektromagnetismus. Ohne seine bahnbrechenden Entdeckungen wäre der Bau des elektrischen Telegrafen nicht möglich gewesen.



Sir William Fothergill Cooke wurde am 4. Mai 1806 in Ealing als Sohn des englischen Anatomieprofessors William Cooke geboren. Nach mehreren Jahren im Dienste des britischen Militärs im indischen Madras nahm Cooke das Studium der Anatomie auf, wo er in einer von Munkes Vorlesungen den Schilling'schen Telegrafen in Aktion erleben durfte. Von der Demonstration tief beeindruckt, änderte er seine Pläne und verschrieb sich fortan der elektrischen Telegraphie.

Da Cooke nur wenige Kenntnisse auf dem Gebiet der Elektrizität besaß, stand er schon bald vor einer Reihe technischer Probleme, die er nicht alleine lösen konnte. Erst durch die Partnerschaft mit Charles Wheatstone gelang es ihm, diese Hürden zu nehmen.

Die Telegrafen von Cooke und Wheatstone markieren einen Meilenstein in der Geschichte der Nachrichtenübertragung, und 1869 wurde Cooke für seine Verdienste zum Ritter geschlagen. Leider hatte sich sein Leben in den Jahren zuvor nicht wie erhofft entwickelt. Er war in finanzielle Nöte geraten, von denen er sich bis zu seinem Lebensende nicht mehr erholen konnte. Cooke starb am 25. Juni 1879 als verarmter Mann.



Sir Charles Wheatstone wurde am 6. Februar 1802 in der Nähe von Gloucester geboren. Er ging als bedeutender Forscher des 19. Jahrhunderts in die Geschichte ein und hinterließ ein wichtiges wissenschaftliches Vermächtnis. Zu seinen bekanntesten Erfindungen gehören das Stereoskop, die *Playfair-Verschlüsselung* und die *Wheatstone'sche Brücke* für die Messung von elektrischen Widerständen.

Als Geschäftspartner von Cooke spielte er eine bedeutende Rolle in der Geschichte des elektrischen Telegrafens. Er war es, der die fachliche Expertise beisteuerte, ohne die Cooke seine Vision nicht hätte umsetzen können. Leicht war die Zusammenarbeit nicht. Wheatstone war eine schwierige Persönlichkeit, und die Partnerschaft mit Cooke war durch fortwährende Differenzen geprägt. Nach außen hin wurde Wheatstone von vielen als ruhig und zurückhaltend beschrieben, gleichsam wurde ihm ein verbissenes Streben nach Ruhm und Anerkennung attestiert [86].

In akademischer Hinsicht war Wheatstone ein honoriert Mann. Bereits 1836 wurde er in die Royal Society aufgenommen und in den Folgejahren mehrfach ausgezeichnet. Im Jahr 1868 erhielt er den Ritterschlag.



Charles Wheatstone (1802 – 1875)

**Abb. 1.18:** Der britische Physiker Charles Wheatstone war der Geschäftspartner von William Fothergill Cooke. Er steuerte die fachliche Expertise für den Bau des Nadeltelegrafen bei.

tenförmiges Brett, auf dem fünf bewegliche Nadeln befestigt waren. In ihrer neutralen Position waren die Nadeln vertikal ausgerichtet und konnten durch das Anlegen einer elektrischen Spannung nach links oder rechts gedreht werden. Um ein Zeichen zu übertragen, wurde die Apparatur so angesteuert, dass eine Nadel nach links und eine andere Nadel nach rechts ausschlug. Hierdurch wurden zwei Diagonalen ausgewählt, an deren Schnittpunkt das codierte Zeichen annotiert war. Da sich auf diese Weise genau 20 Zeichen codieren lassen, konnte der Nadeltelegraf nicht das gesamte Alphabet erfassen. Ein Blick auf das Rautenbrett zeigt, dass sich die Buchstaben ‚C‘, ‚J‘, ‚Q‘, ‚U‘, ‚X‘ und ‚Z‘ nicht übertragen ließen.

Eingesetzt wurde der Nadeltelegraf von der Great Western Railways Company, die Cooke und Wheatstone mit dem Bau einer 21 km langen Telegrafenlinie zwischen Paddington Station in London und West Drayton beauftragte. Am 9. Juli 1839 wurde der Bau beendet und die Telegrafenlinie offiziell in Betrieb genommen.

In der Folgezeit entwickelten Cooke und Wheatstone ihre Erfindung kontinuierlich weiter. Auf den Fünf-Nadel-Telegrafen folgte eine Apparatur mit zwei Nadeln, und später kam der *Zeigertelegraf* hinzu, der

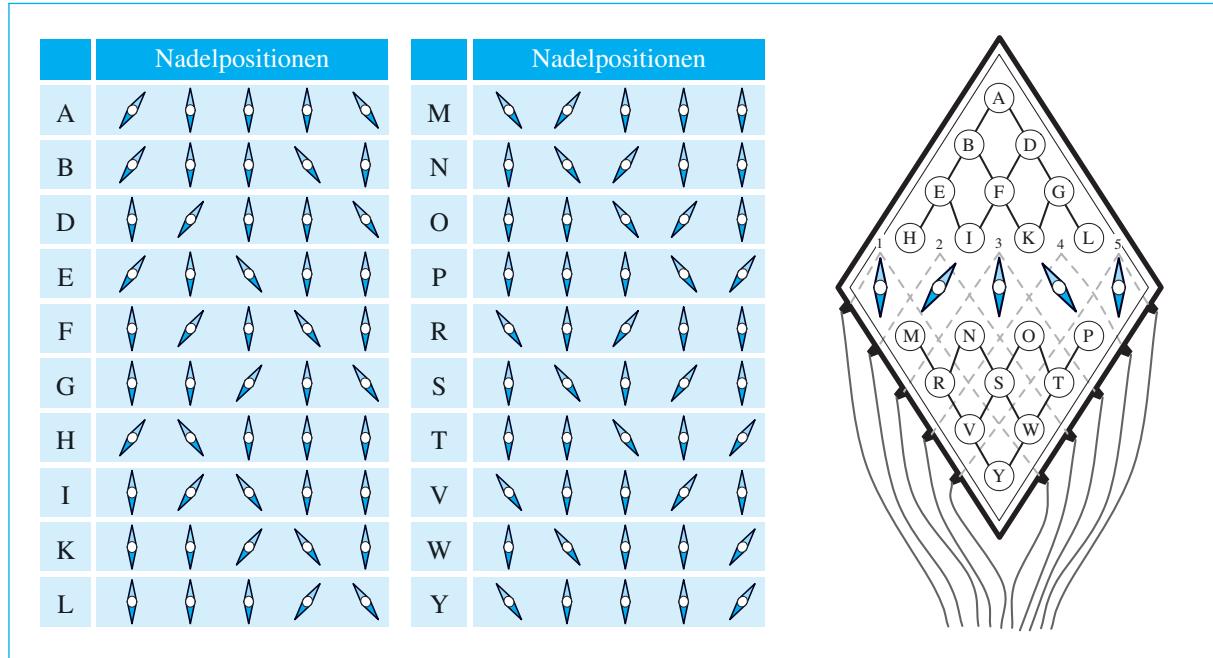


Abb. 1.19: Fünf-Nadel-Telegraf von Cooke und Wheatstone

nur noch eine einzige Nadel besaß. Mit ihm konnten Codewörter zwar nur noch sequenziell, dafür aber schneller und zuverlässiger übertragen werden. Obgleich sich die verschiedenen Varianten des Nadel- und des Zeigertelegrafen von Cooke und Wheatstone in der Praxis gut bewährten, kamen sie außerhalb der britischen Insel nicht zum Einsatz, denn den Rest der Welt sollte schon bald ein anderer Telegraf dominieren.

### 1.3.2 Der Morse-Telegraf

Bereits den fünften Tag in Folge bliesen die Winde stoisch landeinwärts, und so lag die *Sully* noch immer fest vertäut an ihrem Anlegeplatz. Doch dann, am 6. Oktober 1832, war es so weit: Die Winde hatten gedreht und machten den Weg für die so ungeduldig erwartete Atlantiküberquerung frei [18]. Die *Sully* war ein schnelles Schiff und schaffte die Reise von Le Havre nach New York in 6 Wochen. Ihre Hauptaufgabe war der Posttransport, daneben bot sie Platz für eine geringe Zahl betuchter Passagiere. An Bord war dieses Mal ein ganz besonderer Reisegast: der amerikanische Kunstreisende Samuel F. B. Morse (Abbildung 1.20).



Samuel Finley Breese Morse wurde am 27. April 1791 in Charlestown, Massachusetts, geboren. Von 1805 bis 1810 besuchte er das College in

Yale und begann anschließend eine Ausbildung als Buchhändler. Ein Jahr später änderte er seine Pläne und schrieb sich an der Royal Academy of Arts in London als Kunststudent ein. 1815 kehrte Morse in die USA zurück und schaffte es in den Folgejahren mehrfach, mit seinem künstlerischen Talent zu begeistern. Mit seinem kontinuierlich wachsenden Bekanntheitsgrad stieg auch sein Einfluss. 1835 wurde Morse von der New York University zum Professor berufen und von der New York Drawing Association zum Präsidenten gewählt. Ein Jahr später wurde er Gründungsmitglied und erster Präsident der National Academy of Design.

Der Entschluss, einen elektrischen Telegraf zu bauen, fiel im Jahr 1832 an Bord der Sully, als Morse nach einem dreijährigen Europa-Aufenthalt in die USA zurückkehrte. Für Morse hatte die Errichtung eines effizienten Kommunikationsnetzes einen ganz persönlichen Hintergrund. 1825 erkrankte seine Frau während einer seiner Dienstreisen. Die Nachricht hierüber erreichte ihn allerdings so spät, dass er nicht einmal mehr der Beerdigung beiwohnen konnte.

Sein ungewöhnliches Durchhaltevermögen und seine Fähigkeit, sich mit den richtigen Leuten zu umgeben, führten Mor-

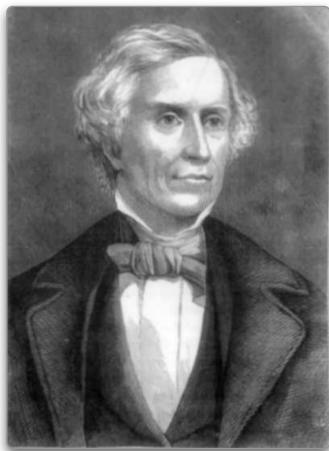
se schließlich zum Erfolg. Sein Telegraf setzte sich gegen die konkurrierenden Übertragungstechniken durch und entwickelte sich in kurzer Zeit zum weltweit führenden Kommunikationsmedium. Den Aufstieg des Telegrafen konnte Morse noch viele Jahre miterleben. Über seine Patente und Firmenbeteiligungen hatte er ein Vermögen verdient und bereits zu Lebzeiten zahlreiche Ehrungen erhalten.

Im Jahr 1871 wurde im Central Park in New York eine Bronzestatue zu Ehren Morses eingeweiht, die noch heute dort zu besichtigen ist. Ihren Höhepunkt erreichte die Einweihungsfeier um 9 Uhr Abends als eine Mitarbeiterin der Western Union Telegraph Company die folgenden Worte zu telegrafieren begann: „*Greeting and thanks to the Telegraph fraternity throughout the world. Glory to God in the Highest, on Earth Peace, Goodwill to men.*“

Dann wurde ein alter Mann unter dem Jubel der Massen an den Telegräfertisch begleitet. Er griff nach dem Sender und führte langsam, aber sicher die Nachricht zu Ende:

... - - - - - . . . . .

„S F B Morse“. Dies waren die letzten Worte, die Samuel Morse in seinem bewegten Leben telegraфиerte. 10 Monate später, am 2. April 1872, schied er im Alter von 80 Jahren aus dem Leben.



Samuel Finley Breese Morse  
(1791 – 1872)

**Abb. 1.20:** Vater der Telegrafie. Samuel Morse in jungen Jahren.

Morse hatte die USA 1829 für drei Jahre verlassen, um sich in mehreren europäischen Ländern der Kunst zu widmen. 1832 kehrte er an Bord der Sully in die USA zurück, ohne zu ahnen, dass diese Schiffsreise sein weiteres Leben von Grund auf verändern würde.

Die Tage auf der Sully waren lang, und die mehrwöchige Seereise bot genug Zeit, um mit anderen Passagieren lebhafte Diskussionen zu führen. Anders als heute war es damals keine Seltenheit, wissenschaftliche Themen zu diskutieren, und so wechselte das Tagesgespräch irgendwann auf das Gebiet der noch jungen Elektrizitätslehre. Von dem amerikanischen Wissenschaftler Charles T. Jackson erfuhr Morse in einem dieser Gespräche, dass sich Strom ohne Zeitverzögerung durch einen beliebig langen Leitungsträger hindurch bewegte. Dieses abenteuerliche Phänomen hatte Morse nachhaltig beeindruckt und entfesselten in ihm eine visionäre Idee:

„If this be so, and the presence of electricity can be made visible in any desired part of the circuit, I see no reason

*why intelligence might not be instantaneously transmitted by electricity to any distance.“*

Samuel Morse [18]

Morse wusste nicht, dass an anderer Stelle bereits fieberhaft an der Konstruktion praxistauglicher elektrischer Telegrafen gearbeitet wurde. Insbesondere wusste er nichts von den Problemen seiner Vorgänger, elektrische Signale über lange Distanzen zu übertragen. Morse war überzeugt, mit seiner visionären Idee der Erste zu sein, und wahrscheinlich war genau dies der Schlüssel zum Erfolg. Wären ihm die Probleme seiner Vorgänger bekannt gewesen, so hätte er seine Idee wohl gleich wieder verworfen.

Am 16. November 1832 erreichte die Sully den Hafen von New York. Auf dem Schiff hatte Morse noch von einer raschen Umsetzung seiner Idee geträumt, doch die Probleme des Alltags holten ihn schnell ein. Seine finanziellen Mittel waren so begrenzt, dass er die meiste Zeit damit beschäftigt war, als Maler den dringend benötigten Lebensunterhalt zu verdienen. Für den Bau seines elektrischen Telegrafens hatte er in den ersten Jahren nach seiner Rückkehr weder Zeit noch Geld.

Dies änderte sich im Jahr 1835 mit dem Ruf Morses an die New York University. Jetzt hatte er als Professor ein gesichertes Einkommen und den notwendigen Freiraum für die Umsetzung seines Projekts. In dieser Zeit entstand sein erster Prototyp, eine vergleichsweise fragile Konstruktion, die aus einer Staffelei und einem daran befestigten Stift bestand, der sich mit einem Elektromagneten hin- und herbewegen ließ (Abbildung 1.21). In seinen ersten Prototypen hatte der Stift zu jeder Zeit Kontakt mit einem Papierstreifen, der mittels eines absinkenden Gewichts langsam vorbeigezogen wurde. Durch das An- und Abschalten des Elektromagneten entstand auf dem Papier eine Zickzackkurve, wie sie in Abbildung 1.22 schematisch dargestellt ist.

In den späteren Entwürfen war der Elektromagnet so angebracht, dass er den Stift anhob. Erst diese Maschinen produzierten die charakteristischen Folgen von Punkten und Strichen, die wir heute gedanklich mit dem Begriff des Morse-Telegrafen verbinden (Abbildung 1.23).

In der ursprünglichen Sendeeinrichtung wurden die elektrischen Impulse mithilfe spezieller Zackenbretter erzeugt. Morse hatte vorgesehen, ausschließlich die Ziffern 1 bis 5 zu übertragen und die Zahlenfolgen anschließend anhand eines Codebuchs in Klartextnachrichten zu übersetzen. Als praktikabel erwies sich seine Konstruktion nicht – aber sie funktionierte.

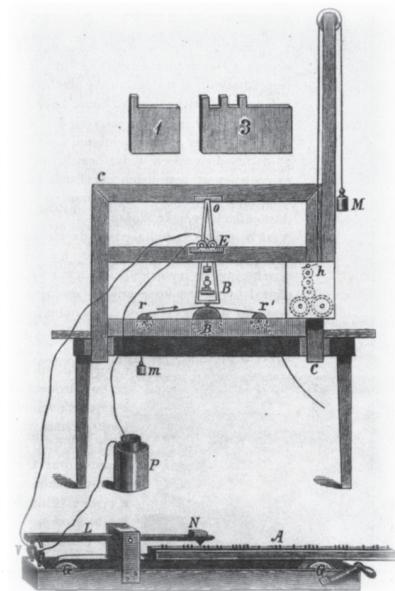


Abb. 1.21: Früher Prototyp des elektrischen Telegrafen von Samuel Morse

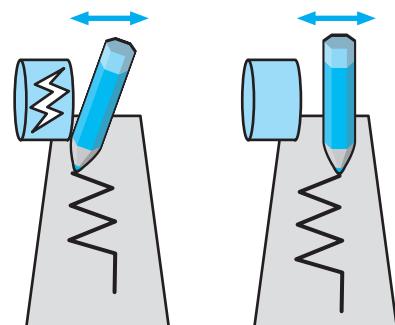
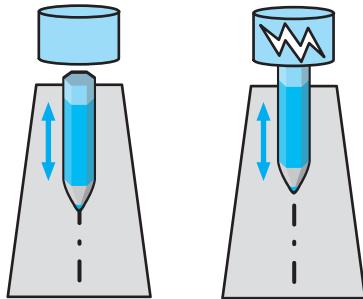


Abb. 1.22: In Morses ersten Entwürfen bestand ein permanenter Kontakt zwischen Stift und Papier. Als Ergebnis entstanden die für die frühen Morse-Telegrafen typischen Zickzackkurven.



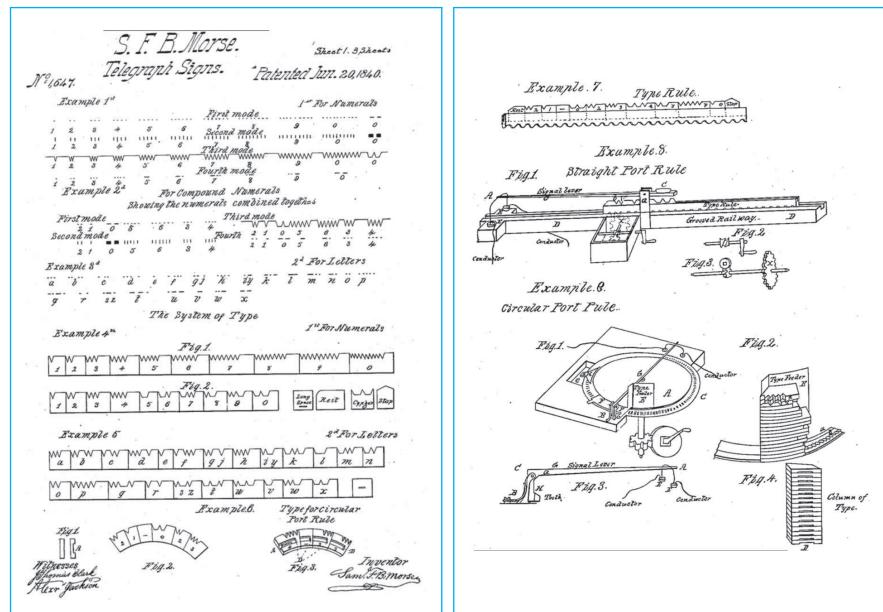
**Abb. 1.23:** Erst in den späteren Entwürfen des Morse-Telegrafen hatte man die Position des Elektromagneten so verändert, dass der Stift durch einen elektrischen Impuls angehoben wurde. Auf diese Weise entstand das typische Muster von Punkten und Strichen, das wir heute mit dem Begriff des Morse-Telegrafen verbinden.

Morses größtes Problem war ein anderes: Wider seiner Erwartung funktionierte die Übertragung nur auf kurzen Strecken, und schon ab einer Leitungslänge von 10 m quittierte der Telegraf seinen Dienst. Mit seinem begrenzten Wissen über die Eigenschaften des elektrischen Stroms war er außer Stande, das Problem selbst zu lösen.

Hilfe erhielt Morse durch den New Yorker Chemieprofessor Leonard D. Gale. Als enger Freund von Joseph Henry war dieser mit den grundlegenden Gesetzmäßigkeiten des Elektromagnetismus vertraut und hierdurch in der Lage, zwei Schwachstellen in Morses Konstruktion zu identifizieren: eine zu geringe Spannung und einen viel zu schwach ausgelegten Elektromagneten. Gales Modifikationen brachten den Erfolg. Ab jetzt ließ sich die Leitung zwischen der Sende- und der Empfangseinrichtung verlängern, ohne die Übertragungsqualität des Telegrafen negativ zu beeinflussen.

Nachdem die prinzipielle Funktionstüchtigkeit gesichert war, meldete Morse seine Apparatur am 3. Oktober 1837 offiziell als Erfindung an. Am 20. Juni 1840 wurde seinem Antrag stattgegeben und der Morse-Telegraf unter der Nummer 1647 patentiert (Abbildung 1.24).

Im September 1837 unterzeichnete Morse einen Vertrag mit dem US-amerikanischer Ingenieur Alfred Vail (Abbildung 1.25). Vails techni-



**Abb. 1.24:** Zwei Seiten aus der Patentschrift von Samuel Morse. Gut zu erkennen sind die Zickzackbretter, mit denen Morse die elektrischen Impulse zu erzeugen versuchte.



Alfred Lewis Vail wurde am 25. September 1807 in Morristown, New Jersey, geboren. Sein Vater war der Besitzer der Speedwell Ironworks company und bescherte Alfred und seinem Bruder eine Kindheit in wohlhabenden Verhältnissen. Nach dem Besuch der Schule arbeitete Vail mehrere Jahre erfolgreich im heimischen Betrieb, danach studierte er von 1832 bis 1836 Theologie an der New York University.

Wegweisend für Vails weiteres Leben war eine Demonstration des Morse-Telegrafen, der er im Jahr 1837 beiwohnte. Vail war von der technischen Konstruktion begeistert und traf im September des gleichen Jahres mit Morse eine vertragliche Vereinbarung, die für die weitere Entwicklung des Telegrafen eine entscheidende Rolle spielte. Vail sicherte zu, den prototypischen Telegrafen auf eigene Kosten weiterzuentwickeln und die notwendigen Teile in der väterlichen Firma zu fertigen. Im Gegenzug sollte er an den erzielten Erlösen beteiligt werden.

Die Partnerschaft zwischen Vail und Morse trug alsbald Früchte. Der entwickelte Telegraf konnte technisch überzeug-

gen, und seine Handhabung war einfach genug, um ihn als Massenmedium etablieren zu können.

Die Partnerschaft zwischen Vail und Morse geriet mit der zunehmenden Verbreitung des Telegrafen allerdings mehr und mehr in eine Schieflage. Morse profitierte nicht nur finanziell in überproportionalem Maße; auch in der Öffentlichkeit wurde der Erfolg fast ausschließlich ihm zugeschrieben. Vail machte diese Situation schwer zu schaffen und so beschloss er vier Jahre nach der erfolgreichen Inbetriebnahme der Washington-Baltimore-Linie, aus dem Projekt auszusteigen. Am 5. Oktober 1848 begründete er seine Entscheidung in einem Brief an seinen Bruder mit den folgenden Worten:

*„The reason why I must give up remaining here is, that I am wearing myself out in the telegraph, for the interest of the patentees, without compensation, and the care and study is accumulating every day.“ [18]*

Vail zog zurück nach Morristown, wo er den Rest seines Lebens verbrachte. Er starb am 18. Januar 1859 im frühen Alter von 51 Jahren.

sches Geschick war außergewöhnlich, und erst durch ihn war Morse in der Lage, den nächsten Schritt zu verwirklichen: die Weiterentwicklung seines Prototyps zu einem marktreifen Produkt.

Eine von Vails ersten Maßnahmen war es, Morses ursprünglichen Entwurf zu vereinfachen. Die komplizierte Sendeeinrichtung wurde durch einen einfachen Handschalter ersetzt, und auch die Idee des Codebuchs war schnell verworfen. Im Jahr 1838 entstand der erste Morse-Code, der eine Nachricht buchstabenweise repräsentierte.

1843 erhielt Morse die große Chance, seinen Traum real werden zu lassen. In diesem Jahr stellte ihm der US-amerikanische Kongress 30.000 \$ für den Bau einer Telegrafenlinie zwischen Washington, D.C., und dem ca. 60 km entfernten Baltimore bereit. Morse hatte zunächst vor, das Kabel unterirdisch zu verlegen, doch schon bald erlitt er den ersten Rückschlag. Die verfügbare Kabelisolierung war so minderwertig, dass bereits nach 9 Meilen keine Übertragung mehr möglich war. Notgedrungen wurden die Leitungen schließlich als Überlandkabel verlegt. Am 24. Mai 1844 hatten Morse und seine Partner das Projekt zu einem erfolgreichen Ende gebracht. Die Washington-Baltimore-Linie war fertiggestellt und nahm offiziell ihren Betrieb auf. Das erste Telegramm hatte ein Zitat aus der Bibel zum Inhalt; es beinhaltete die berühmten Worte:



Alfred Lewis Vail (1807 – 1859)

**Abb. 1.25:** Auch wenn Alfred Vail in der Öffentlichkeit stets in der zweiten Reihe stand, ist er der eigentliche Vater des Morse-Telegrafen. Er entwickelte den Prototyp von Morse und Gail fast im Alleingang zu jenem Apparat weiter, der als *Morse-Telegraf* die Welt veränderte.



Bis heute sind sich die Experten uneinig darüber, wie viel Samuel Morse zu seiner Erfindung selbst beigetragen hat.

Unstrittig ist, dass Morse weder fundierte Kenntnisse im Bereich der Elektrizitätslehre hatte noch die Fähigkeit besaß, seinen prototypischen Entwurf zu einem industriellen Produkt weiterzuentwickeln. Die notwendige Fachexpertise wurde zu großen Teilen von Leonard Gale beigesteuert, der bestens mit den wissenschaftlichen Arbeiten des Physikers Joseph Henry vertraut war. Im Rahmen späterer Patentstreitigkeiten äußerte sich Henry folgendermaßen [18]:

*„I am not aware that Mr. Morse ever made a single original discovery in electricity, magnetism, or electromagnetism, applicable to the invention of the telegraph.“ [18]*

Auch der zurückhaltende Vail sprach in späteren Jahren kritisch über die Zusammenarbeit:

*„Whenever there is work to be done, the professor is taken ill.“ [18]*

Es gilt als wahrscheinlich, dass Alfred Vail nicht nur die finale Apparatur fast im Alleingang konstruierte, sondern auch das letztlich verwendete Morse-Alphabet ersann [48]. Dass wir hierfür kaum schriftliche Beweise in Händen halten, geht auf die vertragliche Vereinbarung zwischen Vail und Morse vom 23. September 1837 zurück. Dort wurde nicht nur vereinbart, dass Vail für seine Arbeit an den Erlösen beteiligt sein würde, sondern auch, dass sämtliche Erfindungen unter Morses Namen veröffentlicht werden.

War Morse lediglich die Galionsfigur eines Projekts, das im Hintergrund von anderen getragen wurde? Nicht ganz. Morses Charakter war von einem Maß an Ehrgeiz, Durchhaltevermögen und visionärer Tatkraft geprägt, wie es nur wenige Menschen besitzen. Neben der wissenschaftlichen Fachkenntnis von Gale und der industriellen Expertise von Vail hatte es einer charismatischen Persönlichkeit wie Morse bedurft, um ein Projekt dieses Ausmaßes zum Erfolg zu führen. So unterschiedlich die Beiträge von Gail, Vail und Morse auch waren: Hätte es auch nur einen der drei nicht gegeben, so wäre dem Morse-Telegrafen der Siegeszug um die Welt höchstwahrscheinlich verwehrt geblieben.

*„What hath God wrought!“ (Numbers, 23:23)*

In der Presse wurde Morses Erfolg bejubelt und tatsächlich bedeutete die gebaute Telegrafenlinie eine kleine Revolution für die Nachrichtenübertragung in den USA. Zu jener Zeit wurden immer noch 90 % der Nachrichten von Reiterkurieren oder Postkutschen transportiert [45].

Codiert war die Nachricht im *amerikanischen Morse-Code* (Abbildung 1.26 links). Die Codetabelle, die aus einer Modifikation des Codes von 1938 hervorging, wurde im Jahr 1944 verabschiedet und fortan für die Nachrichtenübertragung in allen US-amerikanischen Telegrafennetzen verwendet. Außerhalb der USA kam ein geringfügig modifizierter Code zum Einsatz, der 1851 in Berlin festgeschrieben wurde. Er ist in der rechten Hälfte von Abbildung 1.26 zu sehen und wird als *internationaler* oder *kontinentaler Morse-Code* bezeichnet. Obwohl es von nun an einen offiziellen Standard gab, lebte der ältere amerikanische Code weiter. In den USA war er schon zu lange im Einsatz, als dass ihn der internationale Code noch hätte verdrängen können.

Der Bau der Washington-Baltimore-Linie war der Startschuss für die kommerzielle Nutzung der Telegrafie. Am 1. April 1845 öffnete die erste öffentliche Telegrafenstation in Washington ihre Türen, und am 15. Mai 1845 gründete Morse mit seinen Geschäftspartnern die *Magne-*

Amerikanischer Morse-Code				Internationaler Morse-Code			
A	- -	M	- - -	Y	.. ..		
B	- - - -	N	- -	Z	... .		
C	.. .	O	. .	1	- - - -		
D	- - -	P	.....	2	- - - .		
E	.	Q	- - - -	3	.... -		
F	- - -	R	. . .	4	.... - -		
G	- - -	S	...	5	- - - -		
H	....	T	-	6	.....		
I	..	U	- -	7	- - - .		
J	- - - -	V	.... -	8	- - - - -		
K	- - -	W	- - - -	9	- - - - .		
L	- -	X	- - - -	0	- - - - -		

Abb. 1.26: Der amerikanische und der internationale Morse-Code im Vergleich

tic Telegraph Company. Im Jahr 1850 konkurrierten bereits 20 Firmen um die Baugenehmigungen der profitabelsten Linien. Ab jetzt breitete sich das Telegrafennetz in immer größerem Tempo über den Kontinent aus, und im Jahr 1861 wurde die erste transkontinentale Telegrafenlinie durch die *Western Union Telegraph Company* fertiggestellt.

Der zunehmende Ausbau des Netzes führte zur Entstehung von Kommunikationsstrukturen, die an die klassische Briefpost erinnerten. Es gab eine Reihe großer Vermittlungszentren, die das Umland sternförmig mit Telegrafenlinien versorgten. An jede Linie waren kleinere Telegrafenstationen angebunden, ähnlich den Perlen einer Schnur. Gehörten zwei Telegrafenstationen der gleichen Linie an, konnten sie direkt miteinander kommunizieren. Um sich zu identifizieren, besaß jede Station eine eindeutige Kennung, die aus ein oder zwei Buchstaben bestand. Sollte eine Nachricht an eine Station gesendet werden, die nicht auf derselben Linie lag, wurde sie zunächst an das zugehörige Vermittlungszentrum (*relais office*) geschickt. Von dort aus gelangte die Nachricht über eine Direktverbindung (*intercity cable*) zum zuständigen Vermittlungszentrum des Empfängers und wurde anschließend in eine der lokalen Telegrafenlinien eingespeist.

Mit der zunehmenden Verbreitung der Telegrafie wurden die Relaisstationen immer wichtiger und wuchsen zu riesigen Zentren heran. Beispielsweise arbeiteten um das Jahr 1900 im Chicagoer Relaiszentrum der Western Union Telegraph Company knapp 900 Telegrafisten gleichzeitig an der Vermittlung der ein- und ausgehenden Telegramme [18].



Cyrus West Field wurde am 30. November 1819 in Stockbridge, Massachusetts, geboren. Im Alter von 15 Jahren verließ er die Schule und arbeitete in Stockbridge und New York als Kaufmann. In den Folgejahren gelang ihm das, wovon viele Amerikaner träumen: Von einer schlecht bezahlten Hilfskraft in A. T. Stewart's New York department store schaffte er es, zu einem der erfolgreichsten Geschäftsmänner der Stadt aufzusteigen. Der Handel mit Papier bescherte ihm ein Vermögen, und bereits im jungen Alter von 34 Jahren gehörte Field zu den reichsten Männern von New York.

1854 sollte sein Leben eine Kehrtwende nehmen. Nach einem Treffen mit Frederic N. Gisborne fasste Field den Entschluss, das scheinbar Unmögliche zu versuchen: die Verlegung eines Unterwasserkabels zwischen Amerika und Europa. In den Jahren 1854 bis 1865 musste Field unzählige Rückschläge verkraften, doch sein unerschütterlicher Ehrgeiz führte ihn am Ende zum Erfolg.

Ab dem Jahr 1866 verbanden zwei funktionierende Telegrafenleitungen die Kontinente und machten Field über Nacht zu einem gefeierten Mann. Auch finanziell entwickelte sich seine Zukunft prächtig. Die transatlantischen Telegrafenlini-

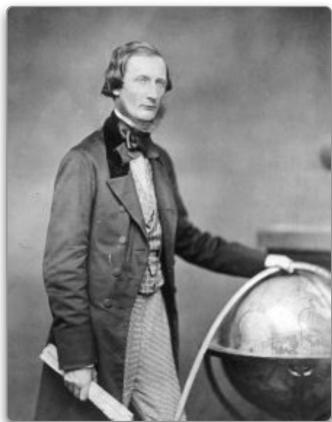
en waren von Beginn an profitabel und bescherten ihm und seinen Investoren ein Vermögen.

Nach der erfolgreichen Verlegung des Transatlantikkabels wandte sich der rastlose Field neuen Aufgabenfeldern zu. Er kaufte sich in die *Western Railroads Company* und die *New York Elevated Railroad Company* ein und verdiente zunächst gutes Geld. Danach traf Field eine folgenschwere Fehlentscheidung. Er investierte einen großen Teil seines Vermögens in Anteile an der *Manhattan Elevated Railroad*, von denen er sich nach einer verlorenen Übernahmeschlacht mit hohen Verlusten trennen musste. Finanziell ruiniert, verbrachte er seinen Lebensabend als gebrochener Mann.

Cyrus Field starb in der Nacht vom 11. auf den 12. Juli 1892 im Alter von 72 Jahren. Er wurde im Familiengrab in Stockbridge beigesetzt, wo ein Grabstein leise an einen Mann erinnert, der uns nicht nur durch seine historische Tat, sondern auch durch seinen beispiellosen Mut, seine Entschlossenheit und sein Durchhaltevermögen für immer in Erinnerung bleiben wird [18]:

*Cyrus West Field*  
To whose courage, energy and perseverance  
The world owes the Atlantic telegraph.

## 1.4 Mission Transatlantik



Cyrus West Field (1819 – 1892)

**Abb. 1.27:** Mit der Verlegung des ersten Transatlantikkabels durch Cyrus W. Field begann der Aufbau weltumspannender Kommunikationsnetze.

Die rasche Verbreitung des Morse-Telegrafen sorgte für einen nachhaltigen Umbruch in der inländischen Nachrichtenübertragung. Botschaften erreichten ihren Empfänger schneller als jemals zuvor, und nach und nach gewöhnte sich das politische, wirtschaftliche und gesellschaftliche Leben an die neue Form der Kommunikation. Eines konnte der Morse-Telegraf bis dato aber nicht beschleunigen: den Nachrichtenverkehr zwischen den Kontinenten. Noch führte kein Weg daran vorbei, Briefe oder Telegramme mit Dampfschiffen auf dem Seeweg zu transportieren.

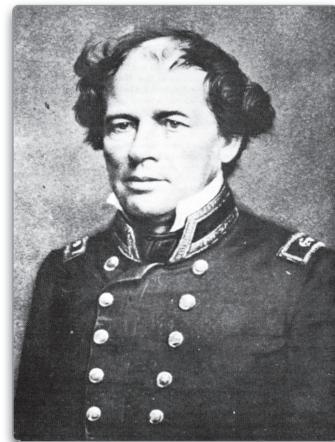
Von einer transatlantischen Telegrafenlinie träumten viele, doch kaum jemand erachtete ein solches Projekt als umsetzbar; für die meisten war die Verlegung eines Unterwasserkabels in den Tiefen des Ozeans nichts weiter als utopisches Wunschedenken, fern ab der Realität. Rückblickend ist es schwer zu ermessen, welch hohes Maß an Tatendrang, Ehrgeiz und Kühnheit ein Mensch besessen haben musste, um sich einer solchen Aufgaben anzunehmen; und dennoch gab es jemanden, der genau dies Tat: Cyrus W. Field (Abbildung 1.27).

Mit der Verlegung des ersten Transatlantikkabels hat der junge Amerikaner ein bewegendes Kapitel der Telekommunikationsgeschichte geschrieben. Wie kaum ein anderes ist es gezeichnet von unbändiger Abenteuerlust, ruhmhaften Erfolgen und zerschmetternden Niederlagen.

Die Geschichte der transatlantischen Telegrafie begann 1854 in New York, als sich die Wege von Cyrus Field und Frederic N. Gisborne kreuzten (vgl. [61]). Gisborne, ein englischer Ingenieur, hatte erste Erfolge mit der Verlegung von Unterwasserkabeln zwischen New Brunswick und Prince Edward's Island vorzuweisen und war auf der Suche nach neuen Geldgebern. Field war ein erfolgreicher Kaufmann und hatte bereits in jungen Jahren ein Vermögen verdient. Mit der Telegrafie hatte er sich in der Vergangenheit kaum beschäftigt, doch das Treffen mit Gisborne sollte dies nachhaltig ändern. Für Field stand fest: Wenn es möglich war, ein Unterwasserkabel durch den Golf von Sankt-Lorenz zu legen, warum sollte es dann nicht gelingen, ein Unterwasserkabel bis nach Europa zu führen? Ab diesem Tag hatte Field ein klares Ziel vor Augen, nichtsahnend, welch steinigen Weg er hier betrat.

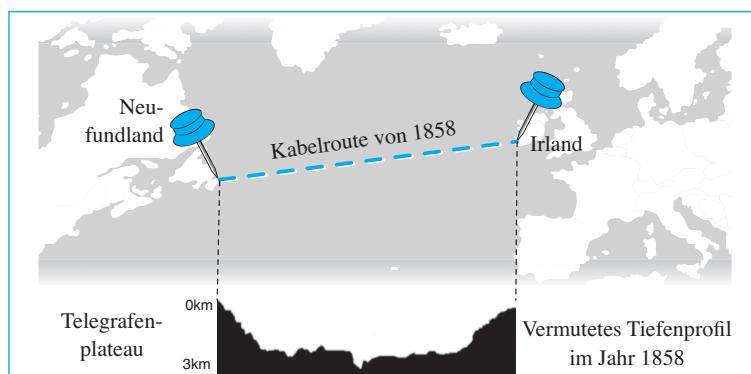
Für die Planung seines ehrgeizigen Projekts griff Field auf die Vermessungsarbeiten des Hydrografen Matthew F. Maury zurück, der in den Jahren 1852 und 1853 zahlreiche Tiefenmessungen auswertete und dabei ein riesiges Plateau unter der Meeresoberfläche entdeckt hatte (Abbildung 1.28). Die Hochebene, die wir heute als *Telegrafenplateau* (*telegraph plateau*) bezeichnen, verbindet Neufundland mit der irischen Küste in einer Tiefe von ca. 1000 m bis 3000 m und war für die Verlegung eines Telegrafenkabels wie geschaffen (Abbildung 1.29).

Genauso wichtig war für Field das wenige Jahre zuvor entdeckte *Gutapercha*. Das kautschukähnliche Material wird aus dem Milchsaft des



Matthew Fontaine Maury  
(1806 – 1873)

**Abb. 1.28:** Matthew F. Maury war ein hochrangiger Offizier der US-Marine und Leiter der ozeanografischen Gesellschaft. Die von ihm erstellten Tiefenprofile des atlantischen Ozeans waren für die Verlegung des ersten transatlantischen Unterseekabels von zentraler Bedeutung.



**Abb. 1.29:** Seit Mitte des 19. Jahrhunderts wissen wir, dass zwischen der neufindländischen und der irischen Küste eine Hochebene verläuft, die an ihrer tiefsten Stelle nur ca. 3 km unter der Wasseroberfläche liegt. Heute wird diese Hochebene als *Telegrafenplateau* (*telegraph plateau*) bezeichnet, in Anerkennung ihrer historischen Bedeutung.

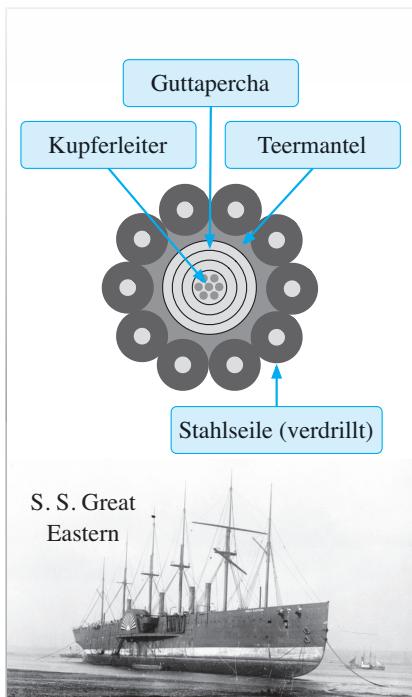
im südostasiatischen Raum beheimateten Guttaperchabaums gewonnen und verfügt über zwei Eigenschaften, die für die Produktion von Unterwasserkabeln unabdingbar sind. Zum einen ist es ein guter Isolator, der auch bei tiefen Temperaturen seine kautschukartige Konsistenz behält. Zum anderen ist das Material insbesondere dann lange haltbar, wenn es in Salzwasser gelagert wird. Mit diesen Eigenschaften war Guttapercha der perfekte Grundstoff für die Isolation des empfindlichen Metallkerns eines Unterwasserkabels.

Trotz der günstigen Randbedingungen gestaltete sich die Umsetzung des Projekts schwieriger als gedacht. Field hatte geplant, zunächst New York und Neufundland durch eine Telegrafenleitung zu verbinden, danach sollte die Verlegung des transatlantischen Kabels erfolgen. Doch permanente Verzögerungen führten dazu, dass die Telegrafenlinie nach Neufundland viel später fertig wurde als geplant und bereits zwei Drittel der Geldreserven verschlang.

Im Juli 1857 gab es positive Nachrichten: Das rund 3000 km lange und 2500 Tonnen schwere Unterseekabel war fertiggestellt und bereit zum verladen. Aufgrund seines enormen Gewichts wurde es in der Mitte aufgetrennt und eine Hälfte auf die H.M.S. Agamemnon und die andere Hälfte auf die U.S.S. Niagara gebracht. Mit den Kabelhälften an Bord verließen beide Schiffe am 5. August 1857 die irische Küste. Der Verlauf der Mission war akribisch geplant: Zunächst würde die Niagara ihre Kabelhälfte ins Meer versenken, danach sollten die Kabel der beiden Schiffe in der Mitte des Atlantiks verspleißt werden und die Agamemnon die weitere Verlegung bis zur neufundländischen Küste übernehmen. Doch alles kam anders. Am 10. Tag riss das Kabel und versank nach 330 zurückgelegten Seemeilen unwiederbringlich in den Tiefen des Atlantiks.

Bereits im Juni 1858 unternahm Fields einen zweiten Versuch. Dieses Mal sollte das Kabel zuerst in der Mitte des Atlantiks verspleißt werden. Anschließend würde die Niagara eine der Kabelhälften ostwärts an die irische Küste und die Agamemnon die andere Hälfte westwärts an die neufundländische Küste bringen. Nach mehreren Fehlversuchen wägte sich Field im Glück. Am 4. August erreichte die Niagara Neufundland und die Agamemnon einen Tag später das irische Valentia. Zum allerersten Mal waren das amerikanische und das europäische Telegrafen- netz durch ein funktionstüchtiges Kabel miteinander verbunden.

Die Öffentlichkeit reagierte euphorisch auf die gelückte Verlegung, doch die Freude währte nicht lange. Bereits wenige Tage nach der Anlandung nahm die Qualität der empfangenen Signale kontinuierlich ab. Inmitten der verzweifelten Versuche, das Kabel zu retten, traf Fields



**Abb. 1.30:** Querschnitt des Transatlantikkabels aus dem Jahr 1865. Im Innern befanden sich 7 Kupferleitungen, die von vier Schichten aus Guttapercha und einem zusätzlichen Teermantel umgeben waren. Die äußere Hülle wurde durch 10 verdrillte Stahlseile gebildet. Sie schützte den empfindlichen Kern und verlieh dem Kabel die notwendige Zugfestigkeit.

Ingenieur Wildman Whitehouse eine folgenschwere Entscheidung. Im Glauben, die Empfangsqualität durch eine Erhöhung der angelegten Spannung zu verbessern, setzte er das Unterseekabel einer immer stärkeren Belastungen aus. Die Folgen waren desaströs: Das Kabel verschmolte auf dem Meeresgrund und war für immer zerstört.

Die Presse reagierte mit Häme und Spott auf den Verlust, und so manch kritischer Geist stellte gar die Frage, ob überhaupt ein Kabel verlegt worden sei; von der *Great Atlantic Bubble* war die Rede.

Positive Schlagzeilen machten derweil andere. Im Oktober 1861 verkündete die Western Union Telegraph Company die Fertigstellung der US-amerikanischen Transkontinentallinie. Von nun an waren die Ost- und die Westküste mit einer durchgängigen Telegrafenlinie verbunden. Für Field wurde die Transkontinentallinie zur Bedrohung, denn Western Union hatte Pläne, sie über den asiatischen Kontinent nach Europa zu verlängern. Der amerikanische und der asiatische Kontinent sind nur durch die Beringstraße getrennt, eine knapp 85 km messende Meerenge mit einer geringen Wassertiefe von 30 bis 50 Metern.

Die Pläne von Western Union gaben neue Hoffnung, das amerikanische Telegrafennetz mit dem europäischen doch noch zu vereinen, und vertieften gleichsam die Überzeugung jener, die eine Verbindung über den Seeweg als aussichtslos erachteten. Am 26. Oktober 1861 schrieb z. B. die New York Times [18]:

*„If there is ever to be electric communication with Europe, it will be by imitating the splendid example the United States has thus given in our transcontinental line. The bubble of the Atlantic submarine line has long ago burst, and it is now seen to be cheaper and more practicable to extend a wire over five-sixths of the globe on land, than one-sixth at the bottom of the sea.“*

Im Jahr 1865 war der Bau der russisch-amerikanischen Telegrafenlinie in vollem Gange. Im Auftrag von Western Union arbeiteten sich mehrere Bautrupps in British Columbia, Alaska, Sibirien und der Beringstraße unabhängig voneinander in Richtung Europa vor.

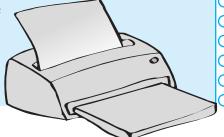
Zur gleichen Zeit unternahm Field einen dritten Versuch, die Telegrafenlinie über den Seeweg zu verbinden. Dieses Mal wollte er mit einem neu entwickelten Kabel den Erfolg erzwingen: Es war aus einem Stück gefertigt und dicker und stabiler als das erste (Abbildung 1.30). Auf den Einsatz der Niagara und der Agamemnon musste Field ab



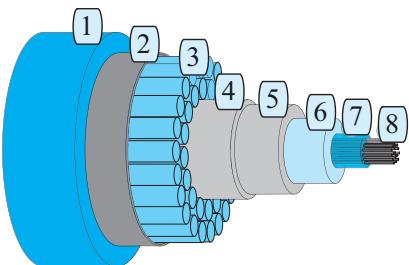
**Abb. 1.31:** Zwei Brücken über den Bulkley River im kanadischen British Columbia, gebaut aus den hinterlassenen Materialien der russisch-amerikanischen Telegrafenlinie [103].

**Beschädigtes Unterseekabel bremst Internetverbindungen in die USA**

23.03.2008: heise online liegen Informationen vor, dass alleine der Deutschen Telekom zwischenzeitlich bis zu 30 GBit/s Datentransferrate in die USA weggebrochen sind. Schuld daran soll ein Defekt im Unterseekabel TAT-14 sein. Der Schaden wird in der Gegend von Calais vermutet. Derzeit kann wetterbedingt kein Wartungsschiff auslaufen. So dauert es möglicherweise bis zu einer Woche, bis TAT-14 wieder seine volle Bandbreite erreicht.



**Abb. 1.32:** Kabelbrüche werden heute nur noch beiläufig erwähnt [68]. Die Reparatur beschädigter Verbindungen ist zu einer Routinetätigkeit geworden.



1. Polyethylen
2. Mylar
3. verdrillte Stahlseile
4. Aluminium-Wasserbarriere
5. Polykarbonat
6. Kupfer- oder Aluminiumrohr
7. Paraffin
8. Lichtwellenleiter

**Abb. 1.33:** Aufbau eines modernen Unterwasserkabels (nach [104])

jetzt verzichten, da das neue Kabel mit einem Gesamtgewicht von rund 7000 Tonnen für beide Schiffe zu schwer war. Verlegt werden sollte es mit der *Great Eastern*, dem größten Dampfschiff der damaligen Zeit. Mit dem Kabel an Bord verließ der Schiffskoloss im Juli 1865 das irische Valentia, doch auch dieses Mal war Field vom Pech verfolgt: Rund 1000 km vor der neufundländischen Küste riss das Kabel und verschwand in den Tiefen des Ozeans.

Ein Jahr später wagte Field einen vierten und letzten Versuch. Dieses Mal verließ die Reise ohne nennenswerte Vorkommnisse, und nach einer 15-tägigen Fahrt erreichte die Great Eastern am 27. Juli 1866 die Küste Neufundlands. Für Field war die Anlandung des Kabels der Beginn einer Glückssträhne. Im August gelang es, das ein Jahr zuvor verloren geglaubte Kabel zu bergen und mit einem neuen Kabelstück zu verspleißen. So kam es, dass im September 1866 zwei vollständig funktionsfähige Telegrafenleitungen die Kontinente verbanden. Fields unbändige Beharrlichkeit hatte sich am Ende ausgezahlt: Mit der erfolgreichen Verlegung des ersten transatlantischen Unterseekabels errang er einen der größten Erfolge in der Geschichte der Kommunikationstechnik und legte damit den Grundstein für die globale Vernetzung der Welt.

Die Western Union Telegraph Company schien dagegen vom Glück verlassen. Der Bau der Telegrafenlinie in den sibirischen Permafrostböden gestaltete sich schwieriger als erwartet, und nur selten kamen die Bautrupps in der geplanten Geschwindigkeit voran. Die Nachricht über Fields Erfolg kam für Western Union daher zu einem denkbar ungünstigen Zeitpunkt. Die erfolgreiche Verlegung der Unterwasserkabel stellte den wirtschaftlichen Nutzen des sibirischen Kabels in Frage, und so entschloss man sich im Juli 1867, den Bau offiziell für beendet zu erklären. Die Leitungen und Masten wurden sich selbst überlassen und dienten den ansässigen Ureinwohnern in der Folgezeit als willkommenes Baumaterial (Abbildung 1.31).

Heute verläuft die Verlegung eines Unterwasserkabels weit weniger spektakulär als zu Fields Zeiten. Dank satellitengestützter Navigation gelingt die Positionierung metergenau, und die Reparatur eines Kabelbruchs ist zu einer Routinetätigkeit geworden (Abbildung 1.32). Auch die heute verlegten Kabel haben mit Fields historischen Pendants nicht mehr viel gemeinsam (Abbildung 1.33). Anstelle metallischer Leiter besitzen moderne Unterwasserkabel einen kunststoffummantelten Glasfaserkern, und auch das damals so unabdingbare Guttapercha hat schon lange ausgedient. Eklatant gestiegen sind hingegen die Datenraten. Amiéti/AEC-3, das bis dato modernste Transatlantikkabel, ist in der Lage, mehr als 300 Terabit pro Sekunde durch den Atlantik zu leiten.

## 1.5 Von der Telegrafie zur Telefonie

Unbestritten hatte der Morse-Telegraf im Laufe des 19. Jahrhunderts die Welt verändert. Das Versenden von Telegrammen war Teil des Alltags geworden, und auch an die unzähligen Masten und Leitungen, die zur damaligen Zeit das Stadt- und Landbild prägten, hatte man sich längst gewöhnt; sie waren die Lebensadern einer neuen Zeit. Um den kontinuierlich wachsenden Kommunikationsbedarf zu decken, versuchten die Netzbetreiber zweierlei. Zum einen trieben sie vehement den Ausbau neuer Telegrafenlinien voran, zum anderen suchten sie parallel nach Möglichkeiten, um die bestehenden Netze effizienter und kostengünstiger zu nutzen.

Auf reges Interesse stießen neu entwickelte Apparaturen, die leichter zu bedienen waren als der ursprüngliche Morse-Telegraf. Wichtige Vertreter waren der *ABC-Telegraf* von Charles Wheatstone und der *Typendrucktelegraf* (*Hughes printer*) von David Edward Hughes (Abbildung 1.34). Hughes' Erfindung war die erste Apparatur, die empfangene Nachrichten im Klartext ausgeben konnte. Auch das Senden wurde einfacher. Die Buchstaben einer Nachricht ließen sich bequem über eine Klaviatur mit 28 zweireihig angeordneten Tasten eingeben und mussten nicht mehr länger händisch in einen Binärkode umgesetzt werden.

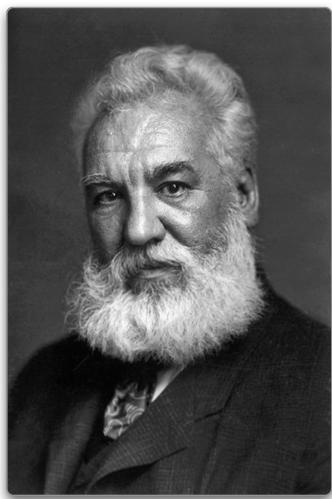
Etwa zur gleichen Zeit entwickelte Wheatstone einen Apparat, der die zu sendenden Morse-Signale von einem vorgestanzten Band ablesen konnte. Seine Erfindung beschleunigte die Übertragung von Morse-Nachrichten um den Faktor 10 und war ein wichtiger Schritt, hin zu einer wirtschaftlicheren Nutzung des Telegrafennetzes.

Neue Erkenntnisse auf dem Gebiet der Elektrotechnik führten ebenfalls zu einer effizienteren Nutzung der Telegrafenleitungen. Ein wichtiger Meilenstein war die Beobachtung, dass sich die Sende- und die Empfangsgeräte so modifizieren ließen, dass über ein und dieselbe physikalische Leitung gleichzeitig in beide Richtungen gesendet werden konnte. Die Entdeckung des *Duplexbetriebs* führte auf einen Schlag zu einer Verdopplung der Leitungskapazität, und eine entsprechende Apparatur wurde 1872 von Joseph B. Stearns zum Patent angemeldet.

Im Jahre 1874 gelang Thomas Alva Edison der Bau eines *Quadruplex-Telegrafen*, der vier Signale gleichzeitig über eine einzige physikalische Leitung übertragen konnte. In jede Richtung wurde eines der Signale durch eine Änderung der Signalamplitude (*Amplitudenmodulation*) und das andere Signal durch eine Verschiebung des Phasenwinkels (*Phasenmodulation*) codiert.



**Abb. 1.34:** Der Typendrucktelegraf von David Edward Hughes. Um die Bedienung des Telegrafen zu vereinfachen, wurde der primitive Morse-Taster durch eine Klaviatur und der Bandschreiber durch einen automatischen Zeichendrucker ersetzt.



Alexander Graham Bell  
(1847 – 1922)

**Abb. 1.35:** Er ging aus der historischen Patentschlacht um die Erfundung des Telefons als Sieger hervor: Alexander Graham Bell.



Thomas Augustus Watson  
(1854 – 1934)

**Abb. 1.36:** Als Assistent von Graham Bell spielte Thomas Watson eine bedeutende Rolle in der Erfundung des Telefons.

### 1.5.1 Suche nach dem harmonischen Telegrafen

Von den Anfangserfolgen motiviert, suchten Ingenieure intensiv nach Möglichkeiten, um die Anzahl der gleichzeitig übertragbaren Signale weiter zu erhöhen. Viele sahen die Lösung des Problems in der Konstruktion eines *harmonischen Telegrafen*, der über das Prinzip der Frequenzmodulation mehrere Morse-Datenströme parallel übertragen sollte. Im Kern stand die Idee, auf der Senderseite mehrere Stimmgabeln in Schwingungen zu versetzen und die entstehenden Wechselströme mit den binären Datenströmen der Morse-Nachrichten zu überlagern. Durch die Verwendung unterschiedlicher Frequenzen müssten sich die Signale dann störungsfrei über eine einzige Leitung übertragen lassen. Sollte es zudem gelingen, die frequenzmodulierten Signale auf der Empfängerseite wieder sauber voneinander zu trennen, so wäre das Ziel erreicht: die gleichzeitige Übertragung mehrerer Morse-Datenströme auf ein und derselben physikalischen Leitung.

#### Alexander Graham Bell

Zu den Ersten, die sich der Herausforderung annahmen, gehörte der in Schottland geborene Taubstummenlehrer Alexander Graham Bell (Abbildung 1.35). 1871 lernte er in Boston zwei Menschen kennen, die seinen Lebensweg entscheidend beeinflussten: Thomas Sanders, Vater eines Schülers, und Gardiner Greene Hubbard, sein späterer Schwiegervater. Am 27. Februar 1875 trafen die drei eine schriftliche Vereinbarung über die Entwicklung eines harmonischen Telegrafen. Bell sollte die Apparatur bauen, im Gegenzug würden Sanders und Hubbard die Finanzierung des Projekts zu jeweils 50 % übernehmen. Unterstützt wurde Bell durch Thomas A. Watson, einen ungewöhnlich begabten Ingenieur, dessen Rolle an jene von Alfred Vail erinnert (Abbildung 1.36). Wie Vail, dem es nie recht gelang, aus dem Schatten von Samuel Morse herauszutreten, war auch Watson ein Mann der zweiten Reihe; und genau wie Vail war auch Watson maßgeblich am Anteil des Erfolgs seines Frontmanns beteiligt. Heute sind sich die meisten Experten darüber einig, dass Bell ohne Watsons Hilfe kaum imstande gewesen wäre, seine visionären Ideen zu verwirklichen [19].

Der Ehrgeiz, mit dem Bell und Watson die Konstruktion des harmonischen Telegrafen vorantrieben, wurde alsbald belohnt. Bereits nach wenigen Wochen konnten die beiden eine prototypische Apparatur vorweisen, mit der sich mehrere Morsezeichen simultan übertragen ließen. Dabei machten sie eine faszinierende Entdeckung. Ihr Telegraf war für die



Alexander Graham Bell erblickte am 3. März 1847 im schottischen Edinburgh das Licht der Welt. Als Kind wurde er zunächst von seiner Mutter zu Hause unterrichtet. Im Alter von 10 Jahren wechselte er auf eine Privatschule in Edinburgh und besuchte ab dem 14. Lebensjahr eine Schule in London. Anschließend studierte er in seiner Geburtsstadt Griechisch und Latein. Nach seinem Studium nahm Bell eine Assistentenstelle im University College in London an, wo sein Vater lehrte und forschte. In dieser Zeit begann er, sich intensiv mit der Physiologie des menschlichen Sprachapparats zu beschäftigen. 1868 ereilte die Familie Bell der erste Schicksalsschlag, als Grahams Bruder Edward einer Tuberkuloseinfektion erlag. Nachdem 1870 auch sein zweiter Bruder Melville der Tuberkulose zum Opfer viel, beschloss die Familie, noch im selben Jahr nach Kanada überzusiedeln. 1871 wanderte Bell in die USA aus und arbeitete zunächst als Taubstummenlehrer in Northampton, Massachusetts. 1872 gründete er selbst eine Gehörlosenschule und wurde 1873 in Boston zum Professor ernannt. In seiner neuen Umgebung hatte Bell die Zeit und die Ressourcen, um sich der Entwicklung des harmonischen Telegrafen zu widmen.

Es war ein Wettlauf mit der Zeit, den er, zumindest juristisch, für sich entschied. Die Zuteilung von Patent 174465 machte Bell zum offiziellen Erfinder des Telefons und zementierte seinen Aufstieg zu einem der erfolgreichsten Unternehmer des 19. Jahrhunderts. Im Jahr 1877 gründete Bell mit seinen Geschäftspartnern Sanders und Hubbard die *Bell Telephone Company*. 1879 ging daraus die *National Bell Telephone Company* und 1880, durch eine weitere Fusion, die *American Bell Telephone Company* hervor. 1885 entstand die *American Telephone and Telegraph Company*, kurz AT&T.

Im Jahr 1880 bekam Bell von der französischen Regierung für die Erfindung des Telefons den Volta-Preis in Höhe von 50.000 Francs zugesprochen. Mit diesem Geld gründete er das *Volta Laboratory*, das heute unter dem Namen *Bell Laboratories* firmiert. Die Institution entwickelte sich zu einer der erfolgreichsten Forschungsstätten des 20. Jahrhunderts, die zahlreiche namhafte Wissenschaftler für sich gewinnen konnte.

Alexander Graham Bell starb am 1. August 1922 in Baddeck, Neuschottland, im Alter von 75 Jahren.

Übertragung primitiver Töne ausgelegt, doch das, was auf der Empfängerseite zu hören war, hatte eine sehr viel komplexere Struktur. In Bell wuchs der Verdacht, dass seine prototypische Konstruktion zu mehr im Stande sein könnte, als primitive Töne zu übertragen.

In einem Brief an Gardiner Green Hubbard äußerte Bell:

*„If I can get a mechanism which will make a current of electricity vary in its intensity as the air varies in density when a sound is passing through it, I can telegraph any sound, even the sound of speech.“ [45]*

Alexander Graham Bell

Am 25. Februar 1875 reichte Bell seinen ersten Patentantrag ein, mit dem Titel „*Improvement in transmitters and receivers for electric telegraphs*“. Erteilt wurde das Patent am 6. April unter der Nummer 161739.

Ab jetzt rückte die Übertragung von Sprache immer stärker in den Mittelpunkt und im Jahr 1875 konnten Bell und Watson einen weiteren Er-

folg vermelden. Durch eine Modifikation der Sendeeinrichtung gelang es den beiden zum ersten Mal, Geräusche in wiedererkennbarer Form auf der Empfängerseite zu replizieren. Das Ergebnis war das berühmte *Gallows telephone*, eine galgenförmige Apparatur, die den Namen Telefon eigentlich gar nicht verdiente; von einer akzeptablen Sprachqualität war das Gerät noch weit entfernt.

Bell und Watson waren sich sicher, auf dem richtigen Weg zu sein, und arbeiteten mit Hochdruck an der Verbesserung ihrer Apparatur. Doch die Monate verstrichen, ohne dass der so sehnlich erhoffte Durchbruch gelang. Bells Geldgeber drangen auf Erfolge, und so wuchs der Druck von Tag zu Tag.

### Elisha Gray



Elisha Gray (1835 – 1901)

**Abb. 1.37:** Gerechtfertigter Verlierer oder betrogener Gewinner? Die von juristischer Finesse und strategischer Subjektivität geprägte Patentschlacht macht es schwierig, eines der spannendsten Kapitel der Telekommunikationsgeschichte wissenschaftlich aufzuarbeiten. Bis heute sind sich Historiker darüber uneinig, wem Ruhm und Ehre für die Erfindung des Telefons tatsächlich gebührt.

Fortschritte machte derweil ein anderer: Elisha Gray (Abbildung 1.37). Genau wie Bell suchte der in Barnesville, Ohio, geborene Gray nach einer Möglichkeit, um mehrere Morse-Datenströme simultan über eine einzige physikalische Leitung zu übertragen. Dabei entdeckte auch er, dass sich das Grundprinzip der harmonischen Telegrafie genauso gut für die elektrische Übertragung von Sprache einsetzen lässt.

Finanziert wurden Grays Forschungen durch Samuel S. White, einen betuchten Zahnnarzt aus Philadelphia. White sah in der Telefonie eine Spielerei und war ausschließlich an der Entwicklung des harmonischen Telegrafen interessiert. Gray ging seiner Entdeckung trotzdem weiter nach und dokumentierte die Ergebnisse in zwei Patentanträgen. In seinem Antrag „*Electric Telegraph for Transmitting Musical Tones*“ beschrieb er seine Version des harmonischen Telegrafen und bekam das Patent am 27. Juli 1875 zugesprochen. Am 14. Februar 1876 reichte er einen weiteren Antrag ein, diesmal mit dem Titel „*Transmitting Vocal Sounds Telegraphically*“. Hierin beschreibt Gray seine Erfindung mit den folgenden Worten:

„Be it known that I, Elisha Gray, of Chicago, in the County of Cook, and State of Illinois, have invented a new art of transmitting vocal sounds telegraphically, of which the following is a specification: It is the object of my invention to transmit the tones of the human voice through a telegraphic circuit, and reproduce them at the receiving end of the line, so that actual conversations can be carried on by persons at long distances apart.“

Elisha Gray, 1876



Elisha Gray wurde am 2. August 1835 in Barnesville, Ohio, als Sohn einer Quäkerfamilie geboren. Aufgewachsen ist er auf einer Farm in der Nähe seiner Geburtsstadt. Bereits in seiner Jugend zeigte Gray ein reges Interesse an technischen Apparaturen und konstruierte im Alter von 10 Jahren einen ersten voll funktionsfähigen Telegrafen [30].

Grays Ausbildung verlief anders als geplant. Durch den frühen Tod seines Vaters war er im Alter von 12 Jahren gezwungen, vorzeitig die Schule zu verlassen und eine Ausbildung als Grobschmied und Bootsbauer zu beginnen.

Seine Wissbegierde konnte das Handwerk auf Dauer nicht stillen, und so entschied er sich im Alter von 22 Jahren, an die Schulbank zurückzukehren. Nach einem fünfjährigen Studium am Oberlin College, in dem er sich insbesondere der Physik und der Elektrotechnik widmete, bekam Gray eine Anstellung als Lehrer, zunächst am Oberlin College und später am Ripon College in Wisconsin.

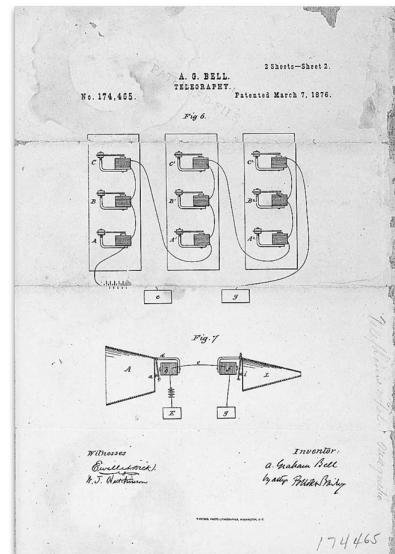
Im Jahr 1867 erhielt Gray sein erstes Patent für ein selbstjustierendes Telegrafenrelais zugesprochen. Dies war der Startschuss für eine einzigartige Erfinderkarriere, die am Ende über 70 Patentanmeldungen zählte. Mit der 1876 eingereichten Patentskizze über die Erfahrung des Telefons trat er in direkte Konkurrenz zu Graham Bell, der einen ähnlichen Antrag am selben Tag einreichte. Der Ausgang der Geschichte ist bekannt: Aus der über mehrere Jahre andauernden Patentschlacht ging Gray am Ende als Verlierer hervor; Graham Bell bekam die alleinigen Verwertungsrechte an der Erfahrung des Telefons zugesprochen.

Für Gray war die juristischen Niederlage ein schwerer Schlag, doch seinen Forschergeist konnte sie nicht brechen. Bis zu seinem Lebensende war er ein aktiver Erfinder und unermüdlich damit beschäftigt, die Kommunikationstechnik weiter zu verbessern. Im Jahr 1899 entwarf er eine Apparatur für die marine Unterwasserkommunikation, und im Dezember 1900 folgten die ersten praktischen Erprobungen. Beenden konnte Gray sein Projekt nicht mehr. Bereits wenige Tage später, am 21. Januar 1901, wurde er durch einen Herzfall abrupt aus dem Leben gerissen.

Elisha Gray hatte nichts weniger als das *Telefon* zum Patent angemeldet. Zugesprochen wurden die Verwertungsrechte aber nicht Gray, sondern seinem Widersacher, Graham Bell. Dieser hatte zwei Stunden zuvor einen Patentantrag mit ähnlichem Inhalt eingereicht. Bells Antrag mit dem Titel *Improvement in Telegraphy* wurde am 7. März 1876 stattgegeben und seine Erfahrung unter der Nummer 174465 patentiert (Abbildung 1.38). Tatsächlich hatte Bell zu dieser Zeit noch gar kein Gerät in Händen, mit dem sich Sprache nach dem beschriebenen Verfahren übertragen ließ. Dass er seinen Antrag trotzdem einreichen durfte, hatte er einer Änderung des amerikanischen Patentrechts zu verdanken. Nach diesem darf eine technische Apparatur auch dann zum Patent angemeldet werden, wenn sie lediglich als Idee existiert und ihre Funktionsweise noch nicht durch einen realen Prototyp nachgewiesen wurde.

Den technischen Durchbruch schafften Bell und Watson am 10. März 1876. Mit einem Mikrofon, das auf Grays Prinzip des *liquid transmitter* basierte, gelang es den beiden, Sprache in verständlicher Form zu übertragen. In seiner Biografie erinnert sich Watson so an diesen historischen Moment:

„It was during one of Bell's experiments on this kind of a telephone that the first sentence was transmitted and un-



**Abb. 1.38:** Eine Seite von Patent 174465, dem wahrscheinlich wertvollsten Dokument, das jemals von einer Patentbehörde ausgestellt wurde.



In Watsons Autobiografie ist ein wichtiges Detail enthalten, über das wir nicht vorschnell hinweglesen wollen. Aus seiner Schilderung der Ereignisse vom 10. März 1876 geht eindeutig hervor, dass der legendäre Satz „*Mr. Watson, come here; I want you!*“ durch einen Sender übertragen wurde, der den elektrischen Strom durch eine in ein Flüssigkeitsbad eingelassene Elektrode modulierte. In seinem Patentantrag hatte Bell aber einen Sender beschrieben, der elektromagnetisch arbeitet und damit nach einem ganz anderen Prinzip funktioniert. Die Möglichkeit, akustische Schallwellen so in elektrischen Strom umzusetzen, wie es am 10. März geschehen ist, hatte er in seinem amerikanischen Antrag nur vage umrissen, und diese Passage in der später angefertigten britischen Patentschrift sogar vollständig entfernt [19, 91].

Ein Blick in Grays Patentskizze offenbart Erstaunliches. Anders als Bell setzte Gray von Anfang an auf einen *liquid transmitter* und widmete sich in seiner Patentskizze ausführlich dessen Aufbau. Die meisten Historiker sehen es heute als gesichert an, dass Bell einen unerlaubten Einblick in den Patentantrag von Gray erhielt oder zumindest indirekt von dessen Inhalt erfuhr. Welche Auswirkungen dieser offensichtliche Verstoß gegen das Patentrecht schlussendlich hatte, ist in der Retrospektive schwer zu beurteilen. Doch eines scheint klar: Es war Grays *liquid transmitter*, der Bell am 10. März seinen historischen Durchbruch feiern ließ.

Es wäre voreilig, Elisha Gray aufgrund dieser Vorkommnisse zum eigentlichen Erfinder des Telefons zu erklären; hierfür sind die Fakten zu ungesichert und die historischen Entwicklungsstränge zu eng miteinander verflochten. Aber zumindest eine Schlussfolgerung scheint unzweifelhaft: Im Wettstreit um die Erfindung des Telefons wurde nicht immer mit fairen Mitteln gespielt.

*derstood. I had made for Bell a new transmitter in which a wire, attached to a diaphragm, touched acidulated water contained in a metal cup, both included in a circuit through the battery and the receiving telephone. The depth of the wire in the acid and consequently the resistance of the circuit was varied as the voice made the diaphragm vibrate, which made the galvanic current undulate in speech form. I carried the transmitter when finished to Exeter Place on the evening of March 10, 1876, intending to spend the night with Bell testing it. Neither of us had the least idea that we were about to try the best transmitter that had yet been devised. We filled the cup with diluted sulfuric acid, and connected it to the wire running between the two rooms. When all was ready I went into Bell's bedroom and stood by the bureau with my ear at the receiving telephone. Almost at once I was astonished to hear Bell's voice coming from it distinctly saying, 'Mr. Watson, come here; I want you!' He had no receiving telephone at his end of the wire so I couldn't answer him, but as the tone of his voice indicated he needed help, I rushed down the hall into his room and found he upset the acid of a battery over his clothes. He forgot the incident in his joy over the success of the new transmitter when I told him how plainly I had heard his words; and his joy was increased when he went to the other end of the wire and heard how distinctly my voice came through."*

Thomas A. Watson [99]

Im Mai 1876 waren Bell und Watson so weit, um ihr Ergebnis zu präsentieren. Ein funktionsfähiger Prototyp des legendären *Butterstamp telephone* wurde an der *American Academy of Arts and Sciences* des *Massachusetts Institute of Technology* (MIT) vorgestellt und anschließend auf der *Philadelphia Centennial Exhibition* ausgestellt [45]. Nachdem es Bell nach anfänglichen Mühen gelang, das Interesse der Jury- und Pressedelegation auf seine Apparatur zu lenken, wurde sein Telefon über Nacht bekannt. Die meisten Journalisten erkannten das große Potenzial seiner Erfindung, doch einige wenige hielten die Telefonie auch weiterhin für eine Spielerei. Beispielsweise betitelte die *Times of London* Bells Erfindung als „*latest american humbug*“ [45].

Am 9. Juli 1876 gründeten Bell, Sanders und Hubbard gemeinsam die *Bell Telephone Company of Massachusetts*, kurz BTC. Was die junge Firma damals am dringendsten benötigte, war Geld. Hubbard versuchte,

die Rechte für 100.000 Dollar an die Western Union Telegraph Company zu veräußern, doch diese lehnte die Offerte dankend ab. Ihr Präsident wird mit den folgenden Worten zitiert [55]:

„What use could this company make of an electrical toy?“

William Orton

Rückblickend war dies die größte Fehlentscheidung in der Geschichte der Western Union Telegraph Company.

Wenn überhaupt, so ist die ablehnende Haltung nur im historischen Kontext zu verstehen. Zu jener Zeit war der Morse-Telegraf ein etabliertes Medium und die Kommunikationsinfrastruktur großflächig ausgebaut. Alleine in den USA existierten im Jahr 1876 über 8000 Telegrafenstationen, die über ein gigantisches Netz von knapp 350 000 km Länge miteinander verbunden waren. Für viele Zeitgenossen war die Telegrafie zu einem so vertrauten Teil ihres Alltags geworden, dass der Gedanke, diese Technik könnte eines Tages überholt sein, außerhalb des Vorstellbaren lag.

Trotzdem wuchs in den Folgemonaten bei der Western Union Telegraph Company die Angst, denn immer deutlicher zeichnete sich ab, dass der Aufstieg der Telefonie nicht mehr aufzuhalten war. In der Führungsstufe des Monopolisten war man sich einig: Würde die Telegrafie ihre Stellung als vorherrschendes Kommunikationsmedium mittelfristig verlieren, so käme dies einer existenziellen Bedrohung gleich. Mit dem Rücken zur Wand, entschloss sich die Firma, das Bell'sche Patent vor Gericht zu Fall zu bringen. Ihre größte Waffe war Elisha Gray, an dessen Erfindung sie die Rechte besaß. Sollte es gelingen, Gray als Urheber des Telefons juristisch festschreiben zu lassen, so wäre Bells Patent nicht mehr das Papier wert, auf dem es gedruckt war.

Was nun begann, ging in die Geschichte als die größte Patentschlacht ein, die jemals um eine technische Erfindung ausgefochten wurde. Mehr als 600 Prozesse wurden geführt, über eine quälend lange Zeit von 11 Jahren. Am Ende entschieden die Gerichte für Bell und legten damit schützende Hände über sein wirtschaftliches und finanzielles Imperium. Für die Western Union Telegraph Company markierte das Urteil den Anfang vom Ende einer über viele Jahre hinweg erarbeiteten Monopolstellung.

In wissenschaftlicher Hinsicht machen es uns die Patentstreitigkeiten schwer, den wahren Erfinder des Telefons ausfindig zu machen. Viele Originalquellen sind durch die damals verfolgten ökonomischen und

*„Consiste in un diaframma vibrante e in un magnete elettrizzato da un filo a spirale che lo avvolge. Vibrando, il diaframma altera la corrente del magnete. Queste alterazioni di corrente, trasmesse all’altro capo del filo, imprimono analoghe vibrazioni al diaframma ricevente e riproducono la parola.“ [102]*

*„It consists of a vibrating diaphragm and an electrified magnet from a wire that wraps around it in a spiral. The vibrating diaphragm alters the current of the magnet.*

*These alterations of current are all transmitted to the other end of the wire, creating analogous vibrations to the receiving diaphragm and thus, reproduce the words.“ [102]*



Antonio Meucci  
(1808 – 1889)

**Abb. 1.39:** In den Jahren 1856 bis 1870 baute Antonio Meucci mehr als 30 Prototypen seines als *Teletrofono* bezeichneten Fernsprechers. Auch er ging aus der Patent-schlacht gegen Graham Bell als Verlierer hervor.

juristischen Interessen nicht neutral verfasst und lassen nur schwerlich erkennen, wem Ruhm und Ehre tatsächlich gebührt. Für einige Experten ist Bell, neben dem Nießnutzer, auch der geistige Vater des Telefons, andere sehen die Urheberschaft dagegen bei Gray [19, 45, 94].

Allzu oft lässt der verbitterte Wettkampf um das Telefonpatent die Tatsache in den Hintergrund treten, dass weder Graham Bell noch Elisha Gray der Erste war, der Töne über eine elektrische Leitung übertrug. Dass dies möglich ist, hatten zwei andere Männer schon lange vorher unter Beweis gestellt: Antonio Meucci und Philipp Reis.

### Antonio Meucci

Der Italiener Antonio Meucci (Abbildung 1.39) wurde am 13. April 1808 in San Frediano, in der Nähe von Florenz, geboren. Er erkannte bereits in den 50er-Jahren, dass sich die Phänomene des Elektromagnetismus für die Übertragung von Sprache nutzen lassen, und installierte in seinem Haus ein telefonähnliches Gerät. Meuccis Arbeit hatte ganz praktische Gründe: Er konstruierte das Haustelefon für die Kommunikation mit seiner Frau, die aufgrund einer frühen rheumatischen Erkrankung nicht mehr in der Lage war, ihr Bett zu verlassen.

Abbildung 1.39 zeigt einen Auszug aus seinen Aufzeichnungen. Was der Italiener dort beschreibt, ist das Grundprinzip, nach dem alle elektromagnetischen Telefone arbeiten.

Meucci gelang es, seine ursprüngliche Konstruktion über die Jahre hinweg zu verbessern, und zwischen 1856 und 1870 entstanden über 30 verschiedene Prototypen. 1871 meldete Meucci seine Erfindung, die er selbst als *Teletrofono* bezeichnete, zum Patent an. Leider ist seine Antragsschrift mit dem Titel *Sound Telegraph* nur vage formuliert. Im Gegensatz zu seinen eigenen Aufzeichnungen wird dort weder von Spulen oder Elektromagneten noch von Batterien oder einer anderen Stromquelle geredet. Auch die Beschreibung der Membran ist dort nicht zu finden. Für seine Erfindung wurde Meucci ein zeitlich begrenztes Patent erteilt, das alle drei Jahre gegen Gebühr erneuert werden musste. Da der materielle Erfolg des Teletrofones zunächst ausblieb, konnte der finanziell angeschlagene Meucci das Patent im Jahr 1874 nicht mehr verlängern.

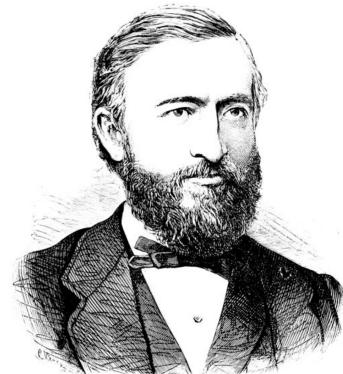
Die Rolle, die Meucci in der Entstehungsgeschichte des Telefons spielt, ist unter Experten umstritten. Während manche seine Leistung nur am Rande erwähnen, sehen andere in ihm den wahren Erfinder des Telefons. Bemerkenswert ist eine Resolution aus dem Jahr 2002, in der das

*U. S. House of Representatives* Meucci posthum für seine Arbeit würdig. Sie trägt die Nummer 296 und schließt mit den Worten:

*„Whereas if Meucci had been able to pay the \$10 fee to maintain the caveat after 1874, no patent could have been issued to Bell: Now, therefore, be it resolved, that it is the sense of the House of Representatives that the life and achievements of Antonio Meucci should be recognized, and his work in the invention of the telephone should be acknowledged.“*

U. S. House of Representatives, 2002

Zu Lebzeiten blieben dem Italiener Ruhm und Ehre verwehrt. Im Rahmen der legendären Patentschlacht versuchte er zwar, seine Ansprüche gegen Bell durchzusetzen, aber auch seine Klage wurde abgewiesen. Von Krankheiten gezeichnet, starb Antonio Meucci als verarmter Mann am 18. Oktober 1889, im Alter von 81 Jahren.



Philipp Reis  
(1834 – 1874)

**Abb. 1.40:** Der deutsche Lehrer Philipp Reis gehört zu den Ersten, denen die elektrische Übertragung von Tönen gelang.

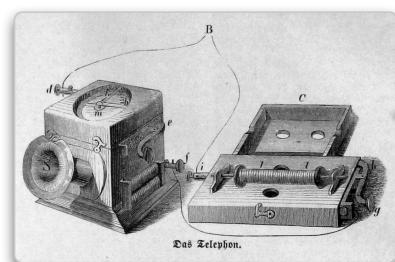
## Philip Reis

Es ist schwer, wenn nicht sogar unmöglich, in einem kurzen Abriss der Geschichte all jene zu würdigen, die zur Erfindung des Telefons beige tragen haben, doch ein Name darf auf keinen Fall unerwähnt bleiben: Philipp Reis (Abbildung 1.40). Seit seiner Jugend war der deutsche Lehrer von der Idee gefesselt, das gesprochene Wort auf elektrischem Wege zu übertragen, doch erst im Erwachsenenalter verfügte er über die Zeit und die Ressourcen, um an der Umsetzung seines Traums zu arbeiten. Mit dem Bau erster Prototypen begann er mit 24 Jahren. Drei Jahre später, am 26. Oktober 1861, präsentierte er seine Apparatur vor den Mitgliedern des Physikalischen Vereins in Frankfurt zum ersten Mal öffentlich.

Reis baute insgesamt 10 verschiedene Varianten seines Senders und 4 verschiedene Varianten seines Empfängers [45]. Abbildung 1.41 zeigt seinen letzten Entwurf, den er am 11. Mai 1862 den Mitgliedern des Freien Deutschen Hochstifts für Wissenschaften, Künste und allgemeine Bildung in Frankfurt vorstellte. Dort fiel auch der legendäre Satz

*„Das Pferd frisst keinen Gurkensalat!“*,

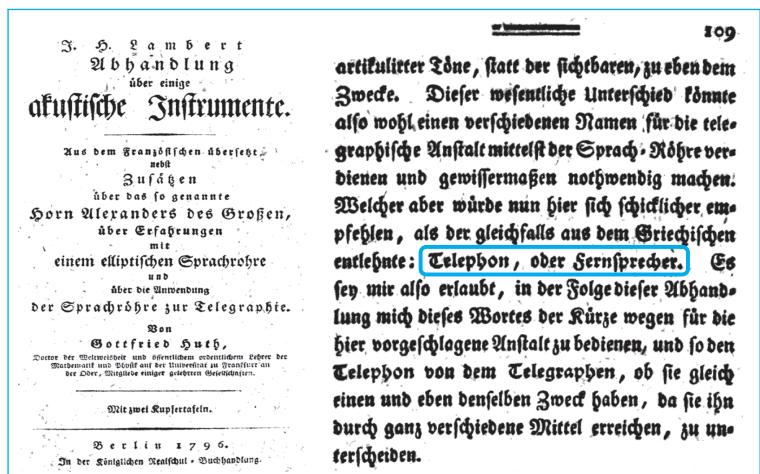
um die Übertragungsqualität des Telefons zu testen.



Das Telefon von Philipp Reis

**Abb. 1.41:** Reis leitete das Wort *Telephon* aus dem Wort *Telegraph* ab. Es ist eine Zusammensetzung der beiden altgriechischen Wörter τῆλε (*tēle*, dt. *fern*) und φωνή (*phōnē*, dt. *Stimme oder Sprache*) (vgl. Abbildung 1.42).

**Abb. 1.42:** Philipp Reis war einer der Ersten, die den Begriff *Telephon* benutzten, aber nicht, wie gelegentlich vermutet, der Schöpfer dieser Wortkombination. Bereits 1796 tauchte der Begriff *Telephon* in einer Abhandlung des deutschen Mathematik- und Physikprofessors Gottfried Huth auf. Dass Philipp Reis in vielen Quellen dennoch als Urheber dieses Worts genannt wird, hat einen einfachen Grund: Huths Arbeit war weitgehend unbekannt, und erst durch Reis wurde der Begriff einer breiteren Öffentlichkeit geläufig und dadurch Teil des Sprachgebrauchs.



In der Folgezeit wurden von der Reis'schen Apparatur mehrere Kopien erstellt und als wissenschaftliche Demonstrationsobjekte verkauft. Einer dieser Nachbauten ging im Jahr 1874 an die Smithsonian Institution in Washington, D.C., wo er fortan vielen Wissenschaftlern als inspirierende Quelle für eigene Ideen diente. Zu diesen Wissenschaftlern gehörte auch Graham Bell. Als er von Joseph Henry das Exponat gezeigt bekam, konnte er freilich noch nicht wissen, dass es zu einer der größten Bedrohungen seines 1876 erteilten Patents werden würde.

Die Gefahr kam in Person von Ashael K. Eaton daher. Der amerikanische Professor hatte selbst ein Telefon entwickelt und versuchte mit den Arbeiten von Reis zu beweisen, dass Bells Erfindung zur Zeit der Antragstellung keinesfalls neu war. Der *Eaton-Spencer case* begann am 22. Juni 1880, und in einer der sich anschließenden Gerichtsverhandlungen wurde auch ein Telefon der Reis'schen Bauart öffentlich demonstriert. Bell hatte Glück, denn der federführende Richter fällte ein Urteil über den Apparat, das kaum hätte vernichtender ausfallen können. Er wird mit den folgenden Worten zitiert [105]:

*„Bell discovered a new art – that of transmitting speech by electricity – and has a right to hold the broadest claim for it which can be permitted in any case; not to the abstract right of sending sounds by telegraph, without any regard to means, but to all means and processes which he has both invented and claimed. [...] An apparatus made by Reis, of Germany, in 1860, and described in several publications before 1876, is relied on to limit the scope of Bell's inventi-*



Philipp Reis erblickte am 7. Januar 1834 im hessischen Gelnhausen das Licht der Welt. Da seine leibliche Mutter bereits ein Jahr nach der Geburt verstarb, verbrachte der Junge einen großen Teil seiner Kindheit und Jugend bei der Großmutter. Seinen Vater verlor er im Alter von 10 Jahren.

Sein weiterer Werdegang war zunächst bodenständig. Mit 16 Jahren wurde Reis Lehrling bei einem Farbwarenhändler und nahm Unterricht in einer Handelsschule. Bereits als Kind war er von den Naturwissenschaften begeistert und beschäftigte sich in seiner Freizeit regelmäßig mit verschiedenen Themen aus der Physik und der Chemie. Die Telegrafie hatte für Reis eine ganz besondere Bedeutung. Von einem „sprechenden Draht“ träumte er bereits in seiner Jugend, doch erst später hatte er die Zeit und die finanziellen Mittel, um seinen Traum in die Tat umzusetzen.

Im Jahr 1851 trat Reis dem Physikalischen Verein in Frankfurt bei. Dort fand er das notwendige Umfeld vor, um seine lange gehegten Ideen zu konkretisieren. 1858 nahm er eine Lehrerstelle in Friedrichsdorf an. In dieser Zeit entstand das

berühmte Modell einer Ohrmuschel, das Reis als Demonstrationsobjekt für seine Schüler konzipierte. Dieses Modell lieferte ihm wichtige Erkenntnisse für sein späteres Projekt: den Bau einer Apparatur für die Übertragung von Tönen mithilfe des galvanischen Stroms. Erstmals öffentlich stellte Reis seine Apparatur, die er selbst als Telefon bezeichnete, im Jahr 1861 vor Mitgliedern des Physikalischen Vereins in Frankfurt vor. An einer wirtschaftlichen Verwertung war er nie ernsthaft interessiert, doch auch die wissenschaftliche Honorierung blieb ihm zu großen Teilen verwehrt. Reis fehlte als einfacher Lehrer die notwendige Reputation, um seine Erfindung die gebührende Anerkennung zu verleihen. Gleich mehrere Fachzeitschriften lehnten es ab, einen Artikel über seine Arbeit zu veröffentlichen.

Trotz der ihm entgegengebrachten Ignoranz war er davon überzeugt, dass aus seiner Erfindung etwas Großes entstehen würde. Aktiv mitwirken konnte er daran nicht mehr. Reis erkrankte früh an Tuberkulose und konnte in den letzten Jahren seines Lebens das Bett nicht mehr verlassen. Am 14. Januar 1874 erlag er seiner schweren Krankheit, eine Woche nach seinem vierzigsten Geburtstag.

*on. Reis appears to have been a man of learning and ingenuity. He used a membrane and electrodes for transmitting sounds, and his apparatus was well known to curious inquirers. The regret of all its admirers was, that articulate speech could not be sent and received by it.“*

Judge Lowell, 27. Juni 1881

Wenig später wird Lowell noch deutlicher:

*„A century of Reis would never have produced a speaking telephone by mere improvement in construction.“*

Tatsächlich gab es einen gravierenden technischen Unterschied zwischen den Apparaturen von Reis und Bell. Reis entdeckte die Telefonie nicht wie Bell und Gray auf der Suche nach dem harmonischen Telegrafen; vielmehr war seine Apparatur eine direkte Weiterentwicklung des Morse-Telegrafen und basierte auf den gleichen elementaren Grundprinzipien. Im Reis'schen Telefon versetzten die akustischen Schallwellen eine Membran in Schwingungen, die mit einem Stromkreis verbunden war. Der Stromkreis wurde durch die Schwingungen in kurzen Abständen geöffnet und geschlossen, sodass die Schallwellen in ein

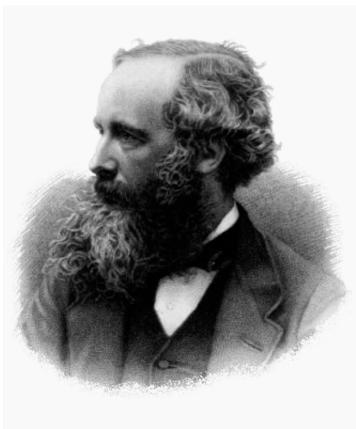
digitales Signal übersetzt wurden, und nicht, wie bei Bell oder Gray, in ein analoges. Der daraus resultierende Qualitätsverlust war so hoch, dass zwar Töne und Gesang wiedererkennbar übertragen wurden, nicht aber das gesprochene Wort. Dennoch haben später durchgeführte Versuche gezeigt, dass durch eine geringfügige Änderung der Reis'schen Apparatur eine analoge Modulation, und damit eine deutlich verbesserte Sprachqualität möglich gewesen wäre.

Alles in allem bewerten Historiker die Leistungen von Reis unterschiedlich. Zu denen, die sich intensiv mit der Geschichte der Telefonie beschäftigt haben, gehört der renommierte britische Physiker Silvanus Thompson. In seinem Werk *Philip Reis: Inventor of the telephone* trug er zahlreiche Originaldokumente zusammen und kommt zu dem Schluss:

„*The honor, to have transmitted the human voice by means of electricity first, owes to rice.*“

Silvanus Thompson [94]

### 1.5.2 Drahtlos durch den Äther



James Clerk Maxwell (1831 – 1879)

**Abb. 1.43:** Im Jahr 1865 postulierte der schottische Physiker James Clerk Maxwell die Existenz elektromagnetischer Wellen. Bis seine theoretischen Überlegungen experimentell verifiziert werden konnten, sollten aber noch Jahre vergehen.

Ende des 19. Jahrhunderts preschte die Grundlagenforschung in den Naturwissenschaften in atemberaubender Geschwindigkeit voran. Der medizinische Fortschritt nahm ehemals todbringenden Krankheiten den Schrecken, und die Erkenntnisse in der Chemie und Biologie ließen tiefer in das Wesen der Natur blicken als jemals zuvor. Auch auf dem Gebiet der Elektrizitätslehre lieferte die Forschung neuartige Erkenntnisse, die unser physikalisches Weltbild bis heute prägen. Zu den Sternstunden des 19. Jahrhunderts gehören zweifelsfrei die Entdeckungen des schottischen Physikers und Mathematikers James Clerk Maxwell (Abbildung 1.43). 1865 offenbarte er in seiner Arbeit mit dem Titel *A Dynamical Theory of the Electromagnetic Field* Erstaunliches. Durch eine Reihe von theoretischen Überlegungen war es ihm gelungen, jene partiellen Differentialgleichungen zu formulieren, die wir heute als die *Maxwell'schen Gleichungen* bezeichnen. Mit seinen unscheinbar anmutenden Formeln sagte er ein bis dato unbekanntes Phänomen voraus, das die Entwicklung der Nachrichtentechnik in eine ganz neue Richtung lenken sollte: die Existenz *elektromagnetischer Wellen*.

Für die moderne Physik war die formulierte Theorie von hoher Relevanz, und gleichsam stellte sie die Wissenschaft vor ein Dilemma:

Maxwell hatte die Existenz elektromagnetischer Wellen allein aus theoretischen Überlegungen heraus vorhergesagt; einen wirklichen Nachweis hatte er für seine Behauptungen aber nicht in Händen. Um eine schnelle Klärung herbeizuführen, startete die Berliner Akademie der Wissenschaften im Jahr 1879 einen Wettbewerb, mit dem Ziel, die Maxwell'sche Theorie experimentell zu stützen. Doch die Jahre verstrichen, ohne dass sich der herbeigesehnte Erfolg einstellte. War das Phänomen der elektromagnetischen Welle doch nur eine Erfindung des Geistes, basierend auf einer unvollständigen oder gar falschen Theorie?

Eine endgültige Klärung erfuhr die Grundlagenfrage im Jahr 1886. Am 11. November führte der deutsche Physiker Heinrich Hertz einen Versuchsaufbau vor, mit dem sich elektromagnetische Wellen zwischen einer Sende- und einer Empfangseinrichtung messbar übertragen ließen (Abbildungen 1.44 und 1.45). Seit diesem Tag waren die von Maxwell postulierten Wellen kein bloßes Gedankengebilde mehr; sie waren zur physikalischen Realität geworden.

Mit seinem legendären Versuchsaufbau aus dem Jahr 1886 hatte Hertz nicht nur eine der wichtigsten Grundlagenfragen der Physik beantwortet, sondern gleichermaßen die prinzipielle Machbarkeit der drahtlosen Nachrichtenübertragung unter Beweis gestellt. Wir wissen nicht, ob es die grenzenlose Freude über die Lösung eines der großen physikalischen Rätsel war oder schlicht die Eigentümlichkeit eines theoretisch interessierten Geistes: Fest steht, dass Hertz diesen zweiten, praktischen Aspekt seiner Entdeckung schlicht nicht erkannte. Doch Hertz war nicht alleine: Auch seine zahlreichen Wissenschaftskollegen waren sich der Relevanz für die Nachrichtentechnik ganz offensichtlich nicht bewusst. Tatsächlich sollten noch Jahre vergehen, bis die Wissenschaft den praktischen Nutzen der elektromagnetischen Wellen in ihrem vollen Umfang verstand.

Hertz sollte all dies nicht mehr erleben. Im Jahr 1894 wurde er durch eine schwere Infektion unvermittelt aus dem Leben gerissen, im Alter von nur 36 Jahren.

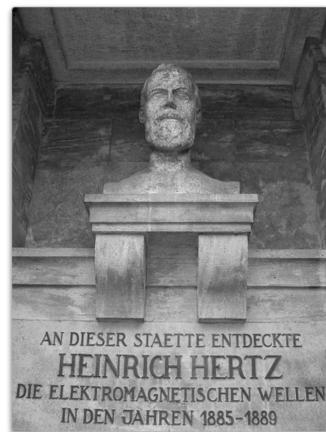
### Guglielmo Marconi

Die Presse reagierte mit einer Reihe von Nachrufen auf den Tod von Hertz und machte seine Arbeit hierdurch einer breiten Öffentlichkeit bekannt. Unter anderem erregte sie die Aufmerksamkeit eines jungen Mannes, der auf einer Urlaubsreise in den Alpen vom Tode des berühmten Physikers las: Guglielmo Marconi. Der zwanzigjährige Italiener war



Heinrich Rudolf Hertz (1857 – 1894)

**Abb. 1.44:** 1886 bestätigte Heinrich Hertz die Maxwell'sche Theorie. In diesem Jahr gelang ihm der experimentelle Nachweis elektromagnetischer Wellen.



**Abb. 1.45:** Eine Büste von Heinrich Hertz erinnert im Eingangsbereich zum Hertz-Hörsaal an das bedeutende Experiment, das am 11. November 1886 an dieser Stätte durchgeführt wurde. Der Hertz-Hörsaal befindet sich auf dem Südcampus des Karlsruher Instituts für Technologie, in unmittelbare Nähe des Haupteingangs.



Guglielmo Marconi war der zweite Sohn des Italieners Giuseppe Marconi und der Irländerin Annie Jameson. Aufgewachsen ist Marconi im italienischen Bologna, wo er sich nach seiner Schulausbildung auch an der dort ansässigen Universität bewarb. Aufgrund seiner schlechten Leistungen wurde ihm die Einschreibung verwehrt, gleichsam fand er in Professor Augusto Righi einen wohlgesonnenen Mentor. Righi erlaubte Marconi den Besuch seiner Vorlesungen und ermöglichte ihm den Zugang zur Universitätsbibliothek.

Marconis intensive Beschäftigung mit der Telegrafie und den Phänomenen des Elektromagnetismus trugen alsbald Früchte. Im Jahr 1895 gelang es ihm, elektrische Signale von seinem Elternhaus drahtlos zu einem rund 2 km entfernten Empfänger zu übertragen. Sein Apparat war eine technische Revolution, und dennoch scheiterte er mit seinem Versuch, seine Heimatregierung als Geldgeber für weitere Forschungsvorhaben zu gewinnen.

Marconis Zukunft wurde in England geschrieben. Mit der finanziellen Unterstützung von Sir William Preece erlangte er 1896 ein Patent auf seine Erfindung und gründete ein Jahr später die *Wireless Telegraph & Signal Company*. Von da an ging es steil bergauf. Marconi war Vorreiter auf einer Wel-

le des technischen Fortschritts: 1899 gelang ihm der drahtlose Austausch von Nachrichten über den Ärmelkanal, und 1901 überbrückte er der Nordatlantik. Dabei schaffte er es stets, seine technologischen Fortschritte inbare Münze umzusetzen. Aus der Wireless Telegraph & Signal Company entstand im Laufe der Zeit ein verästeltes Firmenimperium, das ihn als einen der erfolgreichsten Unternehmer in die Geschichte der Telekommunikation eingehen ließ.

Doch nicht nur in ökonomischer, sondern auch in wissenschaftlicher und politischer Hinsicht war Marconi ein erfolgreicher Mann. Für seine Leistungen erhielt er im Jahr 1909 zusammen mit dem deutscher Wissenschaftler Karl Ferdinand Braun den Physik-Nobelpreis.

Durch sein Lebenswerk besaß Marconi einen beachtlichen Einfluss auf die Wissenschaft und Politik seines Landes. Im Jahr 1914 wurde er zum Senator nominiert und nahm 1919 an der Friedenskonferenz von Versailles teil. 1928 wurde er vom nationalen Forschungsrat, 1930 von der königlichen Akademie und 1933 vom italienischen Enzyklopädischen Institut zum Vorsitzenden gewählt.

1935 folgte er einem Ruf an die Universität Rom, wo ihm jedoch nur noch wenig Zeit gegeben war. Zwei Jahre nach seiner Berufung starb Guglielmo Marconi, am 20. Juli 1937, an den Folgen eines Herzinfarkts.



Guglielmo Marconi  
(1874 – 1937)

**Abb. 1.46:** Mit seinen Arbeiten auf dem Gebiet der Funktechnik hat Guglielmo Marconi der drahtlosen Kommunikation zum Durchbruch verholfen.

sich der großen Bedeutung der Hertz'schen Apparatur für die drahtlose Nachrichtenübertragung sofort bewusst, und er beschloss, nach seiner Rückkehr umgehend mit dem Bau einer Übertragungseinrichtung zu beginnen. Dass sich eine funktionsfähige Apparatur konstruieren ließe, stand für Marconi außer Frage. Sein größtes Problem war ein ganz anderes: Für ihn war der Bau eines drahtlosen Telegrafens eine so naheliegende Antwort auf das Hertz'sche Experiment, dass er sich nicht recht vorstellen konnte, der Erste mit dieser Idee zu sein. In [18] wird Marconi mit den folgenden Worten zitiert:

*„My chief trouble was that the idea was so elementary, so simple in logic, that it seemed difficult for me to believe that no one else had thought of putting it into practice. Surely, I argued, there must be much more mature scientists than myself who had followed the same line of thought and arrived at an almost similar conclusion.“*

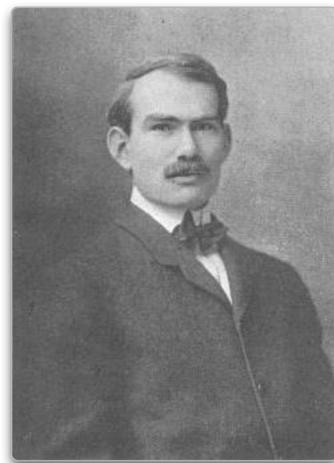
Guglielmo Marconi

Doch Marconi war der Erste. Ein erster Prototyp war schnell gebaut, und im Jahr 1896 war seine Konstruktion weit genug gediehen, um sie in einem Patentantrag offiziell zu dokumentieren. Einen bedingungslosen Fürsprecher fand Marconi in Sir William Preece, Chefingenieur des *British Post Office*. Preece war vom kommerziellen Erfolg der Apparaturen überzeugt und kümmerte sich um die finanzielle Unterstützung. 1897 gründete Marconi die *Wireless Telegraph & Signal Company*, wo er mit Hochdruck an der Verbesserung seiner drahtlosen Kommunikationstechnik arbeitete. Im Jahr 1901 wagte er das Unfassbare: die drahtlose Übertragung einer Nachricht über den atlantischen Ozean. In der Öffentlichkeit wurde der Plan des Italieners skeptisch beurteilt. Wie schon im Fall der Verlegung des ersten Transatlantikkabels wurde sein Vorhaben als Sinnbild einer grenzenlosen Überheblichkeit verspottet. Die Betreiber der profitablen transatlantischen Telegrafenlinien beobachteten Marconis Pläne dagegen sehr genau. Würde die drahtlose Übertragung über eine so große Distanz hinweg gelingen, so stünde ihr Geschäftsmodell mit einem Schlag auf wackligen Füßen.

Marconis erstes Experiment war nur zum Teil erfolgreich. Die mehrfache Übertragung des morse-codierten Buchstabens 'S' hatte zwar prinzipiell funktioniert, allerdings waren viele Beobachter aufgrund der einfachen Nachricht und der wiederkehrend auftretenden Empfangsverluste vom Erfolg des Experiments nicht überzeugt. Einige behaupteten, die empfangenen Signale wären nichts weiter als das atmosphärische Rauschen, andere fühlten sich gänzlich getäuscht und waren der Meinung, es hätte überhaupt keine Übertragung über den Atlantik stattgefunden.

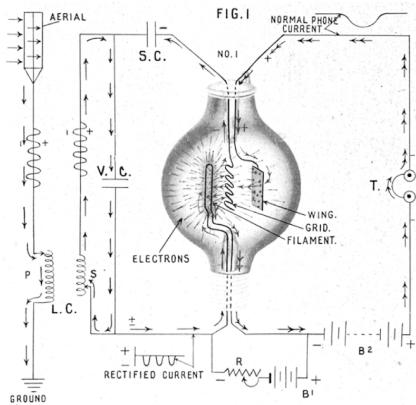
Ein Jahr später stach Marconi mit der S. S. Philadelphia in See und wiederholte seine Experimente. Er testete seine Apparatur in unterschiedlichen Entfernungen mit verschiedenen Sendefrequenzen und dokumentierte akribisch den Einfluss von Tageszeit und Wetter auf die Übertragungsqualität. Die sorgfältig durchgeführten Versuche überzeugten am Ende auch Marconis Kritiker. Ab nun bestand kein Zweifel mehr daran, dass die drahtlose Kommunikation auch über weite Strecken hinweg funktioniert. Der Siegeszug der drahtlosen Übertragungstechnik war nun nicht mehr aufzuhalten.

Insbesondere auf dem Gebiet der marinen Kommunikation führte die neue Technik rasch zu einer Revolution. Auf beiden Seiten des Atlantiks wurden hohe Antennen errichtet und alle großen Schiffe mit Sendeeinrichtungen versehen. Fortan war es möglich, Schiffe auf hoher See mit Nachrichten zu versorgen und im Gegenzug Rückmeldungen zu erhalten oder auf Notrufe zu reagieren.

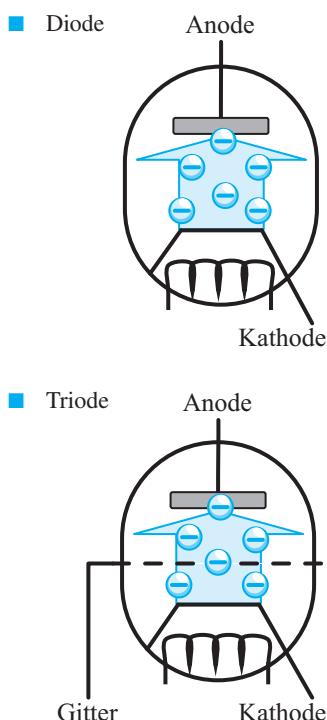


Lee De Forest  
(1873 – 1961)

**Abb. 1.47:** Viele Patente auf dem Gebiet der Kommunikationstechnik sind mit dem Namen Lee De Forest verbunden. Mit dem Audion erfand der Amerikaner den ersten praxistauglichen elektrischen Verstärker, einen der wichtigsten Grundbausteine der analogen Übertragungs- und Schaltungstechnik.



**Abb. 1.48:** Das Audion von Lee de Forest war der Vorläufer der Triodenröhre [24].



**Abb. 1.49:** Die Diodenröhre und die Triodenröhre im Vergleich

Auch auf dem Festland erfreute sich die drahtlose Kommunikation wachsender Beliebtheit. Mit modernen Mikrowellensendern war es auf einmal möglich, mehrere Morse-Datenströme zu bündeln und gemeinsam über große Distanzen zu übertragen – viel kostengünstiger und zuverlässiger als jemals zuvor. Es war die Zeit, in der die wartungsintensiven Überlandleitungen, einst Symbole des schier grenzenlosen Fortschritts, sukzessiv von der Bildfläche verschwanden. Die Erfindungen des 20. Jahrhunderts machten die Kommunikation unsichtbar.

Die drahtlose Kommunikation hatte aber nicht nur die Telegrafie revolutioniert; sie war gleichermaßen die Voraussetzung für die Verbreitung eines Massenmediums, dessen Existenz wir heute als selbstverständlich erachten: das Radio. Die erste Radionachricht wurde im Jahr 1906 durch den kanadischen Rundfunkpionier Reginald Aubrey Fessenden gesendet und vor einer Gruppe ausgewählter Wissenschaftler mediawirksam inszeniert.

### Lee de Forest

Im gleichen Jahr erfuhr die Radiotechnik durch eine Erfindung von Lee de Forest einen bedeutenden Entwicklungsschub (Abbildung 1.47). Mit dem *Audion* schuf der US-amerikanische Erfinder den Vorläufer der *Triodenröhre* (Abbildung 1.48).

Um die grundlegende Funktionsweise der Triode zu verstehen, werfen wir zunächst einen Blick auf die *Diode*, deren schematischer Aufbau in Abbildung 1.49 (oben) skizziert ist. Sie besteht aus einer Vakuumröhre, in die zwei Elektroden eingelassen sind. Die obere Elektrode heißt *Anode*, die untere *Kathode*. Wird die Anode mit dem Pluspol einer Spannungsquelle und die Kathode mit dem Minuspol verbunden, so drängen Elektronen in den geheizten Kathodendraht und werden von der Anode angezogen. Übersteigt die angelegte Spannung einen bestimmten Schwellenwert, so können sich die Elektronen lösen und zur Anode überspringen. So entsteht ein kontinuierlicher Stromfluss von der Kathode zur Anode.

Die Triode unterscheidet sich von der Diode durch eine zusätzliche Elektrode, die mit einem Metallgitter verbunden ist (Abbildung 1.49 unten). Wird an das Gitter eine Spannung angelegt, so wird der Stromfluss von der Kathode zur Anode angeregt oder gedämpft. Von entscheidender Bedeutung ist die Tatsache, dass eine kleine Änderung der Gitterspannung eine große Veränderung des Kathoden-Anoden-Stroms bewirkt, und genau diese Eigenschaft besaß auch das Audion. Lee de

Forest war es gelungen, einen wichtigen Grundbaustein der modernen Analogtechnik zu bauen, der bis dato noch nicht in dieser Form zur Verfügung stand: den *elektronischen Verstärker*.

Trotz dieser Erfolge glaubten im Jahr 1906 nur wenige daran, dass sich das Radio zu einem Massenmedium entwickeln würde. Die Vorstellung, kostengünstige Empfänger zu bauen, die sich für den Gebrauch im privaten Haushalt eignen würden, lag noch immer in weiter Ferne.

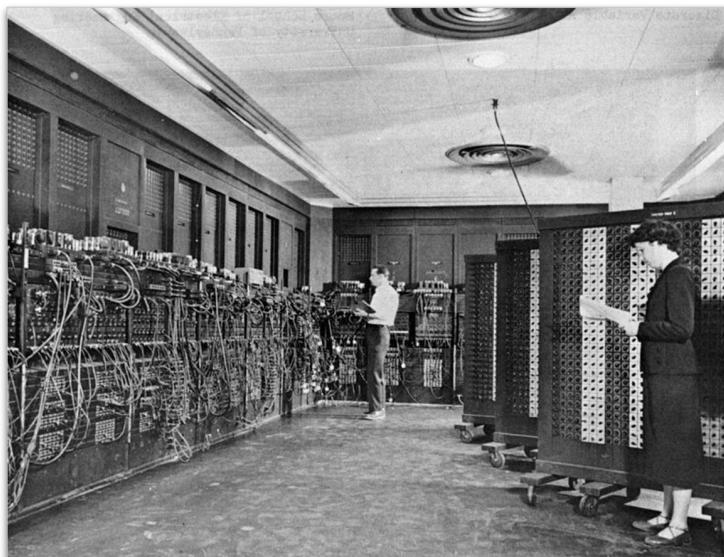
In Deutschland begann der Siegeszug der Radiotechnik in den düsteren Tagen des Dritten Reichs. Die Nationalsozialisten erkannten das propagandistische Potenzial des neuen Mediums und sorgten mit der industriellen Massenherstellung des *Volksempfängers* für eine großflächige Verbreitung (Abbildung 1.50). Wenige Jahre später hielt mit dem *Deutschen Einheits-Fernseh-Empfänger E1* auch der Fernseher Einzug in die heimische Stube. Das neue Medium konnte den Hörfunk in seiner Bedeutung bald überholen und zählt heute, neben dem Internet, zu den wichtigsten Massenmedien unserer Zeit.



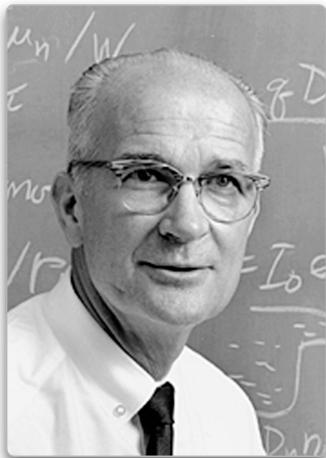
**Abb. 1.50:** Von den Nationalsozialisten wurde das Radio als Propagandainstrument missbraucht. Abgebildet ist ein Volksempfänger aus dem Jahr 1933 [23].

## 1.6 Von der Röhre zum Supercomputer

Die Erfindung der Triodenröhre hatte nicht nur die Entwicklung der Radio- und Fernsehtechnik revolutioniert, sondern auch in einem ganz



**Abb. 1.51:** Der ENIAC (Electronic Numerical Integrator And Computer) war die erste voll funktionsfähige Rechenmaschine, die nahezu allen Definitionen des modernen Computerbegriffs standhält und daher von vielen Experten als der erste wirkliche Computer der Welt angesehen wird. Er setzte sich aus 30 Hardware-einheiten zusammen, die U-förmig angeordnet waren und ein Gesamtgewicht von rund 30 Tonnen ergaben. Der parallele Betrieb von rund 18.000 verbauten Vaku umröhren verursachte eine Leistungsauf nahme von sagenhaften 174.000 Watt. Die Hitzeentwicklung sorgte für eine enorme Beanspruchung des Materials und ließ das Auffinden und Austauschen defekter Röhren alsbald zu einer Routinetätigkeit der ENIAC-Ingenieure werden.



William Shockley  
(1910 – 1989)



John Bardeen  
(1908 – 1991)



Walter Houser Brattain  
(1902 – 1987)

Abb. 1.52: William Shockley, John Bardeen und Walter Brattain

anderen Bereich einen Umbruch herbeigeführt: der Computertechnik. Der Grund dafür ist einfach. Wird die Gitterelektrode so geschaltet, dass der Strom entweder ungehindert oder gar nicht fließen kann, so entsteht ein *digitales Schaltelement*, ohne das es die meisten technischen Entwicklungen des 20. Jahrhunderts niemals gegeben hätte.



Abb. 1.53: Nachbau des 1947 von William Shockley, John Bardeen und Walter Brattain konstruierten Transistors

Ein historischer Moment in der Geschichte der Computertechnik war die Fertigstellung des *Electronic Numerical Integrator And Computer* (ENIAC) im Jahr 1946 (Abbildung 1.51). Der Rechnerkoloss war in vielerlei Hinsicht revolutionär. Mit der Fähigkeit, Verzweigungen und Schleifen auszuführen, besaß der ENIAC die Berechnungsstärke einer universellen Turing-Maschine und wird aus diesem Grund von vielen Experten als der erste wirkliche Computer der Welt angesehen.

Ferner markierte sein Bau einen der drei großen Technologiesprünge in der Computertechnik. Anstelle von elektrischen Relais verarbeiteten die Ingenieure erstmals Triodenröhren in großer Zahl und konnten die Schaltgeschwindigkeit hierdurch um den Faktor 1000 steigern. Damit war es möglich, den ENIAC mit einer Taktfrequenz von 100 Kilohertz zu betreiben, einer für damalige Verhältnisse schwindelerregenden Geschwindigkeit.

Der zweite Technologiesprung gelang durch die Erfindung des *Transistors*. Dessen Funktionsprinzip ließ sich Julius Edgar Lilienfeld schon 1928 patentieren, bis zur Konstruktion eines brauchbaren Prototyps sollten aber noch mehrere Jahre vergehen. Der erste technisch verwertbare Transistor wurde 1947 unter der Leitung von William Shockley, John Bardeen und Walter Brattain an den Bell Laboratories in New York gebaut (Abbildung 1.52 und 1.53). Ein halbes Jahr später war die Entwicklung so weit fortgeschritten, dass der Durchbruch in einer Presseerklärung vom 1. Juli 1948 öffentlich verkündet wurde:

*„An amazingly simple device, capable of performing efficiently nearly all the functions of an ordinary vacuum tube, was demonstrated for the first time yesterday at Bell Telephone Laboratories where it was invented. Known as the Transistor, the device works on an entirely new physical principle discovered by the Laboratories in the course of fundamental research into the electrical properties of solids. Although the device is still in the laboratory stage, Bell scientists and engineers expect it may have far-reaching significance in electronics and electrical communication.“*

Bell Laboratories, 1. Juli 1948 [9]

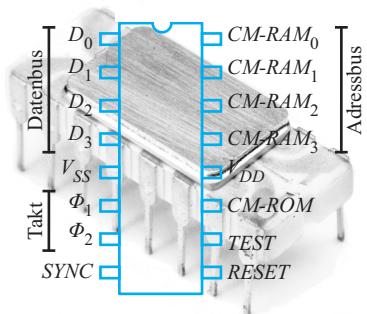
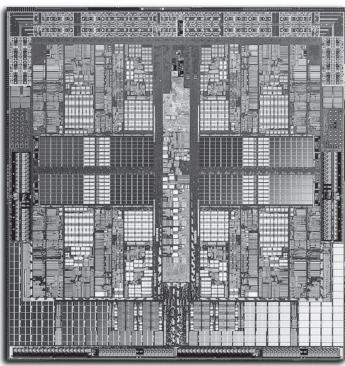


Abb. 1.54: Der 4004-Mikroprozessor

Zunächst waren Transistoren nur als diskrete Bauteile verfügbar und standen in direkter Konkurrenz zur Triodenröhre. In den Folgejahren gelang es den Ingenieuren, Transistoren immer weiter zu miniaturisieren, und im Jahr 1958 fand Jack Kilby einen Weg, mehrere Schaltelemente gemeinsam auf einem kleinen Stück Silizium zu produzieren. Mit dem *integrierten Schaltkreis* (*integrated circuit*, kurz IC) war es binnen weniger Jahre gelungen, den dritten großen Technologiesprung in der Geschichte der Computertechnik einzuleiten. Kilbys Entdeckung markiert die Geburtsstunde der *Hochintegration*, ohne die Computer niemals die Leistungsfähigkeit erreicht hätten, die wir heute als selbstverständlich erachten.

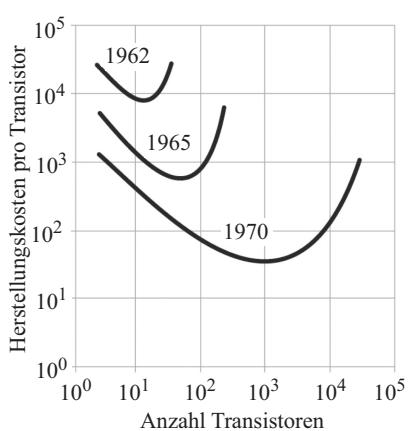
Ein Jahr später schuf der Halbleiterhersteller Fairchild Semiconductor mit der *Planartechnik* die Grundlage der Massenfertigung. Zu den damaligen Fairchild-Mitarbeitern gehörten Robert Noyce, Gordon Moore und Andrew Grove, die einen entscheidenden Einfluss auf die zukünftige Entwicklung der Computertechnik haben sollten. 1968 gründeten sie die *Integrated Electronics Corporation*, kurz Intel. Ursprünglich hatten die Gründerväter das Ziel, aus der jungen Firma einen erfolgreichen Speicherhersteller zu formen, doch schon bald wurde die Zukunft von



**Abb. 1.55:** Integrierter Schaltkreis mit mehreren Millionen Transistoren

Intel in eine andere Richtung gelenkt. Im Jahr 1971 gelang es dem Ingenieur Federico Faggin, alle Komponenten eines Prozessors auf einem einzigen Stück Silizium zu integrieren, und kurze Zeit später brachte die Firma mit dem Intel 4004 den ersten in Serie gefertigten Mikroprozessor auf den Markt (Abbildung 1.54).

Der 4004-Prozessor wurde in einer Strukturbreite von 10 µm gefertigt und mit einer Taktfrequenz von 740 kHz betrieben. Seine Rechenleistung entsprach in etwa der einer ENIAC und zeigte auf eindrucksvolle Weise, wie schnell sich die Computertechnik entwickelt hatte. In weniger als drei Jahrzehnten war den Ingenieuren das Kunststück gelungen, mit einem fingernagelgroßen Schaltkreis die gleiche Rechenleistung zu erzielen, für die einst ein raumfüllender Rechnerkoloss vonnöten war. Aus heutiger Sicht wirkt der 4004 dennoch wie ein Relikt aus längst vergangenen Tagen. Intern arbeitete er mit einer Bitbreite von lediglich 4 Bit und verfügte neben 16 Datenregistern über magere 4 Stapelregister. Mit modernen Prozessoren hat die Architektur des 4004 nicht mehr viel gemein. Dort rechnen mehrere parallel betriebene Prozessorkerne im Gigahertzbereich, die jeder für sich die Leistung früherer Großrechner um ein Vielfaches übertreffen. Auch die Anzahl der Transistoren spricht Bände. Setzte sich der 4004-Prozessor aus gerade einmal 2250 Transistoren zusammen, so liegt die Anzahl bei modernen Prozessoren weit jenseits der Milliarden-Grenze (Abbildung 1.55).



**Abb. 1.56:** Im Jahre 1965 prognostizierte Gordon Moore die exponentielle Zunahme der Chip-Komplexität. In leicht abgewandelter Form ist das Moore'sche Gesetz bis heute gültig.

*„The complexity for minimum component costs has increased at a rate of roughly a factor of two per year [...] Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years. That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000. I*

*believe that such a large circuit can be built on a single wafer.“*

Gordon Moore, 1965 [63]

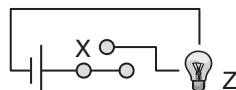
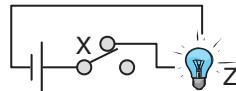
Jeder weiß, dass Moores Gesetz eines Tages seine Gültigkeit verlieren wird, schließlich steigt die Exponentielle Kurve so rasant an, dass die Menge der auf einem Chip integrierten Transistoren schon bald die gesuchte Anzahl an Elementarteilchen unseres Universums übersteigen würde. Wann der Tag kommen wird, an dem das Gesetz seine Gültigkeit verliert, weiß heute niemand. Mehrfach wurde in der Vergangenheit eine Sättigung der realen Wachstumskurve vorausgesagt, aber jedes Mal haben neue Erkenntnisse auf dem Gebiet der Halbleitertechnik dazu geführt, dass wir der von Moore prognostizierten Wachstumskurve folgen konnten.

## 1.7 Informations- und Codierungstheorie

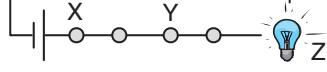
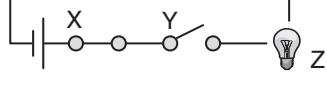
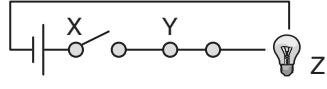
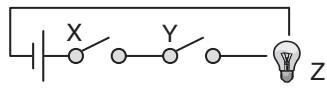
Die 30er- und 40er-Jahre des 20. Jahrhunderts waren von zwei Entwicklungen geprägt, die zunächst unabhängig voneinander verliefen.

- In den Forschungsabteilungen der großen Telefondienstleister arbeiteten zahllose Ingenieure mit Hochdruck daran, die analoge Übertragungstechnik zu perfektionieren. Als eine der größten Herausforderungen entpuppte sich wieder einmal das Problem der langen Leitung. Wird z. B. ein Telefongespräch über eine große Distanz analog übertragen, so nimmt die Stärke des elektrischen Signals mit der Zeit so stark ab, dass es in regelmäßigen Abständen verstärkt werden muss. Viele Ingenieure suchten die Lösung in der Konstruktion optimierter Signalverstärker, stießen dabei aber immer wieder auf dasselbe fundamentale Problem: Jede Verstärkung des Nutzsignals führt zwangsläufig zu einer Erhöhung des Rauschanteils und damit zu einer kontinuierlich sinkenden Signalqualität. Mit dem Einsatz verbesserter Bauelemente war es zwar möglich, die beobachteten Symptome zu lindern, das Grundproblem blieb aber ungelöst.
- Durch die Nutzung der Triodenröhre als Schaltelement hatte eine technologische Entwicklung an Fahrt aufgenommen, an deren Ende ein revolutionäres Funktionsprinzip stand: die digitale Informationsverarbeitung. An die Stelle von kontinuierlichen Spannungspegeln treten hier die zwei binären Zustände 0 und 1, die durch die Anwendung elementarer Schaltoperationen logisch miteinander

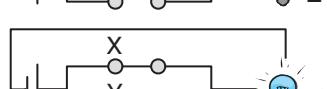
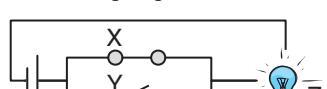
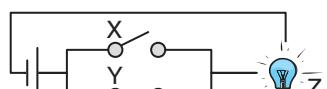
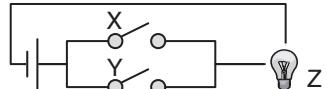
■ Negation



■ Konjunktion (UND-Verknüpfung)



■ Disjunktion (ODER-Verknüpfung)



**Abb. 1.57:** Alle digitalen Schaltungen lassen sich auf drei Grundverknüpfungen reduzieren.



Claude Elwood Shannon wurde am 30. April 1916 in Petoskey geboren, einem Küstenstädtchen in Michigan, am Rande der Großen Seen. Aufgewachsen ist er im nahe gelegenen Gaylord und besuchte dort die Highschool. 1932 schrieb er sich als Student an der University of Michigan ein und studierte bis 1936 Mathematik und Elektrotechnik. In beiden Fächern erhielt er einen Bachelorabschluss. Anschließend setzte er sein Studium am Massachusetts Institute of Technology (MIT) in Boston fort, wo ihm 1937 der erste große Wurf gelang. In seiner Masterthesis zeigte er, dass sich Hardwareschaltungen durch aussagenlogische Formeln beschreiben lassen und sich damit das vollständige mathematische Instrumentarium der booleschen Algebren für die Optimierung digitaler Schaltkreise verwenden lässt [78, 79]. Jeder Student der informatiknahen Fächer ist heute mit dieser Theorie vertraut, auch wenn ihm Shannons Name in diesem Zusammenhang vielleicht nie zu Ohren gekommen ist.

1940 reichte Shannon am MIT eine Doktorarbeit über die theoretischen Grundlagen der Mendel'schen Vererbungstheorie ein [80], die für die Entwicklung der modernen Genetik aber unbedeutend war. Nach seiner Promotion folgte ein kurzer Aufenthalt am *Institute for Advanced Study* (IAS) in Princeton. Das IAS ist eines der renommiertesten Forschungsinstitute der USA, das in der Vergangenheit namhaften Größen wie Albert Einstein, Kurt Gödel, Robert Oppenheimer und John von Neumann eine wissenschaftliche Heimat bot.

1941 wechselte Shannon zu den AT&T Bell Labs in New Jersey, wo er sich unter dem Druck des zweiten Weltkriegs intensiv mit kryptoanalytischen Fragestellungen befasste. Es

ist ein kurioses Detail der Geschichte, dass ihn ausgerechnet die Beschäftigung mit diesen Themen jene elementaren Zusammenhänge entdecken ließ, aus denen er wenige Jahre später die Informationstheorie entwickelte.

Publiziert hatte er seine Theorie der Information im Jahr 1948 in seiner berühmten Arbeit mit dem Titel „A mathematical theory of communication“. Dieses Werk markiert den Höhepunkt seiner Karriere.

1956 zog Shannon erneut nach Boston und arbeitete dort bis zu seiner Emeritierung im Jahr 1978 als Professor am MIT. Abseits seiner wissenschaftlichen Tätigkeiten war er für seine außergewöhnlichen Hobbys bekannt, wie das Jonglieren oder Einradfahren. Gleichermaßen sorgte Shannon mit einer Reihe mechanischer Apparaturen für Aufmerksamkeit, die er mit großem Elan in seiner Freizeit konstruierte. Hierzu gehörten eine mechanische Maus, die sich autonom den Weg durch ein Labyrinth bahnen konnte, sowie ein früher Vorläufer des Schachcomputers.

Claude Elwood Shannon starb am 24. Februar 2001. Die digitale Revolution, die einen großen Teil seiner theoretischen Überlegungen zum Leben erweckt hatte, war zu dieser Zeit bereits in vollem Gang, und dennoch kam sie für ihn zu spät. Seine letzten Lebensjahre waren von einer schweren Alzheimer-Krankheit geprägt, die Shannons brillanten Geist schon vor seinem Ableben brutal in die Knie zwang. Er hinterließ seine Frau, einen Sohn und eine Tochter.

Für sein Lebenswerk hat Shannon weltweit große Anerkennung erfahren. „Pionier des Computerzeitalters“, „Vater des Bits“, „Gründer der Informationstheorie“, „eines der größten Genies des 20. Jahrhunderts“: Dies sind nur einige Superlative, die mit seinem Namen gerne verbunden werden.

verknüpft werden. Die bekanntesten Grundoperationen sind die Negation, die Konjunktion (UND-Verknüpfung) und die Disjunktion (ODER-Verknüpfung) (Abbildung 1.57). Werden sie in der richtigen Art und Weise miteinander kombiniert, so lassen sich damit alle erdenklichen Hardwareschaltungen konstruieren. Sie sind das Grundlixier der digitalen Schaltungstechnik – damals wie heute.

Die Vorstellung lag damals fern, die Übermittlung eines Telefongesprächs könnte irgend etwas mit den Rechenprogrammen der ersten Computer zu tun haben, und so wurden die Entwicklungen in den beiden Technologiegebieten zunächst unabhängig voneinander vorange-

## A Mathematical Theory of Communication

By C. E. SHANNON

### INTRODUCTION

THE recent development of various methods of modulation such as PCM and PPM which exchange bandwidth for signal-to-noise ratio has intensified the interest in a general theory of communication. A basis for such a theory is contained in the important papers of Nyquist<sup>1</sup> and Hartley<sup>2</sup> on this subject. In the present paper we will extend the theory to include a number of new factors, in particular the effect of noise in the channel, and the savings possible due to the statistical structure of the original message and due to the nature of the final destination of the information.

The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point. Frequently the messages have *meaning*; that is they refer to or are correlated according to some system with certain physical or conceptual entities. These semantic aspects of communication are irrelevant to the engineering problem. The significant aspect is that the actual message is one *selected from a set of possible messages*. The system must be designed to operate for each possible selection, not just the one which will actually be chosen since this is unknown at the time of design.

Published in THE BELL SYSTEM TECHNICAL JOURNAL  
 Vol. 27, pp. 379-423, 623-656, July, October, 1948  
 Copyright 1948 by AMERICAN TELEPHONE AND TELEGRAPH Co.  
 Printed in U. S. A.

**Abb. 1.58:** In „A mathematical theory of communication“ hat Claude Shannon das theoretische Fundament gelegt, auf dem die moderne Kommunikationstechnik errichtet ist. Die Arbeit wurde 1948 in zwei Teilen im *Bell System Technical Journal* publiziert [81] und ein Jahr später von der University of Illinois Press unter dem Titel *The mathematical theory of communication* in Buchform verlegt [82].

Ist Ihnen die unscheinbare Namensänderung aufgefallen, die sich im Jahr 1949 vollzogen hat? Bereits wenige Wochen nach dem Erscheinen der ursprünglichen Arbeit war klar, wie universell Shannons Ergebnisse tatsächlich waren, und so wurde schon bald nicht mehr von *einer*, sondern von *der* Theorie der Kommunikation gesprochen.

trieben. Noch ahnte niemand, wie tiefgründig beide Welt in Wirklichkeit miteinander verflochten sind.

1948 gelang der Brückenschlag. In diesem Jahr veröffentlichte Claude Shannon eine wissenschaftliche Arbeit mit dem Titel „A mathematical theory of communication“ (Abbildung 1.58). Die renommierte Zeitschrift *Scientific American* bezeichnete die Arbeit 1990 als die *Magna Carta* des Informationszeitalters [43], und die Autoren übertreiben damit in keiner Weise. Mit der Publikation einer einzigen Arbeit hatte Shannon eine neue Forschungsrichtung begründet, die wir heute als *Informationstheorie* bezeichnen; sie ist die theoretische Grundlage, auf der die gesamte moderne Datenübertragungs- und Speichertechnik beruht. Tatsächlich markiert die Shannon'sche Arbeit eine Zeitenwende in der bewegten Geschichte der Telekommunikation. Die rigorose mathematische Sichtweise, die mit ihr in den Vordergrund trat, machte die einstige Ingenieurdisziplin zu einer Wissenschaft.

Im Gegensatz zu den meisten anderen Wissenschaftlern seiner Zeit betrachtete Shannon das Problem der Datenübertragung von einem abstrakten Standpunkt aus. Auf diese Weise war es ihm gelungen, Er-

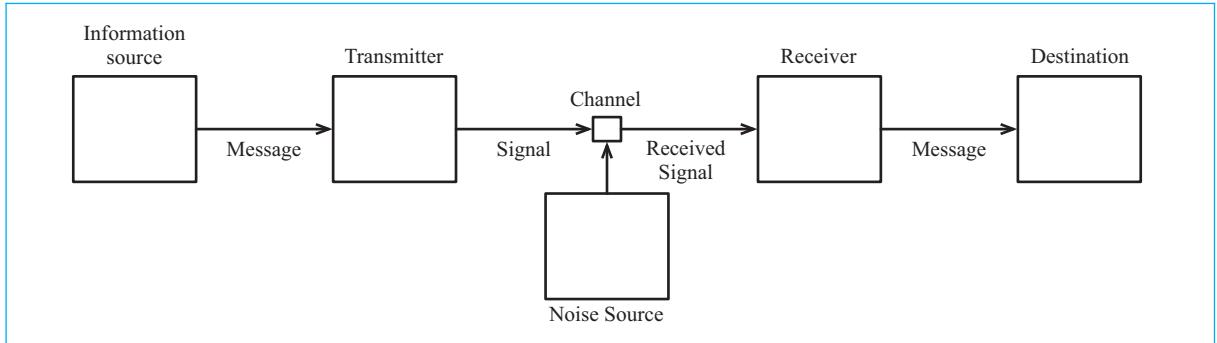
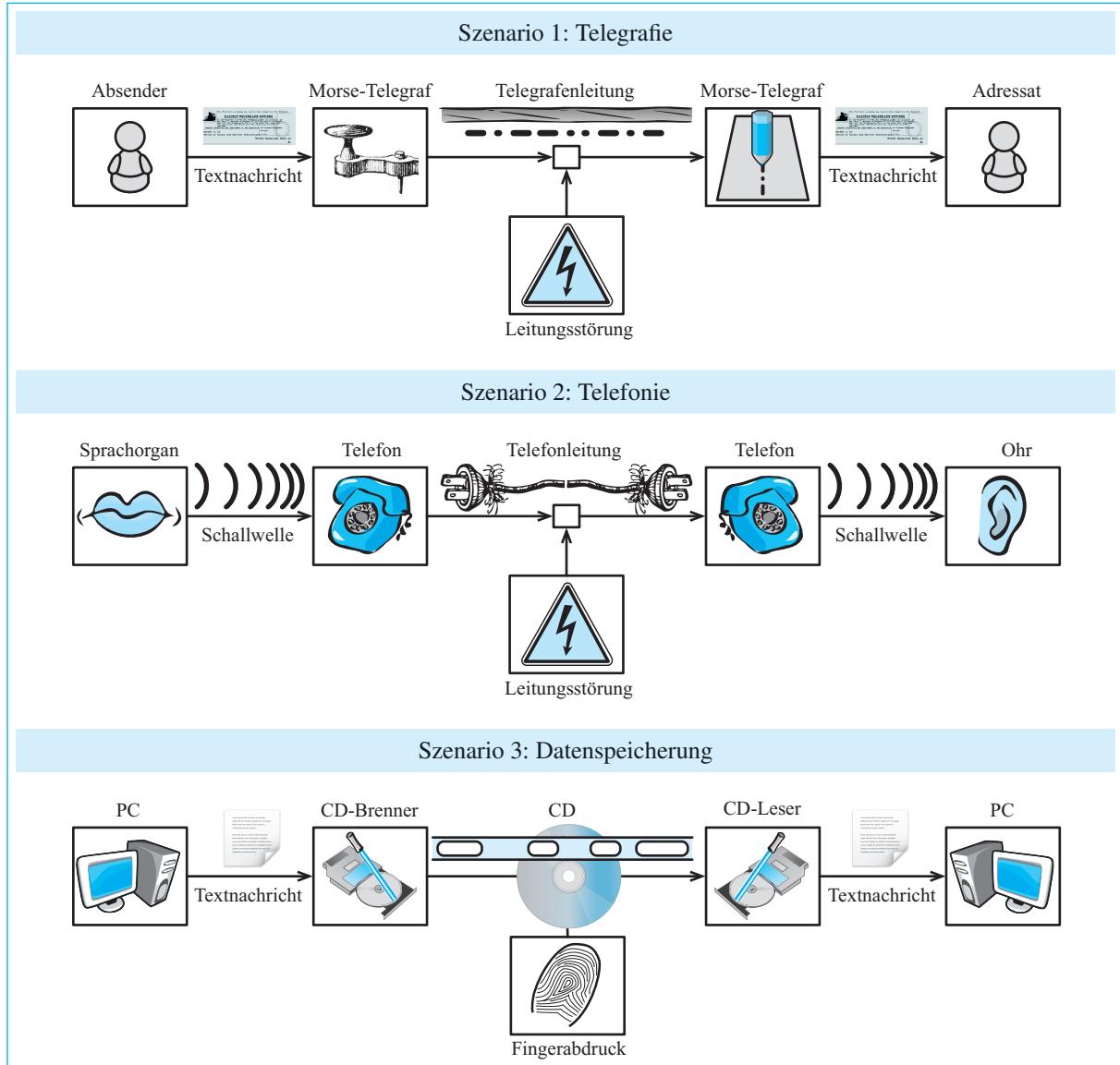


Abb. 1.59: Das Shannon'sche Kommunikationsmodell aus dem Jahr 1948 [81]

kenntnisse von so allgemeiner Natur abzuleiten, dass sie für alle Arten der Nachrichtenübertragung gelten, egal ob wir die digitale Übertragung eines Morse-Signals, die analoge Übertragung eines Telefongesprächs oder den Nachrichtenstrom eines modernen Kommunikationssatelliten betrachten. Es ist die Allgemeinheit seiner Ergebnisse, die viel tiefer in den abstrakten Begriff der *Information* hineinblicken ließ, als es jede anwendungsgetriebene Arbeit zuvor geschafft hatte.

Zu Beginn seiner Arbeit definiert Shannon ein allgemeines Kommunikationsmodell, das sich aus fünf Komponenten zusammensetzt (Abbildung 1.59). Eine davon ist die *Kommunikationsquelle* (*information source*). Sie wählt eine von mehreren möglichen Nachrichten aus und übergibt sie an den *Sender* (*transmitter*). Dort wird die Nachricht in ein Signal gewandelt und auf den *Übertragungskanal* (*communication channel*) geschickt. Am Zielort wird der Übersetzungsprozess in umgekehrter Richtung vollzogen. Der *Empfänger* (*receiver*) wandelt das Signal zurück in eine Nachricht und gibt sie weiter an die *Informationssenke* (*destination*). In einem realen Anwendungsszenario werden einige Nachrichten unverehrt den Empfänger erreichen, andere dagegen nicht. Die beeinflussenden Faktoren, die zu einer Verfälschung der gesendeten Signale führen können, modellierte Shannon über eine *Störquelle* (*noise source*) in der Mitte der Übertragungsstrecke.

Die in Abbildung 1.60 dargestellte Auswahl zeigt, welche Bedeutung den abstrakten Komponenten in realen Anwendungsszenarien kommt. In der Telegrafie produziert die Informationsquelle beispielsweise eine Nachricht in Form eines geschriebenen Textes, und die Informationssenke nimmt diese auch in der gleichen Form wieder entgegen. Der Sender hat hier die Aufgabe, das geschriebene Wort in Morse-Codewörter zu übersetzen, die anschließend als elektrische Signale die



**Abb. 1.60:** Shannons Kommunikationsmodell ist allgemein genug, um alle typischen Anwendungsszenarien zu erfassen.

Übertragungsstrecke durchlaufen. Der Empfänger macht die Übersetzung rückgängig, indem er aus den empfangenen Signalen die Originallnachricht extrahiert.

Das zweite Beispiel demonstriert das Gesagte anhand eines Telefongesprächs. Hier entsprechen die Informationsquelle und die Informationssenke dem menschlichen Sprach- und Hörorgan, und hinter dem Sender und dem Empfänger verbergen sich diejenigen Baukomponenten eines Festnetz- oder Mobiltelefons, die für die Umsetzung der Schallwellen in elektrische Signale bzw. deren Rückwandlung zuständig sind.

Das letzte Beispiel macht deutlich, dass wir das Shannon'sche Kommunikationsmodell genauso gut auf Szenarien aus dem Gebiet der Datenspeicherung anwenden können. Die Übertragungsstrecke wird in diesen Fällen nicht durch ein Medium des räumlichen Transports gebildet, sondern durch einen Datenträger, der die Signale über die Zeit persistiert.

Das Aufstellen eines abstrakten Kommunikationsmodells war der erste Schritt zu Shannons allgemeiner Theorie der Information, doch das grundlegend Neue war etwas anderes: Mitte der 40er-Jahre begann Shannon zu verstehen, dass eine übertragene Nachricht unabhängig von ihrem Kommunikationsmedium betrachtet werden kann. Im Mittelpunkt seiner Überlegungen stand nicht mehr länger das Medium selbst, sondern die *Information*, und es ist Shannons Verdienst, dass wir heute einen präzisen mathematischen Zugang zu diesem vormals vage definierten Begriff besitzen. In Shannons Theorie der Information ist das *Bit* die fundamentale Einheit, und wir wissen heute, dass sich jede Nachricht, ob analog oder digital, auf eine Folge von Nullen und Einsen reduzieren lässt.

Mit seiner entwickelten Theorie hat Shannon die verschiedenen Arten der Nachrichtenübertragung aber nicht nur in der Art ihrer Betrachtung zusammengeführt. Er konnte gleichermaßen zeigen, dass die verschiedenen Übertragungsmedien, so unterschiedlich sie von außen auch erscheinen, den gleichen fundamentalen Gesetzmäßigkeiten unterliegen. Im Kern waren seine Untersuchungen von zwei Fragestellungen geprägt:

- Wie viel Information enthält eine Nachricht? In seiner Arbeit führt Shannon den Begriff der *Entropie* einer Datenquelle ein und konnte den Informationsgehalt von Nachrichten hierüber quantitativ erfassen. Diese Untersuchungen sind von unschätzbarem Wert für die Datenkompression. Sind die Nachrichten eines Senders so gestaltet, dass ihre Symbolabfolgen gewisse statistische Strukturen aufweisen, so können wir mithilfe der Shannon'schen Formeln die Anzahl an Bits ausrechnen, die zur Darstellung einer solchen Nachricht unbedingt benötigt wird. Damit versetzt uns die Shannon'sche Theorie in die Lage, die Güte von Algorithmen aus dem Bereich Datenkompression quantitativ zu beurteilen.

- Nahezu alle in der Praxis angetroffenen Kommunikationskanäle sind rauschbehaftet; wir müssen also stets damit rechnen, dass ein Bit während der Übertragung seinen Wert verändert. Um Übertragungsfehler zu kompensieren, wurden in der Vergangenheit zahlreiche Codierungen ersonnen, mit denen sich Fehler erkennen oder korrigieren lassen. Es ist leicht einzusehen, dass kein solches Verfahren perfekt sein kann, d. h., die Wahrscheinlichkeit, dass ein Übertragungsfehler unerkannt bleibt, ist stets größer 0. Shannon hat einen großen Teil seiner Arbeit der Übertragung von Nachrichten über rauschbehaftete Kanäle gewidmet und dabei eine Entdeckung gemacht, die zu den wissenschaftlichen Sternstunden des 20. Jahrhunderts gehört. Noch fehlt uns das notwendige Begriffsgerüst, um sein fulminantes Ergebnis in der gebührenden Präzision zu verstehen, und so müssen wir uns noch eine Weile Geduld üben. In Kapitel 7 ist es dann so weit. In Abschnitt 7.5 werden wir Shannons berühmtes *Kanalcodierungstheorem (Noisy-Channel Coding Theorem)* ausführlich besprechen.

Mit seinen theoretischen Überlegungen hatte Shannon bewiesen, dass für viele Probleme der Kommunikationstechnik Lösungen existieren, allerdings sind die angestellten Überlegungen an bedeutenden Stellen nicht konstruktiv. Im Sinne eines klassischen mathematischen Existenzbeweises konnte er beispielsweise zeigen, dass Codierungen mit bestimmten Eigenschaften existieren; auf die Frage, wie diese Codierungen aussehen, geschweige denn, wie sie zu berechnen sind, liefert seine Argumentationskette aber keine praktisch verwertbare Antwort. In diesem Licht der Dinge ist zu verstehen, dass sich in den Folgejahren unzählige Forscher in einem spannenden Wettlauf auf die Suche nach Codierungen begaben, die im Shannon'schen Sinne optimal sind. Es war die Suche nach dem heiligen Gral der Informationstheorie.

Die Abbildungen 1.61 und 1.62 geben einen groben Überblick über die Entwicklung in der zweiten Hälfte des 20. Jahrhunderts. Den Anfang machte Richard Hamming im Jahr 1950. In [39] publizierte er mehrere Codes, die wir heute, seinem Namen zu Ehren, als *Hamming-Codes* bezeichnen. Entwickelt hatte er sie bereits im Jahr 1947, und auch in der Shannon'schen Arbeit von 1948 werden sie bereits erwähnt; aus patentrechtlichen Gründen erschien seine Publikation aber erst später. Hamming-Codes werden im Bereich der Fehlerkorrektur eingesetzt und sind Gegenstand von Abschnitt 6.4.2.

Im Jahr 1949 publizierte Robert M. Fano eine Arbeit mit dem Titel „Transmission of Information“ und schlug dort eine Codierung vor, die im Bereich der Datenkompression eine wichtige Rolle spielt. Auch sie war 1948 bereits bekannt und wird in der Shannon'schen Arbeit eben-

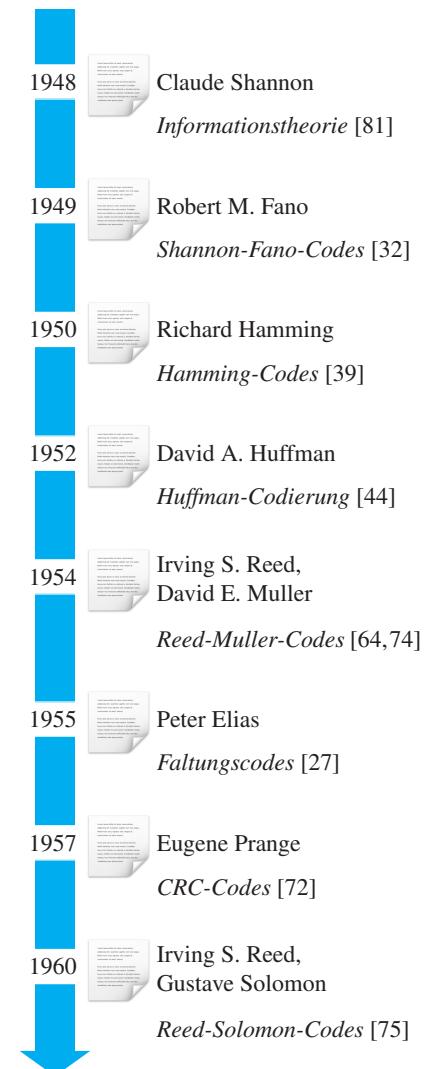
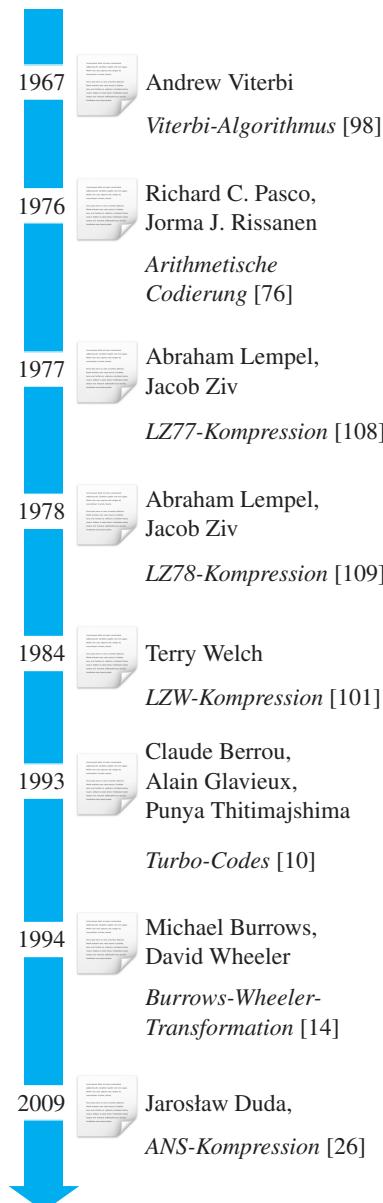


Abb. 1.61: Meilensteine der Informationstheorie



**Abb. 1.62:** Meilensteine der Informationstheorie (Fortsetzung)

falls erwähnt. Wir werden die Codierung in Abschnitt 4.4.2 genauer untersuchen und dabei aufzeigen, dass sie nicht in jedem Fall optimale Ergebnisse liefert. Abhilfe bringt die Huffman-Codierung, die wir in Abschnitt 4.4.3 besprechen. Sie stammt aus dem Jahr 1952 und ist eines der wichtigsten Verfahren in der modernen Datenkompression [44].

1954 schufen Irving S. Reed und David E. Muller die Klasse der *Reed-Muller-Codes*, die wir in Abschnitt 6.4.8 genauer untersuchen. Ein Jahr später führte Peter Elias mit den *Faltungscodes (convolutional codes)* eine wichtige Klasse fehlerkorrigierender Codes ein, der wir uns in Abschnitt 6.5 zuwenden.

1957 wurden Codes entdeckt, die auf dem Prinzip der *zyklischen Redundanzprüfung (Cyclic Redundancy Check, kurz CRC)* beruhen. Sie spielen heute eine wichtige Rolle im Bereich der Fehlererkennung und werden unter anderem eingesetzt, um die Kommunikation zweier PCs in einem Netzwerk oder den Datenaustausch zweier Steuergeräte in einem Kraftfahrzeug abzusichern. In Abschnitt 6.4.3 kommen wir auf diese wichtige Codeklasse zurück.

1960 schlugen Irving S. Reed und Gustave Solomon fehlerkorrigierende Codes vor, die auf den algebraischen Eigenschaften von Polynomen beruhen. Verschiedene Weiterentwicklungen der Reed-Solomon-Codes sorgen beispielsweise dafür, dass sich die Daten einer CD oder DVD auch dann noch fehlerfrei decodieren lassen, wenn die Oberfläche durch Kratzer oder Staub an mehreren Stellen unlesbar geworden ist. Mit den Reed-Solomon-Codes beschäftigen wir uns ausführlich in Abschnitt 6.4.5.

1967 ebnete Andrew Viterbi den Weg für die praktische Verwertung von Faltungscodes. Erst durch den *Viterbi-Algorithmus*, den wir in Abschnitt 6.5.1 besprechen, war es möglich, Übertragungsfehler mit geringem Rechenaufwand zu erkennen und zu korrigieren.

Die 70er-Jahre standen ganz im Zeichen von neuen Algorithmen zur Datenkompression. Den Anfang machte die Entdeckung der *arithmetischen Codierung* im Jahr 1976 (Abschnitt 4.5). Ein Jahr später publizierten Abraham Lempel und Jacob Ziv mit dem *LZ77-Algorithmus* ein semantisches Kompressionsverfahren auf der Basis dynamisch erzeugter Wörterbücher. 1978 präsentierten sie mit dem *LZ78-Algorithmus* ein alternatives Verfahren, das 1984 von Terry A. Welch aufgegriffen und weiter optimiert wurde. Eine genaue Analyse dieser Algorithmen nehmen wir in den Abschnitten 4.7.1, 4.7.3 und 4.7.4 vor.

Weitere wichtige Entdeckungen folgten in den 90er-Jahren. Im Jahr 1993 präsentierten Claude Berrou, Alain Glavieux und Punya Thitimajshima mit dem *Turbo-Codes* eine neue Klasse fehlerkorrigierender Codes, die einen hohen Fehlerkorrigierungsgrad bei geringem Rechenaufwand ermöglicht. Diese Algorithmen haben sich in den letzten Jahren zu einem Standard in der drahtlosen Kommunikation entwickelt.

jshima mit den *Turbo-Codes* eine Weiterentwicklung der Faltungscodes, die sehr nahe an das von Shannon entdeckte Optimalitätslimit heranreichen.

1994 veröffentlichten Michael Burrows und David Wheeler die *Burrows-Wheeler-Transformation*, die heute in vielen Anwendungsszenarien als Vorverarbeitung zum Einsatz kommt. In Abschnitt 4.8 werden wir uns ausführlich mit dieser Transformation beschäftigen, die durch eine ungewöhnliche Eleganz und Einfachheit besticht.

2009 entwickelte der polnische Informatiker Jarosław Duda eine Familie von Algorithmen, die an die Güte der arithmetischen Codierung heranreicht, aber kaum langsamer ist als die ohnehin sehr schnelle Huffman-Codierung [26]. Die unter dem Namen ANS (*Asymmetric Numeral Systems*) bekannten Algorithmen wurden von der Industrie rasch aufgegriffen und sind mittlerweile großflächig in praktischer Verwendung. In Abschnitt 4.6 gehen wir im Detail auf die Funktionsweise dieser neuen Algorithmen ein.

Hier findet unsere Reise in die Geschichte der Nachrichtentechnik ihr vorläufiges Ende. Es war kein kurzer Rückblick, und trotzdem ist er weit davon entfernt, ein vollständiges Bild zu zeichnen. Jede der von uns besuchten Etappen ist vielschichtig genug, um ein eigenes Kapitel oder gar ein ganzes Buch zu füllen. Am Ende dieses Kapitels möchte ich deshalb auf mehrere Werke verweisen, die sich ausführlich mit der Geschichte der Nachrichtentechnik als Ganzes oder mit einer bestimmten Etappe beschäftigen.

Eine exzellente wissenschaftliche Abhandlung über die Geschichte der Nachrichtentechnik, von den Anfängen bis in die Mitte des 20. Jahrhunderts, hat uns Volker Aschoff in einem zweibändigen Werk hinterlassen [3, 4]. Ein ähnlich guter Rundumschlag ist Anton A. Huurdeman in [45] gelungen. Auf das Allerwärmste möchte ich jedem Leser das Buch von Tom Standage [86] und das Buch von Lewis Coe [18] empfehlen. Beide sind spannend geschrieben und lassen das Zeitalter der Telegrafie in bunten Farben auferstehen. Derjenige Leser, der sich noch tiefer in die Materie einarbeiten möchte, findet in [8, 19, 42, 55, 56, 93] weiteren wertvollen Lesestoff.

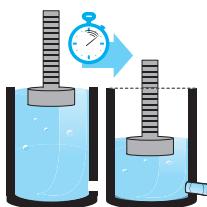
## 1.8 Übungsaufgaben

### Aufgabe 1.1



**Webcode  
1045**

Zu Beginn dieses Kapitels haben wir den Synchrontelegrafen des Aineias diskutiert. In dieser Aufgabe wollen wir überlegen, wie effizient die Nachrichtenübertragung war.

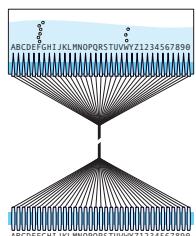


- Aus den Aufzeichnungen des Polybios wissen wir, dass die verwendeten Gefäße annähernd 150 cm hoch waren und ihr Durchmesser ungefähr 50 cm betrug. Die Abflussöffnung hatte einen Durchmesser von ca. 5 mm. Versuchen Sie abzuschätzen, wie lange es gedauert haben muss, das Gefäß vollständig zu entleeren.
- Könnte man die Apparatur durch eine Änderung der Gefäßgeometrie verbessern?
- Wäre es möglich gewesen, mit dem Synchrontelegrafen beliebige Texte zu übertragen?

### Aufgabe 1.2



**Webcode  
1104**



In Abschnitt 1.3 haben Sie gelernt, dass der Soemmerring-Telegraf auf dem chemischen Prinzip der Elektrolyse beruhte. Sicher erinnern Sie sich daran, dass Buchstaben paarweise übertragen wurden. Wie konnte der Empfänger wissen, in welcher Reihenfolge er die gleichzeitig ankommenden Buchstaben notieren muss?

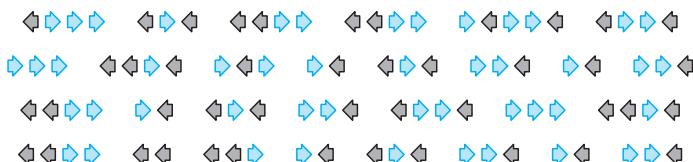
Hinweis: Für die Beantwortung dieser Frage benötigen Sie ein wenig Grundwissen aus der Chemie.

### Aufgabe 1.3



**Webcode  
1300**

In Abschnitt 1.3.1 haben Sie den Nadeltelegrafen von Gauß und Weber kennengelernt. Hier ist eines der ersten Telegramme, die beide Professoren untereinander austauschten:



- Decodieren Sie das Telegramm.
- Gauß und Weber sahen vor, dass zwischen den einzelnen Zeichen und Wörtern Pausen eingefügt werden, um eine fehlerfreie Übertragung zu ermöglichen. Welche Probleme entstehen, wenn die künstlichen Pausen eingespart würden? Welche Lösungsmöglichkeiten fallen Ihnen ein?

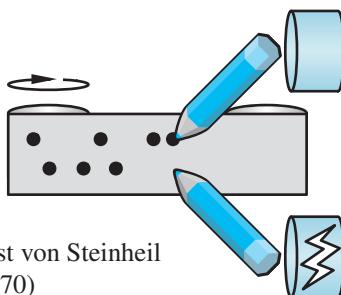
Der Telegraf von Gauß und Weber war eine große und unhandliche Apparatur, die ausschließlich dazu dienen konnte, die prinzipielle Machbarkeit der elektrischen Datenübertragung zu demonstrieren. Dennoch wussten die beiden Forscher um das Potenzial ihrer Erfindung. Sie teilten ihre Ergebnisse dem Münchener Professor Carl August von Steinheil mit, der umgehend mit dem Bau einer verbesserten Apparatur begann. Der *Steinheil-Telegraf* besaß zwei übereinander positionierten Stifte, an denen ein Papierstreifen vorbeigeführt wurde. Die Stifte konnten durch Elektromagnete unabhängig voneinander bewegt werden und hinterließen auf dem Papierstreifen ein zweizeiliges Punktemuster (vgl. [56]).

## Aufgabe 1.4



Webcode

1305



## Carl August von Steinheil (1801 – 1870)

Steinheil sah vor, Nachrichten entsprechend der nachstehenden Tabelle zu codieren [56]:

A	•••	I	•	T	•	3	••••
B	•••••	L	••••	U	••••	4	•••••
C, K	•..	M	••••	W	••..	5	••..•
D	••	N	•••	Z	••..	6	•..•..
E	..	O	••••	Ch	•••••	7	•..••
F	•••	P	••..•	Sch	••••	8	••••
G	••..•	R	..•	1	•••••	9	••..•
H	•••••	S	••..•	2	•••••	0	•••••

- a) Entziffern Sie die folgende Nachricht:

The image shows a single horizontal row of Braille characters. Each character consists of a 2x3 grid of dots. The sequence of characters represents the word "WATER". The first character has dots in positions (1,1), (1,2), (1,3), (2,1), and (2,3). The second character has dots in positions (1,1), (1,2), (1,3), (2,1), and (2,2). The third character has dots in positions (1,1), (1,2), (1,3), (2,1), and (2,3). The fourth character has dots in positions (1,1), (1,2), (1,3), (2,1), and (2,2). The fifth character has dots in positions (1,1), (1,2), (1,3), (2,1), and (2,2). The sixth character has dots in positions (1,1), (1,2), (1,3), (2,1), and (2,2).

- b) Worin unterscheidet sich der Steinheil-Telegraf vom Morse-Telegraf?

## Aufgabe 1.5

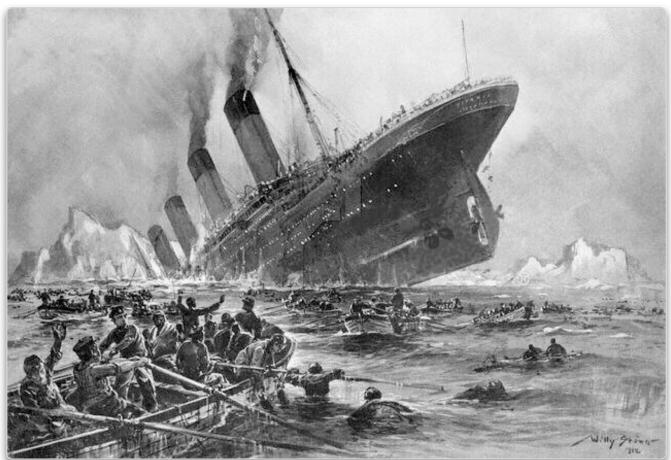


## **Webcode 1423**

In der Nacht des 14. April 1912 ereignete sich 400 Meilen vor der Küste Neufundlands eine der schlimmsten Katastrophen in der Geschichte der Seefahrt. Um 23:40 wurde der Rumpf der RMS Titanic auf ihrer Jungfernreise von Southampton nach New York durch die Kollision mit einem Eisberg an mehreren Stellen aufgerissen. Nachdem die Besatzung die bedrohliche Situation begriff, versuchte sie, Schiffe in der Umgebung auf ihre Notlage aufmerksam zu machen. Unter anderem telegraфиerte die Titanic wieder und wieder die folgende Botschaft:

Die Nachricht blieb nicht ungehört. Gegen 0:30 wurden die Signale von der 58 Meilen entfernten RMS Carpathia empfangen, die umgehend Kurs in Richtung der Titanic setzte. Trotz voller Kraft schaffte es die Carpathia nicht, die Unglücksstelle rechtzeitig zu erreichen. Am frühen Morgen des 15. April 1912 war der mehr als zwei Stunden währende Kampf verloren. Die Titanic konnte dem eindringenden Wasser nicht mehr standhalten und sank auf den Meeresgrund, wo sie noch heute liegt. Nur 706 der 2223 Menschen an Bord schafften es, sich in den wenigen Rettungsbooten in Sicherheit zu bringen. Für 1517 Menschen gab es keine Rettung; sie verloren ihr Leben im eiskalten Wasser des Atlantischen Ozeans.





- a) Decodieren Sie die Morse-Nachricht der Titanic.
  - b) Finden Sie die Bedeutung der einzelnen Worte heraus.

Das Schicksal der Titanic war von besonderer Tragik: Nur 10 Meilen von der Unglücksstelle entfernt befand sich die SS California. Das Schiff war nahe genug, um den Havaristen zu erreichen und hatte genug Rettungsboote an Bord, um alle Passagiere zu retten. Leider war die Telegrafenstation in dieser Nacht unbesetzt und die Leuchtraketen der Titanic wurden von der Besatzung der California als feierliches Feuerwerk interpretiert.

In den Anfangstagen der Morse-Telegrafie dachte noch niemand an die Notwendigkeit einer abhörsicheren Übertragung. Nachrichten wurden stets im Klartext telegrafiert, und genau dies wurde mit der zunehmenden Kommerzialisierung der Übertragungsdienste zu einem ernstzunehmenden Problem. Als Abhilfe verschlüsselten viele Firmen ihre Nachrichten vor dem Versenden mit einem individuellen Code. Der Einsatz solcher Codes brachte einen positiven Nebeneffekt mit sich: Durch die Verwendung knapper Kunstwörter wurden viele Textnachrichten kürzer, und so sank der Preis für ihre Übertragung.

## Aufgabe 1.6



## Webcode

1599

Dass eine solche Umcodierung nicht nur Vorteile mit sich bringt, bekam der Wollhändler Frank J. Primrose am eigenen Leib zu spüren. Um Kosten zu sparen, verwendete er die Abkürzung BAY für I HAVE BOUGHT und die Abkürzung QUO für 500.000 POUNDS. Im Juni 1887 ließ Primrose die folgende Nachricht von Philadelphia zu seinen Kaufleuten nach Kansas telegrafieren [86]:

In der Telegrafenstation in Kansas wurde die Nachricht wie folgt empfangen:

Wahrscheinlich aus Versehen wurde bei der Übertragung des zweiten Worts ein zusätzlicher Punkt übertragen.

- a) Entschlüsseln Sie die gesendete und die empfangene Nachricht und klären Sie auf, warum Primrose an diesem Tag die stattliche Summe von 20.000 \$ verlor.
  - b) Wie hängt Ihrer Meinung nach die Reduktion von Redundanz mit der Fähigkeit zusammen, Fehler zu erkennen?

Eine kleine Notiz am Rande: Mit dem Versuch, den Verlust von der Western Union Telegraph Company auf juristischem Wege zurückzuerhalten, scheiterte Primrose auf ganzer Linie. Das Gericht verurteilte die Firma lediglich dazu, dem Kläger die Kosten von 1.50 \$ für das Senden des Telegramms zu erstatten [86].

## Aufgabe 1.7



## Webcode

1648

In Abschnitt 1.5 haben Sie gelernt, dass sich die Kapazität einer Telegrafenleitung im Duplexbetrieb verzweifachen lässt. Zwei Jahre nach dieser Entdeckung fand der französische Ingenieur Jean-Maurice-Émile Baudot eine Möglichkeit, die Menge der übertragenen Zeichen weiter zu steigern. Die Telegrafenleitung wurde durch einen Multiplexer gespeist, an den bis zu sechs einzelne Sender angeschlossen waren. Der Multiplexer hatte die Aufgabe, den Ausgang von jeweils einem Sender auf die Telegrafenleitung zu schalten und die anderen Sender in dieser Zeit zu blockieren (*Zeitmultiplexverfahren*). Nach dem Rotationsprinzip wurde im ersten Zeitfenster der erste Sender verbunden, im zweiten

Zeitfenster der zweite und so fort. Die einzelnen Sender arbeiteten im Duplexbetrieb, sodass es mit der Baudot-Apparatur möglich war, eine physikalische Leitung in zwölf virtuelle Leitungen aufzuspalten.

Baudot wollte einen Telegrafen konstruieren, der Nachrichten automatisch codieren und decodieren kann, und verzichtete aus diesem Grund bewusst auf den Einsatz des Morse-Alphabets. Der von ihm verwendete Code war ein früher Vertreter dessen, was wir heute als *bitorientierte Zeichencodes fester Länge* bezeichnen. Er fällt damit in eine Klasse, zu der auch viele der heute gebräuchlichen Codes wie ASCII oder der Unicode gehören. Ferner ist der *Baudot-Code* ein sogenannter *Umschaltcode*. Er unterscheidet zwischen einer Buchstaben- und einer Zahlentabelle:

Buchstabentabelle				Zahlentabelle					
A	· · · ·	F	· · · · ·	R	· · · · ·	1	· · · ·	1 /	· · · · ·
É	· · · · ·	G	· · · · ·	S	· · · · ·	2	· · · ·	2 /	· · · · ·
E	· · · · ·	H	· · · · ·	T	· · · · ·	3	· · · ·	3 /	· · · · ·
I	· · · · · ·	J	· · · · ·	V	· · · · ·	4	· · · · ·	4 /	· · · · ·
O	· · · · ·	K	· · · · ·	W	· · · · ·	5	· · · · ·	5 /	· · · · ·
U	· · · · ·	L	· · · · ·	X	· · · · ·	6	· · · · ·	7 /	· · · · ·
Y	· · · · ·	M	· · · · ·	Z	· · · · ·	7	· · · · ·	9 /	· · · · ·
B	· · · · ·	N	· · · · ·	—	· · · · ·	8	· · · · ·	.	· · · · ·
C	· · · · ·	P	· · · · · ·			9	· · · · ·	,	· · · · ·
D	· · · · · ·	Q	· · · · · ·			0	· · · · ·	:	· · · · ·
Steuerzeichen									
Leerzeichen und gleichzeitiges Umschalten auf die Buchstabentabelle								· · · · ·	
Leerzeichen und gleichzeitiges Umschalten auf die Zahlentabelle								· · · · ·	
Löschen des letzten Zeichens								· · · · ·	

Welche der beiden Tabellen zur Decodierung des aktuell gelesenen Bitmusters verwendet wird, entscheidet das vorangegangene Leerzeichen. Für dieses hält der Baudot-Code zwei Codewörter vor, die als Seiteneffekt das Hin- und Herschalten zwischen beiden Tabellen bewirken. Zusätzlich stellt der Baudot-Code ein spezielles Bitmuster zum Löschen des vorangegangenen Zeichens bereit.

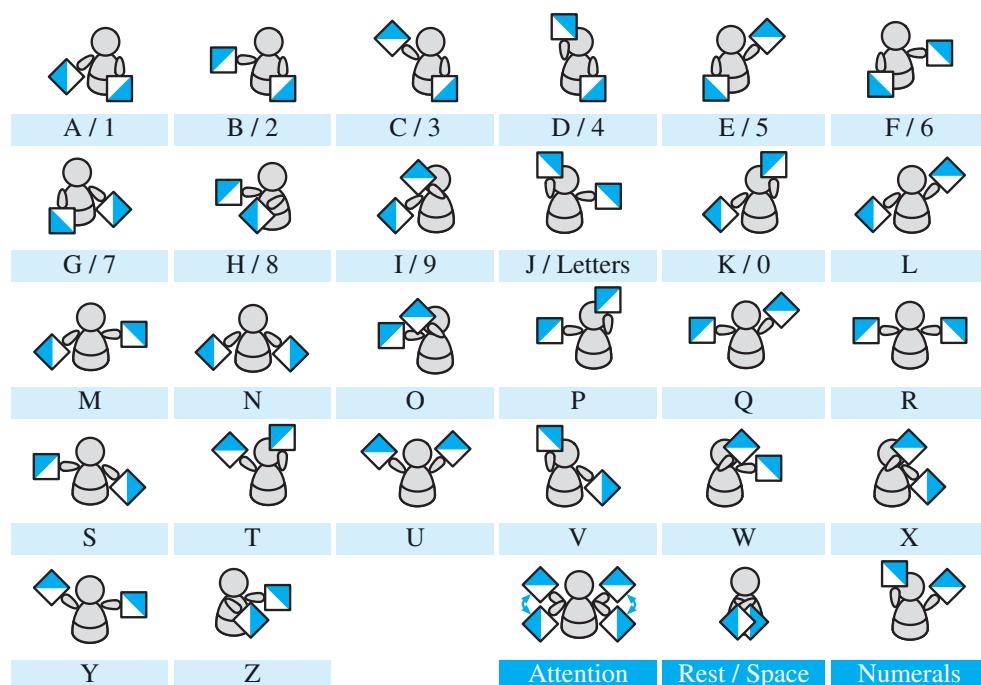
- a) Wie lässt sich sicherstellen, dass die Decodierung immer mit der Zahlentabelle beginnt?
  - b) Entziffern Sie die folgende Nachricht:

...8... 88... 8...8 888 8...8 8...8 888 88...8 8...8 888

Übrigens: Jean-Maurice-Émile Baudot ist der Namensgeber der Einheit *Baud*, die zur Angabe der Schrittgeschwindigkeit verwendet wird. Eine Schrittgeschwindigkeit von 1 Baud bedeutet, dass pro Sekunde genau ein Symbol übertragen wird.

Das Winkeralphabet ist, genau wie der Baudot-Code, ein Umschaltcode. Früher wurde es vor allem in der maritimen Kommunikation eingesetzt, verlor durch den Einsatz des Sprechfunsks dann aber weitgehend an Bedeutung. Heute ist diese Form der Nachrichtenübertragung, wenn überhaupt, nur noch im militärischen Umfeld anzutreffen. Das Winkeralphabet basiert auf der Idee, eine Nachricht mithilfe von zwei Flaggen zu übertragen, deren Winkelstellungen jeweils einen Buchstaben, eine Ziffer oder ein Steuerzeichen codieren.

Die Übertragung einer Nachricht beginnt mit dem Senden des *Attention-Signals*. Hierzu werden die Arme ausgebreitet und beide Flaggen gewunken. Dies geschieht so lange, bis der Empfänger das Zeichen K zurücksendet und hierdurch seine Empfangsbereitschaft signalisiert. Danach wird die Nachricht zeichenweise übertragen. Genau wie im Baudot-Code besteht auch hier die Möglichkeit, zwischen einer Zeichen- und einen Zahlencodierung umzuschalten. In die Zahlentabelle wird durch eine spezielle Flaggenstellung gewechselt, und das Zurückschalten wird durch das Senden des Buchstabens J bewirkt.



a) Decodieren Sie die folgende Nachricht:



### Aufgabe 1.8



Webcode

1729

- b) Wie schnell müssen die Flaggen gewechselt werden, um eine Übertragungsgeschwindigkeit von 10 Wörtern pro Minute zu erreichen? Benutzen Sie in der Berechnung PARIS als Referenzwort.

---

**Aufgabe 1.9****Webcode****1876**

Nach dem Moore'schen Gesetz verdoppelt sich die Anzahl der Transistoren, die auf einem einzigen Stück Silizium integriert werden, alle 20 Monate.

- a) In welchem Jahr würde die Anzahl der Transistoren auf einem einzigen Chip die geschätzte Anzahl der Elementarteilchen unseres Universums übersteigen? Schätzungen zufolge gibt es im Universum ca.  $10^{87}$  Elementarteilchen.
- b) Offensichtlich muss das Gesetz von Moore irgendwann seine Gültigkeit verlieren. Sehen Sie darin einen Widerspruch zu seiner inhaltlichen Aussage?



## 2 Mathematische Grundlagen

In diesem Kapitel werden Sie ...

- mit Moduln rechnen,
- die algebraische Struktur von Gruppen, Körpern und Ringen verstehen,
- die Eigenheiten von endlichen Körpern erkunden und
- einen Querbezug zu den Polynomringen herstellen,
- Vektorräume konstruieren und deren Generator- und Kontrollmatrizen berechnen.

$$x^{-1} \circ x =$$

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

$$(x^{-1} \circ x) \circ (x^{-1} \circ$$

$$((x^{-1} \circ x) \circ x^{-1}) \circ$$

$$(x^{-1} \circ (x \circ$$

$$(x^{-1} \circ$$

$$x^{n-1} - 1 = \prod_{\substack{\alpha \in \mathbb{F} \\ \alpha \neq 0}} (x - \alpha)$$

## 2.1 Motivation

In diesem Kapitel stellen wir die mathematischen Hilfsmittel bereit, die für ein tiefergehendes Verständnis der Informations- und Codierungstheorie unabdingbar sind; sie werden uns in den nachfolgenden Kapiteln auf Schritt und Tritt begleiten.

Wir beginnen in Abschnitt 2.2 mit einem Ausflug in die modulare Arithmetik, einem wichtigen Teilgebiet der Zahlentheorie. Danach wenden wir uns in Abschnitt 2.3 der Algebra zu. Dort werden wir mit Gruppen, Körpern und Ringen drei grundlegende algebraische Strukturen besprechen und dabei aufzeigen, dass die bekannten Polynome eine für unsere Zwecke besonders wichtige Ringstruktur bilden. Der eingeschlagene Weg wird uns in Abschnitt 2.4 zu jenen Strukturen führen, die in der Informations- und Codierungstheorie eine prominente Rolle spielen: die endlichen Körper. In Abschnitt 2.5 geht unser Ausflug in die Mathematik zu Ende. Dort werden wir die aus der linearen Algebra vertrauten Vektorräume einführen und diese mit den anderen Strukturen in Beziehung setzen.

In den folgenden Abschnitten bauen wir auf Mathematikkenntnissen im Umfang der allgemeinen Hochschulreife auf [12, 50, 87]. Insbesondere erwarten wir den sichere Umgang mit den folgenden Zahlenmengen:

- $\mathbb{N} := \{0, 1, 2, 3, \dots\}$  (☞ Menge der natürlichen Zahlen)
- $\mathbb{Z} := \{\dots, -2, -1, 0, 1, 2, \dots\}$  (☞ Menge der ganzen Zahlen)
- $\mathbb{Q} := \left\{ \frac{p}{q} \mid p, q \in \mathbb{Z}, q \neq 0 \right\}$  (☞ Menge der rationalen Zahlen)
- $\mathbb{R}$  (☞ Menge der reellen Zahlen)

Ein wichtiger Hinweis vorab: Die Menge der natürlichen Zahlen ist in der Literatur nicht eindeutig definiert. In manchen Fällen ist die 0 enthalten, in anderen Fällen ist die kleinste natürliche Zahl die 1. In diesem Buch ist die 0 ein Element von  $\mathbb{N}$ . Die Menge  $\mathbb{N} \setminus \{0\}$  bezeichnen wir als die Menge der *positiven Zahlen* und benutzen für sie das Symbol  $\mathbb{N}^+$ . Immer dann, wenn wir von *nichtnegativen Zahlen* reden, ist die Menge  $\mathbb{N}$  gemeint, d. h. die positiven Zahlen einschließlich der 0.

## 2.2 Modulare Arithmetik

Wir beginnen mit der Definition einer Menge, die für unsere Zwecke besonders wichtig ist: die Menge  $\mathbb{Z}_m$ . Sie enthält alle ganzzahligen Reste, die bei der Division einer natürlichen Zahl durch  $m$  entstehen können:

$$\mathbb{Z}_m := \{0, \dots, m-1\}$$

Mit den Elementen dieser Menge können wir fast genauso rechnen, wie wir es von den ganzen Zahlen her gewohnt sind; wir müssen lediglich darauf achten, bei der Addition oder der Multiplikation zweier Elemente nicht aus der Menge  $\mathbb{Z}_m$  herauszutreten. Dies gelingt, indem wir die Summe  $x+y$  oder das Produkt  $x \cdot y$  „modulo  $m$ “ berechnen, d.h., wir verwenden als Ergebnis immer den Rest, der bei der Division durch  $m$  entsteht (Abbildung 2.1). Aufgrund der Zerlegung

$$3+2 = 5 = 1 \cdot 5 + 0 \quad (2.1)$$

$$3 \cdot 2 = 6 = 1 \cdot 5 + 1 \quad (2.2)$$

ergibt  $3+2$  in  $\mathbb{Z}_5$  das Ergebnis 0 und  $3 \cdot 2$  das Ergebnis 1. Durch die Modulo-Berechnung führt die Addition und die Multiplikation aus der Menge  $\mathbb{Z}_m$  nicht hinaus. Wir sagen,  $,+'$  und  $,\cdot'$  sind auf dieser Menge abgeschlossen.

Bezeichnen wir den Rest, der bei der Division einer ganzen Zahl  $x$  durch eine ganze Zahl  $m$  entsteht, mit  $x \bmod m$ , dann können wir für die Addition und die Multiplikation in  $\mathbb{Z}_5$  die folgenden Schreibweisen verwenden:

$$3+2 = 5 \bmod 5 = 0 \quad (2.3)$$

$$3 \cdot 2 = 6 \bmod 5 = 1 \quad (2.4)$$

Als Nächstes führen wir eine weitere Sprech- und Schreibweise ein:



### Definition 2.1 (Kongruenz)

Zwei Zahlen  $x, y \in \mathbb{Z}$  heißen *kongruent modulo m*, geschrieben als

$$x \equiv y \pmod{m},$$

wenn ihre Division durch den *Modul m* denselben Rest ergibt.

Aus dieser Definition folgt, dass zwei ganze Zahlen  $x$  und  $y$  genau dann kongruent sind, wenn sie sich um ein Vielfaches von  $m$  unterscheiden:

$$x \equiv y \pmod{m} \Leftrightarrow x - y = k \cdot m \text{ für ein } k \in \mathbb{Z} \quad (2.5)$$

#### ■ Addition in $\mathbb{Z}_4$

+	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

#### ■ Multiplikation in $\mathbb{Z}_4$

.	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

#### ■ Addition in $\mathbb{Z}_5$

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

#### ■ Multiplikation in $\mathbb{Z}_5$

.	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Abb. 2.1: Addition und Multiplikation auf den Mengen  $\mathbb{Z}_4$  und  $\mathbb{Z}_5$

Haben Sie bemerkt, dass wir das Schlüsselwort ‚mod‘ in diesem Abschnitt auf zwei verschiedene Weisen verwenden haben? In den Gleichungen (2.3) und (2.4) beschreibt ‚mod‘ eine Funktion, die den ganzzahligen Rest berechnet, der bei der Division von  $x$  durch  $m$  entsteht. Diese Art der Verwendung verkörpert die funktionale Sichtweise auf die modulare Arithmetik. Sie ist insbesondere bei Informatikern verbreitet, da jeder Programmierer eine solche Modulo-Funktion aus der täglichen Arbeit kennt.

In Definition 2.1 wird das Schlüsselwort ‚mod‘ hingegen *relational* verwendet. Die relationale Sichtweise spiegelt das Denken in Äquivalenzklassen wider und wird von den meisten Mathematikern als die natürlichere empfunden.

Glücklicherweise können wir die beiden Sichtweisen auf einfache Weise miteinander verbinden. Es gilt:

$$x \equiv y \pmod{m} \Leftrightarrow x \bmod m = y \bmod m$$

Im Folgenden werden wir das Schlüsselwort ‚mod‘ auf beide Arten verwenden. Missverständnisse haben wir dabei nicht zu befürchten, denn aus dem Kontext wird immer zweifelsfrei hervorgehen, ob ‚mod‘ eine Funktion oder das Bestehen bzw. Nichtbestehen einer Relation ausdrückt.

Bei einem Blick in fremde Literatur gibt es dann aber doch eine Kleinigkeit zu beachten: Die Kongruenz zweier Zahlen  $x$  und  $y$  wird dort manchmal mit dem gewöhnlichen Gleichheitszeichen ausgedrückt und nicht, wie hier, mit dem Symbol  $\equiv$ . Der Ausdruck  $x+y \equiv 0 \pmod{m}$  liest sich dann folgendermaßen:

$$x+y = 0 \pmod{m}$$

Trotzdem geht auch hier aus der Art der Verwendung zweifelsfrei hervor, ob ‚mod‘ funktional oder relational verwendet wird.



Ein Blick auf die Verknüpfungstabellen in Abbildung 2.1 offenbart zwei interessante Eigenschaften:

- Betrachten wir die Addition in  $\mathbb{Z}_m$ , so fällt auf, dass jedes Element durch die Addition einer bestimmten anderen Zahl auf die 0 abgebildet werden kann. Mit anderen Worten: Für jedes vorgelegte Element  $x \in \mathbb{Z}_m$  hat die Gleichung

$$x+y \equiv 0 \pmod{m} \quad (2.6)$$

eine eindeutig bestimmte Lösung  $y \in \mathbb{Z}_m$ . Den Wert von  $y$  können wir auch sofort angeben. Für  $x=0$  ist es die Zahl  $y=0$  und für  $x>0$  die Zahl  $y=m-x$ .

- Betrachten wir die Multiplikation in  $\mathbb{Z}_m$  in Bezug auf die Lösbarkeit der Gleichung

$$x \cdot y \equiv 1 \pmod{m}, \quad (2.7)$$

so ist die Situation nicht mehr ganz so einfach. In  $\mathbb{Z}_5$  können wir diese Gleichung tatsächlich für jedes von 0 verschiedene Element lösen, in  $\mathbb{Z}_4$  dagegen nicht. Zum Beispiel gibt es in  $\mathbb{Z}_4$  keine Möglichkeit, die Zahl  $x=2$  durch die Multiplikation mit einer anderen Zahl in die 1 zu überführen. Als Ergebnis erhalten wir immer eine der beiden Zahlen 0 oder 2.

Wir wollen an dieser Stelle genauer überlegen, unter welchen Bedingungen Gleichung (2.7) lösbar ist und unter welchen nicht. Zunächst halten wir fest, dass die Kongruenz (2.7) nach dem oben Gesagten genau dann gilt, wenn eine Zahl  $k \in \mathbb{Z}$  mit

$$x \cdot y - 1 = k \cdot m \quad (2.8)$$

existiert. Abhängig davon, in welcher Beziehung die Zahlen  $x$  und  $m$  zueinander stehen, unterscheiden wir zwei Fälle:

- Fall 1:  $x$  und  $m$  sind nicht teilerfremd.

In diesem Fall können wir sowohl  $x$  als auch  $m$  als ein Produkt mit einem gemeinsamen Faktor  $p > 1$  darstellen. Damit lässt sich Gleichung (2.8) in die Form  $(p \cdot x') \cdot y - 1 = k \cdot (p \cdot m')$  bringen und weiter umformen zu

$$p \cdot (x' \cdot y - k \cdot m') = 1.$$

Das bedeutet, dass die Zahl  $p$  ein Faktor von 1 ist, was wegen  $p > 1$  nicht sein kann. Der Widerspruch zeigt, dass wir unsere Annahme, die Kongruenz (2.7) sei lösbar, verwerfen müssen.

■ Fall 2:  $x$  und  $m$  sind teilerfremd.

Wir zeigen zunächst für alle  $x \neq 0$ , dass für zwei verschiedene Zahlen  $y, z \in \mathbb{Z}_m$  auch die Produkte  $x \cdot y$  und  $x \cdot z$  verschieden sind. Wir beweisen dies durch Widerspruch und nehmen an, die Kongruenz  $x \cdot y \equiv x \cdot z \pmod{m}$  sei für zwei Zahlen  $y, z \in \mathbb{Z}_m$  mit  $y > z$  wahr. Daraus folgt, dass eine Zahl  $k \in \mathbb{Z}$  existiert, mit  $x \cdot y - x \cdot z = k \cdot m$ . Diese Gleichung können wir umschreiben zu  $x \cdot (y - z) = k \cdot m$ . Der Modul  $m$  tritt auf der rechten Seite als Faktor auf. Folgerichtig muss  $m$  auch ein Faktor der linken Seite sein und damit entweder  $x$  oder  $y - z$  teilen. Beides führt zu einem Widerspruch. Per Annahme sind  $x$  und  $m$  teilerfremd und der Wert von  $y - z$  liegt im Bereich zwischen 1 und  $m - 1$ .

Damit haben wir bewiesen, dass die Funktionen  $f_x : \mathbb{Z}_m \rightarrow \mathbb{Z}_m$  mit  $f_x(y) = x \cdot y$  für alle  $x \neq 0$  injektiv sind (Abbildung 2.2). Jede injektive Funktion, die eine endliche Menge auf sich selbst abbildet, muss aber zwangsläufig surjektiv sein (Abbildung 2.3). Das bedeutet wiederum, dass jedes Element  $\mathbb{Z}_m$  als Ergebnis einer Multiplikation auftauchen muss und damit insbesondere auch die 1. Kurzum: Die Kongruenz (2.7) hat für alle  $x \neq 0$  eine Lösung.

Damit haben wir einen Beweis für einen wichtigen Hilfssatz gefunden, der uns später, in Abschnitt 6.2, viel Arbeit ersparen wird:

### Satz 2.1

Für jedes Element  $x \in \mathbb{Z}_m$  gilt:

$$x \cdot y \equiv 1 \pmod{m} \text{ hat eine Lösung} \Leftrightarrow x, m \text{ sind teilerfremd}$$

Oder, was dasselbe ist:

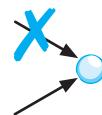
$$x \cdot y \equiv 1 \pmod{m} \text{ hat eine Lösung} \Leftrightarrow \text{ggT}(x, m) = 1$$

Im Falle einer Lösung ist  $y$  eindeutig bestimmt.

Die Abkürzung  $\text{ggT}(x, m)$  steht für den *größten gemeinsamen Teiler* von  $x$  und  $m$ , d. h. für die größte natürliche Zahl, die  $x$  und  $m$  teilt.

Satz 2.1 ebnet uns den Weg, um für ein Element  $x \in \mathbb{Z}_m$  zu entscheiden, ob in  $\mathbb{Z}_m$  ein multiplikatives Inverses vorhanden ist oder nicht. Hierzu müssen wir lediglich den größten gemeinsamen Teiler von  $x$  und  $m$  ausrechnen und das Ergebnis mit 1 vergleichen. Genau dann, wenn der Vergleich positiv ausfällt, existiert ein inverses Element.

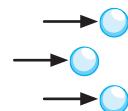
■ Injektive Funktionen



Jedes Element der Zielmenge besitzt höchstens ein Urbild.

  $f(x) = f(y) \Rightarrow x = y$

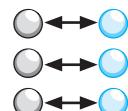
■ Surjektive Funktionen



Jedes Element der Zielmenge besitzt mindestens ein Urbild.

 Für alle  $y$  existiert ein  $x$  mit  $f(x) = y$

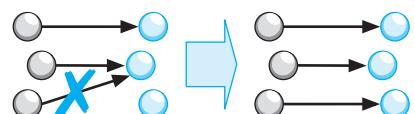
■ Bijektive Funktionen



Zwischen der Definitionsmenge und der Zielmenge besteht eine Eins-zu-eins-Beziehung.

  $f$  ist injektiv und surjektiv

**Abb. 2.2:** Injektive, surjektive und bijektive Funktionen

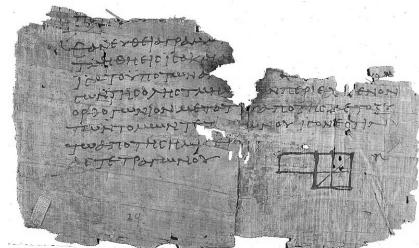


**Abb. 2.3:** Sind die Definitionsmenge und die Zielmenge endlich und gleichmächtig (beide Mengen enthalten gleich viele Elemente), so ist eine injektive Funktion immer auch surjektiv. Erst bei unendlichen Mengen geht dieser Zusammenhang verloren.



Euklid von Alexandria  
(ca. 365 v. Chr. – ca. 300 v. Chr.)

**Abb. 2.4:** Euklid von Alexandria gehörte zu den bedeutendsten und einflussreichsten Mathematikern des antiken Griechenlands. Sein wissenschaftliches Vermächtnis umfasst neben fundamentalen Erkenntnissen aus dem Bereich der Geometrie auch wichtige Ergebnisse aus der Zahlentheorie und der Algebra. Auch die axiomatische Methode, auf der weite Teile der modernen Mathematik beruhen, hat ihre Wurzeln in den Arbeiten von Euklid.



**Abb. 2.5:** Eines der ältesten bisher gefundenen Originalfragmente von Euklids historischem Werk *Die Elemente*

## Euklidischer Algorithmus

Für die Berechnung des größten gemeinsamen Teilers existiert ein altes Verfahren, das bereits im antiken Griechenland bekannt war. Beschrieben wurde es um 300 v. Chr. von Euklid von Alexandria, im 7. Buch seines epochalen Werks *Die Elemente* (Abbildungen 2.4 und 2.5). Konkret beschäftigte sich Euklid mit dem geometrischen Problem, für zwei Strecken  $AB$  und  $CD$  eine gemeinsame Maßzahl zu finden, sodass sich sowohl die Länge von  $AB$  als auch die Länge von  $CD$  als ganzzahliges Vielfaches dieser Maßzahl ausdrücken lässt. In der deutschen Ausgabe der *Elemente* sind seine Worte folgendermaßen übersetzt [92]:

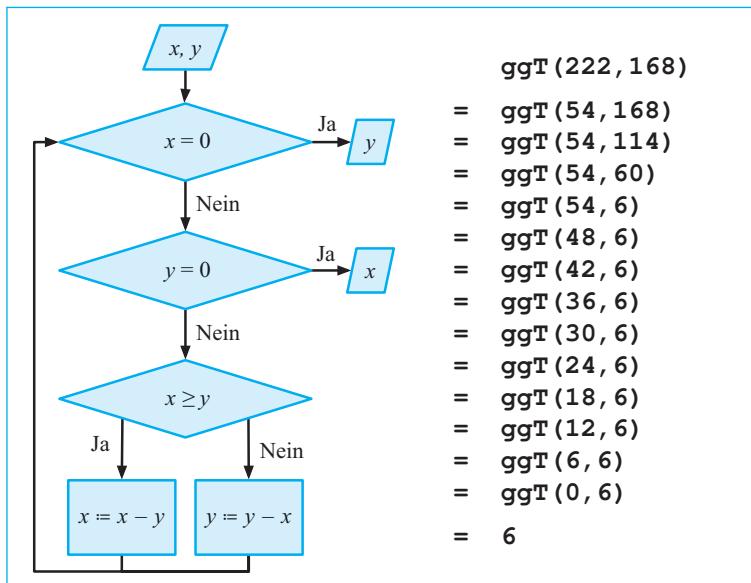
„Wenn  $CD$  aber  $AB$  nicht misst, und man nimmt bei  $AB$ ,  $CD$  abwechselnd immer das Kleinere vom Größeren weg, dann muss [schließlich] eine Zahl übrig bleiben, die die vorangehende misst.“

Euklid von Alexandria

Identifizieren wir  $x$  mit der Länge von  $AB$  und  $y$  mit der Länge von  $CD$ , so ist das berechnete Maß der größte gemeinsame Teiler von  $x$  und  $y$ , kurz  $\text{ggT}(x, y)$ . Die Berechnung, wie sie von Euklid vorgeschlagen wurde, folgt dem Schema der *Wechselwegnahme*. Ausgehend von zwei Zahlen  $x, y$  wird so lange die jeweils kleinere von der jeweils größeren abgezogen, bis eine der beiden gleich 0 ist. Die von 0 verschiedene Zahl ist dann das gesuchte Ergebnis. Abbildung 2.6 erläutert den Ablauf des *euklidischen Algorithmus* anhand eines Flussdiagramms und der exemplarischen Berechnung von  $\text{ggT}(222, 168)$ .

Dass der euklidische Algorithmus funktioniert und tatsächlich für alle Zahlenpaare den größten gemeinsamen Teiler liefert, hat zwei Gründe:

- Der größte gemeinsame Teiler von  $x$  und  $y$  ist gleichsam der größte gemeinsame Teiler von  $\min\{x, y\}$  und  $|x - y|$ . Durch das Prinzip der Wechselwegnahme wird die Berechnung von  $\text{ggT}(x, y)$  damit auf ein einfacheres Problem mit der gleichen Lösung reduziert.
- Da die Summe der beiden Zahlen in jedem Berechnungsschritt kleiner, aber niemals negativ wird, muss der Algorithmus terminieren. Nach endlich vielen Schritten wird einer der beiden Operanden den Wert 0 erreichen und hierdurch das Ergebnis offenlegen. Wegen  $\text{ggT}(0, y) = y$  und  $\text{ggT}(x, 0) = x$  ist der von 0 verschiedene Operand der gesuchte größte gemeinsame Teiler.



**Abb. 2.6:** Algorithmus zur Berechnung des größten gemeinsamen Teilers, abgeleitet aus der Arbeit von Euklid. Nach dem Prinzip der Wechselwegnahme wird die jeweils größere Zahl so lange um die kleinere reduziert, bis eine davon gleich 0 ist. In diesem Fall ist die von 0 verschiedene Zahl der größte gemeinsame Teiler.

Sicher ist Ihnen aufgefallen, dass wir in unserem Beispiel eine Vielzahl von Subtraktionen durchführen mussten, bis das Ergebnis endlich feststand. Dies nährt den Verdacht, dass der euklidische Algorithmus zwar effektiv ist, aber alles andere als effizient. Tatsächlich lässt sich der Algorithmus auf recht einfache Weise optimieren, indem die Subtraktion durch die Modulo-Operation ersetzt wird (Abbildung 2.7). Die Beispielrechnung zeigt, dass wir im Vergleich zu Euklids Originalalgorithmus jetzt wesentlich weniger Rechenschritte benötigen.

Das Ergebnis bleibt dennoch korrekt. Durch die Modulo-Operation werden lediglich mehrere Subtraktionsschritte zu einer einzelnen Operation zusammengefasst; der eigentliche Ablauf des Algorithmus bleibt hierdurch unberührt.

Die Verwendung der Modulo-Operation bringt einen weiteren Vorteil mit sich. Da der Divisionsrest stets kleiner ist als der Divisor, ist abwechselnd der eine oder der andere Operand der größere von beiden. Haben wir z. B. in der aktuellen Iteration den ersten durch den zweiten Operanden dividiert, so werden wir als Nächstes den zweiten durch den ersten dividieren. Damit können wir den Algorithmus noch effizienter implementieren, indem wir stets den ersten durch den zweiten Operanden teilen und die Werte beider Operanden anschließend vertauschen. Auf diese Weise können wir die beiden Vergleiche  $x = 0$  und  $x \geq y$  ersatzlos streichen und erhalten als Ergebnis das Ablaufdiagramm aus

**Abb. 2.7:** Der euklidische Algorithmus lässt sich beschleunigen, indem die Subtraktion durch die Modulo-Operation ausgetauscht wird. Hierdurch werden alle Subtraktionen, die nacheinander den gleichen Operanden reduzieren, zu einer einzigen Operation zusammengefasst. Die eingesparten Rechenschritte sind grau hinterlegt dargestellt.

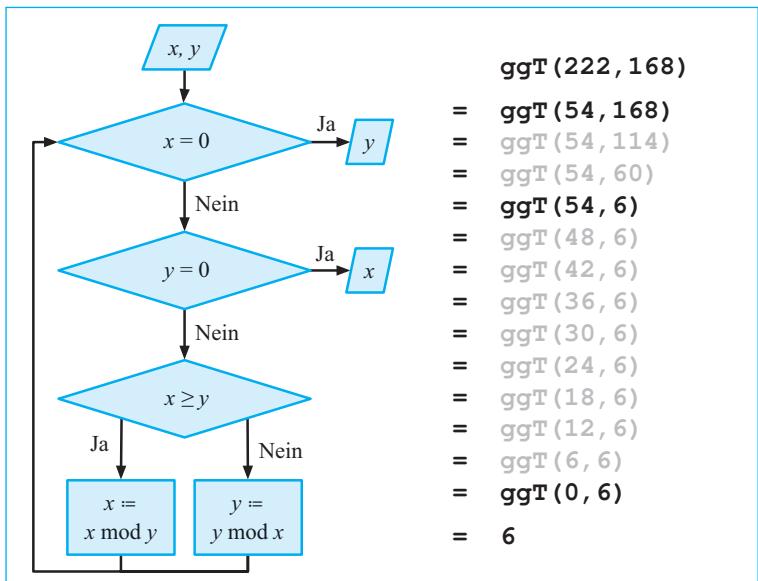
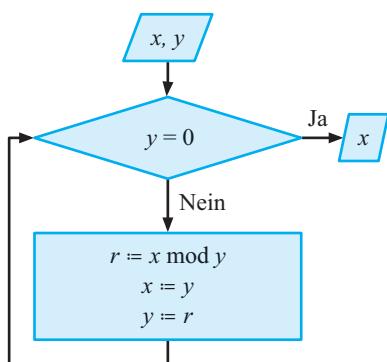


Abbildung 2.8. Dies ist die Form, in der die meisten Lehrbücher den euklidischen Algorithmus präsentieren.

Als Nächstes führen wir eine Menge ein, auf die wir später gleich mehrfach zurückgreifen werden. Sie ist eng mit der uns bekannten Menge  $\mathbb{Z}_m$  verwandt, enthält jedoch nur solche Elemente, die zu  $m$  teilerfremd sind:

$$\mathbb{Z}_m^* := \{x \in \mathbb{Z}_m \mid \text{ggT}(x, m) = 1\}$$

Nach Satz 2.1 umfasst die Menge  $\mathbb{Z}_m^*$  genau jene Elemente aus  $\mathbb{Z}_m$ , für die sich die Kongruenz (2.7) lösen lässt (Abbildung 2.9). Einen wichtigen Spezialfall erhalten wir, wenn der Modul  $m$  eine Primzahl ist. In diesem Fall ist jedes von 0 verschiedene Element  $x \in \mathbb{Z}_m$  wegen  $\text{ggT}(x, m) = 1$  auch ein Element von  $\mathbb{Z}_m^*$ :



### Satz 2.2

Für jede Primzahl  $p$  ist  $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\}$ .

**Abb. 2.8:** Moderne Formulierung des euklidischen Algorithmus

$\mathbb{Z}_m^*$  besitzt die interessante Eigenschaft, dass sie bezüglich der Multiplikation modulo  $m$  abgeschlossen ist, d.h., wir erhalten als Produkt zweier Zahlen aus  $\mathbb{Z}_m^*$  stets eine Zahl aus der Menge  $\mathbb{Z}_m^*$  zurück (Abbildung 2.10).

## 2.3 Algebraische Strukturen

In diesem Abschnitt besprechen wir mehrere algebraische Strukturen, die im Bereich der Informations- und Codierungstheorie eine hervorgehobene Rolle spielen. Ganz allgemein gesprochen verbirgt sich hinter einer algebraischen Struktur eine Menge, auf der eine oder mehrere Verknüpfungen mit bestimmten Eigenschaften definiert sind. Die Eigenschaften, die wir in den folgenden Abschnitten einfordern, werden auf den ersten Blick nicht immer einsichtig sein, und manchmal sogar willkürlich wirken. Doch seien Sie unbesorgt. Wir werden den abstrakten Begriffen durch die Angabe zahlreicher Beispiele ein Gesicht verleihen und dabei viele Mengen und Verknüpfungen entdecken, die Ihnen aus der Schulmathematik vertraut sind. Damit haben wir auch schon den Sinn hinter der Definition algebraischer Strukturen aufgedeckt. In ihrem Kern steckt die Idee, von den uns bekannten mathematischen Objekten zu abstrahieren und die dort geltenden Gesetzmäßigkeiten in die Breite zu tragen.

### 2.3.1 Gruppen

Der Begriff der *Gruppe* beschreibt eine mathematische Struktur, die uns in verschiedenen Ausprägungen immer wieder begegnen wird. Wir werden ihn zunächst anhand eines konkreten Beispiels einführen und die gewonnenen Erkenntnisse anschließend in einer formalen Definition verallgemeinern.

Die Gruppe, die wir als Beispiel betrachten, wird durch die Menge  $M = \{\spadesuit, \heartsuit, \diamondsuit, \clubsuit\}$  und die in Abbildung 2.11 definierte Verknüpfung „ $\circ$ “ gebildet. Ein flüchtiger Blick auf die Verknüpfungstabelle reicht aus, um eine erste wichtige Eigenschaft zu erkennen: „ $\circ$ “ ist *abgeschlossen*. Verknüpfen wir zwei Elemente aus  $M$  miteinander, so erhalten wir stets wieder ein Element aus  $M$  zurück.

Eine andere wichtige Eigenschaft von „ $\circ$ “ geht aus Abbildung 2.12 hervor. Die dort durchgeführte Rechnung zeigt, dass es offenbar keine Rolle spielt, in welcher Reihenfolge die Elemente aus  $M$  miteinander verknüpft werden. Kurzum: Die Verknüpfung „ $\circ$ “ ist *assoziativ*, d. h., für beliebige Elemente  $x, y, z \in M$  gilt die folgende Beziehung:

$$x \circ (y \circ z) = (x \circ y) \circ z$$

Eine Struktur mit den beiden herausgearbeiteten Eigenschaften wird in der Algebra als *Halbgruppe* bezeichnet.

$\mathbb{Z}_1^*$	=	{}
$\mathbb{Z}_2^*$	=	{1}
$\mathbb{Z}_3^*$	=	{1, 2}
$\mathbb{Z}_4^*$	=	{1, 3}
$\mathbb{Z}_5^*$	=	{1, 2, 3, 4}
$\mathbb{Z}_6^*$	=	{1, 5}
$\mathbb{Z}_7^*$	=	{1, 2, 3, 4, 5, 6}
$\mathbb{Z}_8^*$	=	{1, 3, 5, 7}
$\mathbb{Z}_9^*$	=	{1, 2, 4, 5, 7, 8}

Abb. 2.9: Die Mengen  $\mathbb{Z}_1^*$  bis  $\mathbb{Z}_9^*$

■ Addition in  $\mathbb{Z}_8^*$

+	1	3	5	7
1	2	4	6	0
3	4	6	0	2
5	6	0	2	4
7	0	2	4	6

■ Multiplikation in  $\mathbb{Z}_8^*$

.	1	3	5	7
1	1	3	5	7
3	3	1	7	5
5	5	7	1	3
7	7	5	3	1

Abb. 2.10: Die Menge  $\mathbb{Z}_m^*$  ist bezüglich der Multiplikation abgeschlossen. Im Fall der Addition gilt die Abgeschlossenheit nur dann, wenn der Modul  $m$  eine Primzahl ist. In diesem Beispiel ist  $m = 8$  und die Addition nicht abgeschlossen.

○	♠	♡	◊	♣
♠	♠	♡	◊	♣
♡	♡	◊	♣	♠
◊	◊	♣	♠	♡
♣	♣	♠	♡	◊

**Abb. 2.11:** Die Menge  $\{\spadesuit, \heartsuit, \diamondsuit, \clubsuit\}$  bildet zusammen mit der angegebenen Verknüpfung  $,\circ$  eine Gruppe.

Zusätzlich existiert in unserer Beispielmenge ein Element mit einer ganz besonderen Eigenschaft: Verknüpfen wir ein beliebiges Element  $x \in \{\spadesuit, \heartsuit, \diamondsuit, \clubsuit\}$  von links oder von rechts mit  $\spadesuit$ , so erhalten wir  $x$  unverändert zurück. Gibt es in einer Halbgruppe ein solches *neutrales Element*, so sprechen wir von einem *Monoid*.

In unserem Beispiel gilt aber noch mehr: Die Menge  $M$  ist so reichhaltig und die Verknüpfung  $,\circ$  so geschickt gewählt, dass die Gleichung

$$x \circ y = \spadesuit$$

für jedes  $x \in \{\spadesuit, \heartsuit, \diamondsuit, \clubsuit\}$  eine Lösung besitzt. Das bedeutet, dass wir von jedem beliebigen Element  $x$  aus das neutrale Element  $\spadesuit$  erreichen können, indem wir  $x$  mit einem passenden Element  $y$  verknüpfen.

Die vier Eigenschaften zusammen machen unsere Beispielmenge zu jener Struktur, die in der Algebra als *Gruppe* bezeichnet wird. Wir fassen zusammen:

$$\begin{aligned} \spadesuit \circ (\spadesuit \circ \spadesuit) &= (\spadesuit \circ \spadesuit) \circ \spadesuit = \spadesuit \\ \spadesuit \circ (\spadesuit \circ \heartsuit) &= (\spadesuit \circ \spadesuit) \circ \heartsuit = \heartsuit \\ \spadesuit \circ (\spadesuit \circ \diamondsuit) &= (\spadesuit \circ \spadesuit) \circ \diamondsuit = \diamondsuit \\ \spadesuit \circ (\spadesuit \circ \clubsuit) &= (\spadesuit \circ \spadesuit) \circ \clubsuit = \clubsuit \\ \spadesuit \circ (\heartsuit \circ \spadesuit) &= (\heartsuit \circ \spadesuit) \circ \spadesuit = \heartsuit \\ \spadesuit \circ (\heartsuit \circ \heartsuit) &= (\heartsuit \circ \heartsuit) \circ \heartsuit = \diamondsuit \\ \spadesuit \circ (\heartsuit \circ \diamondsuit) &= (\heartsuit \circ \heartsuit) \circ \diamondsuit = \clubsuit \\ \spadesuit \circ (\heartsuit \circ \clubsuit) &= (\heartsuit \circ \heartsuit) \circ \clubsuit = \spadesuit \\ \spadesuit \circ (\diamondsuit \circ \spadesuit) &= (\diamondsuit \circ \spadesuit) \circ \spadesuit = \diamondsuit \\ \spadesuit \circ (\diamondsuit \circ \heartsuit) &= (\diamondsuit \circ \spadesuit) \circ \heartsuit = \clubsuit \\ \spadesuit \circ (\diamondsuit \circ \diamondsuit) &= (\diamondsuit \circ \spadesuit) \circ \diamondsuit = \spadesuit \\ \spadesuit \circ (\diamondsuit \circ \clubsuit) &= (\diamondsuit \circ \spadesuit) \circ \clubsuit = \heartsuit \\ &\dots \\ \clubsuit \circ (\clubsuit \circ \spadesuit) &= (\clubsuit \circ \spadesuit) \circ \spadesuit = \diamondsuit \\ \clubsuit \circ (\clubsuit \circ \heartsuit) &= (\clubsuit \circ \spadesuit) \circ \heartsuit = \clubsuit \\ \clubsuit \circ (\clubsuit \circ \diamondsuit) &= (\clubsuit \circ \spadesuit) \circ \diamondsuit = \spadesuit \\ \clubsuit \circ (\clubsuit \circ \clubsuit) &= (\clubsuit \circ \spadesuit) \circ \clubsuit = \heartsuit \end{aligned}$$

**Abb. 2.12:** Die Verknüpfung  $,\circ$  ist assoziativ. Es spielt keine Rolle, in welcher Reihenfolge wir die Elemente miteinander verknüpfen.



## Definition 2.2 (Halbgruppe, Gruppe)

Eine Menge  $M$  und eine Operation  $,\circ$ , geschrieben als  $(M, \circ)$ , bilden eine *Halbgruppe*, wenn die folgenden Eigenschaften gelten:

■ Abgeschlossenheit

$x \circ y \in M$  für alle  $x, y \in M$

■ Assoziativität

$(x \circ y) \circ z = x \circ (y \circ z)$  für alle  $x, y, z \in M$

Eine Halbgruppe  $(M, \circ)$  ist eine *Gruppe*, wenn zusätzlich gilt:

■ Existenz eines neutralen Elements

Es gibt ein  $e \in M$ , sodass für alle  $x \in M$  gilt:  $x \circ e = x$

■ Existenz inverser Elemente

Für alle  $x \in M$  existiert ein Element  $x^{-1} \in M$  mit  $x \circ x^{-1} = e$

Ein zweiter Blick auf die Verknüpfungstabelle in Abbildung 2.11 zeigt, dass unsere Beispielgruppe noch eine weitere elementare Eigenschaft erfüllt. Verknüpfen wir zwei beliebige Elemente  $x$  und  $y$  miteinander, so ist es unerheblich, ob wir  $x$  als linken und  $y$  als rechten Operanden verwenden oder umgekehrt. Mit anderen Worten: Die Verknüpfung

, $\circ$ ' ist *kommutativ*. Gruppen mit dieser Eigenschaft heißen *kommutative Gruppen* oder *Abel'sche Gruppen*, benannt nach dem norwegischen Mathematiker Niels Henrik Abel (Abbildung 2.13).

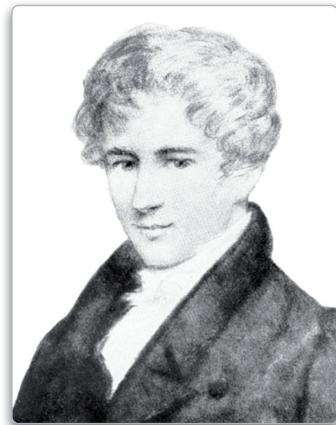


### Definition 2.3 (Kommutative Gruppe, Abel'sche Gruppe)

Eine Gruppe  $(M, \circ)$  heißt *kommutative Gruppe* oder *Abel'sche Gruppe*, wenn sie zusätzlich die folgende Eigenschaft erfüllt:

- Kommutativität

$$\text{☞ } x \circ y = y \circ x \quad \text{für alle } x, y \in M$$



Niels Henrik Abel  
(1802 – 1829)

Gruppen, die für ', $\circ$ ' das Symbol ',+' verwenden, werden gerne als *additive Gruppen* bezeichnet. Analog hierzu wird von einer *multiplikativen Gruppe* gesprochen, wenn das Symbol ',.' benutzt wird. Bei additiven Gruppen wird das neutrale Element in der Regel mit 0 und das inverse Element von  $x$  mit  $-x$  bezeichnet. In multiplikativen Gruppen wird für das neutrale Element zumeist die 1 verwendet und für die Bezeichnung der inversen Elemente entweder die Schreibweise  $x^{-1}$  beibehalten oder die Symbolik  $\frac{1}{x}$  verwendet. Behalten Sie in diesem Zusammenhang stets im Gedächtnis, dass ',+' und ',.' lediglich als Symbole verwendet werden; sie können für beliebige Verknüpfungen stehen und müssen nicht zwangsläufig der wohlvertrauten Addition oder Multiplikation entsprechen. Das Gleiche gilt für die Symbole 0 und 1.

Ist  $(M, \circ)$  eine Gruppe und geht aus dem Kontext hervor, welche Verknüpfung mit ', $\circ$ ' gemeint ist, so werden wir manchmal auch  $M$  als Gruppe bezeichnen und auf die Nennung der Verknüpfung verzichten. Das Gleiche gilt für die algebraischen Strukturen, die wir später behandeln. Schauen wir nun ein paar weitere Gruppen an:

- Additive Gruppe  $(\mathbb{Z}, +)$

Die Menge der ganzen Zahlen, zusammen mit der gewöhnlichen Addition, ist eine kommutative Gruppe. Die Verknüpfung ist assoziativ und kommutativ und führt aus der Menge  $\mathbb{Z}$  nicht hinaus. Das neutrale Element ist die 0, und jedes Element  $x$  lässt sich durch die Addition von  $-x$  auf das neutrale Element zurückführen.

- Additive Gruppe  $(\mathbb{Z}_m, +)$

Für jede positive Zahl  $m \in \mathbb{N}^+$  ist  $(\mathbb{Z}_m, +)$  eine kommutative Gruppe (Abbildung 2.14 oben). Die modulare Arithmetik sorgt dafür, dass die Verknüpfung auf der Menge  $\mathbb{Z}_m$  abgeschlossen ist. Ferner ändert

**Abb. 2.13:** Mit seinen Untersuchungen zur Lösung von Polynomgleichungen höherer Grade hat der norwegische Mathematiker Niels Henrik Abel maßgebliche Impulse für die Entwicklung der Gruppentheorie geliefert. Ihm zu Ehren werden kommutative Gruppen heute auch als Abel'sche Gruppen bezeichnet.

Dem brillanten Mathematiker war kein langes Leben vergönnt. Abel starb an den Folgen der Tuberkulose im jungen Alter von nur 26 Jahren.

Additive Gruppen ( $\mathbb{Z}_m, +$ )						
■ ( $\mathbb{Z}_2, +$ )	■ ( $\mathbb{Z}_3, +$ )	■ ( $\mathbb{Z}_4, +$ )	■ ( $\mathbb{Z}_5, +$ )			
$\begin{array}{ c c c } \hline + & 0 & 1 \\ \hline 0 & 0 & 1 \\ \hline 1 & 1 & 0 \\ \hline \end{array}$	$\begin{array}{ c c c c } \hline + & 0 & 1 & 2 \\ \hline 0 & 0 & 1 & 2 \\ \hline 1 & 1 & 2 & 0 \\ \hline 2 & 2 & 0 & 1 \\ \hline \end{array}$	$\begin{array}{ c c c c c } \hline + & 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 1 & 2 & 3 \\ \hline 1 & 1 & 2 & 3 & 0 \\ \hline 2 & 2 & 3 & 0 & 1 \\ \hline 3 & 3 & 0 & 1 & 2 \\ \hline \end{array}$	$\begin{array}{ c c c c c c } \hline + & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 0 & 1 & 2 & 3 & 4 \\ \hline 1 & 1 & 2 & 3 & 4 & 0 \\ \hline 2 & 2 & 3 & 4 & 0 & 1 \\ \hline 3 & 3 & 4 & 0 & 1 & 2 \\ \hline 4 & 4 & 0 & 1 & 2 & 3 \\ \hline \end{array}$			
Multiplikative Gruppen ( $\mathbb{Z}_p \setminus \{0\}, \cdot$ )						
■ ( $\mathbb{Z}_2 \setminus \{0\}, \cdot$ )	■ ( $\mathbb{Z}_3 \setminus \{0\}, \cdot$ )	■ ( $\mathbb{Z}_5 \setminus \{0\}, \cdot$ )	■ ( $\mathbb{Z}_7 \setminus \{0\}, \cdot$ )			
$\begin{array}{ c c } \hline \cdot & 1 \\ \hline 1 & 1 \\ \hline \end{array}$	$\begin{array}{ c c c } \hline \cdot & 1 & 2 \\ \hline 1 & 1 & 2 \\ \hline 2 & 2 & 1 \\ \hline \end{array}$	$\begin{array}{ c c c c c } \hline \cdot & 1 & 2 & 3 & 4 \\ \hline 1 & 1 & 2 & 3 & 4 \\ \hline 2 & 2 & 4 & 1 & 3 \\ \hline 3 & 3 & 1 & 4 & 2 \\ \hline 4 & 4 & 3 & 2 & 1 \\ \hline \end{array}$	$\begin{array}{ c c c c c c c } \hline \cdot & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 2 & 2 & 4 & 6 & 1 & 3 & 5 \\ \hline 3 & 3 & 6 & 2 & 5 & 1 & 4 \\ \hline 4 & 4 & 1 & 5 & 2 & 6 & 3 \\ \hline 5 & 5 & 3 & 1 & 6 & 4 & 2 \\ \hline 6 & 6 & 5 & 4 & 3 & 2 & 1 \\ \hline \end{array}$			

Abb. 2.14: Beispiele der additiven Gruppen ( $\mathbb{Z}_m, +$ ) (oben) und der multiplikativen Gruppen ( $\mathbb{Z}_p \setminus \{0\}, \cdot$ ) (unten)

die Modulo-Berechnung nichts an der Assoziativität und Kommutativität der Addition, und das neutrale Element 0 ist in  $\mathbb{Z}_m$  ebenfalls enthalten. Darüber hinaus existiert für jedes Element  $x \in \mathbb{Z}_m$  ein inverses Element. Die 0 ist zu sich selbst invers und jedes Element  $x > 0$  lässt sich durch die Addition von  $m - x$  auf die 0 zurückführen. Damit sind alle Eigenschaften einer kommutativen Gruppe erfüllt.

Eine für unsere Zwecke besonders wichtige Gruppe dieses Typs erhalten wir für den Fall  $m = 2$ .  $\mathbb{Z}_m$  enthält dann nur noch die Elemente 0 und 1, und wir können die Addition jetzt sogar auf der logischen Ebene interpretieren. Die Summe  $x+y$  entspricht der XOR-Verknüpfung, wenn wir die Elemente 1 und 0 als die logischen Wahrheitswerte *Wahr* bzw. *Falsch* auffassen.

### ■ Multiplikative Gruppe $(\mathbb{Q} \setminus \{0\}, \cdot)$

Die Menge der von 0 verschiedenen rationalen Zahlen  $\mathbb{Q}$  – dies sind die Brüche  $\frac{p}{q}$  mit  $p \neq 0$  und  $q \neq 0$  – bildet mit der gewöhnlichen Multiplikation eine kommutative Gruppe. Die Verknüpfung ist sowohl assoziativ als auch kommutativ und führt aus der Menge  $\mathbb{Q} \setminus \{0\}$  nicht hinaus. Das neutrale Element ist die 1 und durch die Multiplikation von  $\frac{1}{x}$  können wir aus jeder Zahl  $x$  das neutrale Element zurückgewinnen.

### ■ Multiplikative Gruppe $(\mathbb{Z}_p \setminus \{0\}, \cdot)$

Ist  $p$  eine Primzahl, dann bildet die Menge  $\mathbb{Z}_p \setminus \{0\}$ , zusammen mit der Multiplikation modulo  $p$ , eine kommutative Gruppe. Abbildung 2.14 (unten) zeigt die entsprechenden Verknüpfungstabellen für die Primzahlen von 2 bis 7. An den Tabellen lassen sich zwei wesentliche Eigenschaften ablesen: Zum einen führt die Multiplikation zweier Zahlen aus  $\mathbb{Z}_p \setminus \{0\}$  nicht aus der Menge  $\mathbb{Z}_p \setminus \{0\}$  heraus; insbesondere erhalten wir als Ergebnis niemals den Wert 0. Zum anderen gibt es zu jedem  $x$  ein inverses Element  $y$ , sodass  $x \cdot y$  das neutrale Element 1 ergibt. Dass wir ein solches  $y$  immer finden können, wenn  $p$  eine Primzahl ist, garantiert uns Satz 2.1. Aus ihm folgt weiterhin, dass die Gruppeneigenschaft verloren geht, wenn wir für  $p$  eine faktorisierbare Zahl wählen. In diesem Fall besitzen genau jene Elemente keine multiplikativen Inversen, die einen gemeinsamen Faktor mit  $p$  teilen.

### ■ Multiplikative Gruppe $(\mathbb{Z}_m^*, \cdot)$

Wir haben gerade gesehen, dass die Menge  $\mathbb{Z}_p \setminus \{0\}$  genau dann eine Gruppe bildet, wenn  $p$  eine Primzahl ist; nur so ist gewährleistet, dass zu jedem Element auch ein multiplikatives Inverses existiert. Sei nun  $m$  eine beliebige natürliche Zahl  $\geq 2$ . Entfernen wir sämtliche Elemente aus der Menge  $\mathbb{Z}_m$ , die kein inverses Element besitzen, so müsste es möglich sein, wieder zu einer Gruppe zu gelangen. Die Menge, die wir als Ergebnis erhalten, kennen wir schon: Es ist die Menge  $\mathbb{Z}_m^*$  aus Abschnitt 2.2. Dass  $\mathbb{Z}_m^*$  tatsächlich eine Gruppe bildet, lässt sich leicht überprüfen. Die Multiplikation ist auf dieser Menge abgeschlossen und das neutrale Element (die Zahl 1) ist ebenfalls in ihr enthalten. Ferner stellt die Definition von  $\mathbb{Z}_m^*$  sicher, dass für jedes Element ein multiplikatives Inverses existiert.

Abbildung 2.15 zeigt die Verknüpfungstabellen der multiplikativen Gruppen  $\mathbb{Z}_7^*$ ,  $\mathbb{Z}_8^*$  und  $\mathbb{Z}_9^*$ . Die Tabelle von  $\mathbb{Z}_7^*$  kennen wir bereits aus Abbildung 2.14. Sie gleicht eins zu eins der Verknüpfungstabelle von  $\mathbb{Z}_7 \setminus \{0\}$ , und wegen Satz 2.2 ist dies auch kein Wunder. Für jede Primzahl  $p$  sind die Mengen  $\mathbb{Z}_p \setminus \{0\}$  und  $\mathbb{Z}_p^*$  identisch, und damit entpuppen sich die Gruppen  $\mathbb{Z}_p$  als Spezialfälle der Gruppen  $\mathbb{Z}_m^*$ .

### • $(\mathbb{Z}_7^*, \cdot)$

.	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	1	3	5
3	3	6	2	5	1	4
4	4	1	5	2	6	3
5	5	3	1	6	4	2
6	6	5	4	3	2	1

$$\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$$

### • $(\mathbb{Z}_8^*, \cdot)$

.	1	3	5	7
1	1	3	5	7
3	3	1	7	5
5	5	7	1	3
7	7	5	3	1

$$\mathbb{Z}_8^* = \{1, 3, 5, 7\}$$

### • $(\mathbb{Z}_9^*, \cdot)$

.	1	2	4	5	7	8
1	1	2	4	5	7	8
2	2	4	8	1	5	7
4	4	8	7	2	1	5
5	5	1	2	7	8	4
7	7	5	1	8	4	2
8	8	7	5	4	2	1

$$\mathbb{Z}_9^* = \{1, 2, 4, 5, 7, 8\}$$

Abb. 2.15: Die multiplikativen Gruppen  $\mathbb{Z}_7^*$ ,  $\mathbb{Z}_8^*$  und  $\mathbb{Z}_9^*$

**Abb. 2.16:** Die Verknüpfungstabelle der symmetrischen Gruppe  $S_3$ . Die Grundmenge besteht aus allen Permutationen, die sich auf einer Menge mit drei Elementen bilden lassen, hier auf der Menge  $\{1, 2, 3\}$ . Die Gruppenoperation „ $\circ$ “ ist die Komposition (Hintereinanderausführung) zweier Permutationen.  $S_3$  erfüllt alle Gruppeneigenschaften, die Operation „ $\circ$ “ ist aber nicht kommutativ. Damit haben wir ein Beispiel einer Gruppe gefunden, die keine Abel'sche Gruppe ist.

$\circ$	(1, 2, 3)	(2, 3, 1)	(3, 1, 2)	(1, 3, 2)	(3, 2, 1)	(2, 1, 3)
(1, 2, 3)	(1, 2, 3)	(2, 3, 1)	(3, 1, 2)	(1, 3, 2)	(3, 2, 1)	(2, 1, 3)
(2, 3, 1)	(2, 3, 1)	(3, 1, 2)	(1, 2, 3)	(2, 1, 3)	(1, 3, 2)	(3, 2, 1)
(3, 1, 2)	(3, 1, 2)	(1, 2, 3)	(2, 3, 1)	(3, 2, 1)	(2, 1, 3)	(1, 3, 2)
(1, 3, 2)	(1, 3, 2)	(3, 2, 1)	(2, 1, 3)	(1, 2, 3)	(2, 3, 1)	(3, 1, 2)
(3, 2, 1)	(3, 2, 1)	(2, 1, 3)	(1, 3, 2)	(3, 1, 2)	(1, 2, 3)	(2, 3, 1)
(2, 1, 3)	(2, 1, 3)	(1, 3, 2)	(3, 2, 1)	(2, 3, 1)	(3, 1, 2)	(1, 2, 3)

### ■ Symmetrische Gruppe $S_n$

In der *symmetrischen Gruppe*  $S_n = (M, \circ)$  ist  $M$  die Menge aller Permutationen (Vertauschungen) einer  $n$ -elementigen Menge und „ $\circ$ “ die Komposition, d. h. die Hintereinanderausführung zweier Permutationen. Beachten Sie die Reihenfolge:  $x \circ y$  bedeutet, dass  $x$  ausgeführt wird, nachdem  $y$  ausgeführt wurde. Der Parameter  $n$  ist der *Grad* der Gruppe.

Formal können wir eine Permutation  $\pi$  als eine bijektive Abbildung auf der Menge  $\{1, \dots, n\}$  ansehen und den Funktionswert  $\pi(i)$  gedanklich so interpretieren, dass das Element an Position  $i$  auf die Position  $\pi(i)$  verschoben wird. Für die Angabe einer konkreten Permutation nutzen wir die folgende Schreibweise:

$$\pi = (\pi(1), \pi(2), \dots, \pi(n))$$

Aufgrund der Bijektivität können wir jede Umsortierung von Elementen durch eine inverse Permutation  $\pi^{-1}$  rückgängig machen. Bezeichnen wir mit  $\text{id}$  die *identische Abbildung*, die alle Elemente in ihrer ursprünglichen Form belässt, so gilt die Beziehung  $\pi \circ \pi^{-1} = \text{id}$ .

Als Beispiel ist in Abbildung 2.16 die symmetrische Gruppe  $S_3$  zu sehen. Sie umfasst 6 Permutationen, da sich die Elemente einer dreielementigen Menge auf genau 6 verschiedene Möglichkeiten vertauschen lassen.

Wir können uns leicht davon überzeugen, dass  $S_3$  alle Gruppeneigenschaften erfüllt. Zunächst ist klar, dass „ $\circ$ “ eine abgeschlossene Verknüpfung ist, da die Hintereinanderausführung zweier Permutationen wieder eine Permutation ergibt. Des Weiteren spielt es keine Rolle, ob wir drei Permutationen  $\pi_1, \pi_2, \pi_3$  in der Reihenfolge  $\pi_1 \circ (\pi_2 \circ \pi_3)$  oder  $(\pi_1 \circ \pi_2) \circ \pi_3$  ausführen, d. h., die Komposition ist assoziativ. Ferner existiert mit  $\text{id} = (1, 2, 3)$  ein neutrales Ele-

ment, und jede Vertauschung lässt sich durch eine inverse Permutation rückgängig machen. Damit sind sämtliche Gruppeneigenschaften erfüllt.

Beachten Sie, dass die Symmetriegruppe  $S_3$  nicht kommutativ ist. Beispielsweise liefert  $(3,2,1) \circ (2,3,1)$  das Ergebnis  $(2,1,3)$  und  $(2,3,1) \circ (3,2,1)$  das Ergebnis  $(1,3,2)$ .

Vielleicht ist Ihnen aufgefallen, dass in allen Beispielen Folgendes galt: Die inversen Elemente waren nicht nur rechtsinvers, wie in Definition 2.2 gefordert, sondern auch linksinvers und zudem eindeutig bestimmt. Das Gleiche galt für das neutrale Element. Der folgende Satz beweist, dass dies kein Zufall war:

### Satz 2.3

In jeder Gruppe  $(M, \circ)$  gilt:

- a) Die inversen Elemente sind auch *linksinvers*.   $x^{-1} \circ x = e$
- b) Das neutrale Element ist auch *linksneutral*.   $e \circ x = x$
- c) Die inversen Elemente und das neutrale Element sind eindeutig bestimmt.

Beweis:

a)

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

b)

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

c) Gäbe es zwei neutrale Elemente  $e_1$  und  $e_2$  mit  $e_1 \neq e_2$ , so wäre  $e_1 = e_2 \circ e_1$ , da  $e_2$  auch linksneutral ist. Daraus folgt  $e_1 = e_2$ , im Widerspruch zur Annahme.

Gäbe es für ein  $x$  zwei inverse Elemente  $y$  und  $z$  mit  $y \neq z$ , so wäre

$$y = e \circ y = (z \circ x) \circ y = z \circ (x \circ y) = z \circ e = z,$$

im Widerspruch zu  $y \neq z$ . □



Permutationen werden in der Literatur sehr unterschiedlich notiert. Manchmal wird eine zweizeilige Schreibweise verwendet, in der die Elemente der Grundmenge oben und die Funktionswerte unten angeordnet sind. Beispielsweise bedeutet

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 1 & 4 \end{pmatrix},$$

dass die 1 auf die 2, die 2 auf die 3, die 3 auf die 1 und die 4 auf sich selbst abgebildet wird. Unsere Notation ist eine Kurzform dieser Schreibweise, bei der nur die untere Zeile beibehalten wird:

$$\pi = (2, 3, 1, 4)$$

Eine alternative Notation basiert auf der Eigenschaft von Permutationen, Zyklen zu bilden. Führen wir z. B.  $\pi$  immer wieder aus, so wird die 1 zunächst auf die 2, dann auf die 3 und schließlich wieder auf die 1 abgebildet. Für die 2 und die 3 erhalten wir das gleiche Ergebnis und sagen, die Menge  $\{1, 2, 3\}$  bildet einen Zyklus, geschrieben als  $(1 \ 2 \ 3)$ . Die 4 bildet ihren eigenen Zyklus, da sie immer wieder auf sich selbst abgebildet wird. Damit können wir unsere Beispielpermutationen auch so notieren:

$$\pi = (1 \ 2 \ 3)(4)$$

Dies ist die *Zyk lenschreibweise* für Permutationen. Auf die Angabe von Zyklen der Länge 1 wird dabei gerne verzichtet, da die Eindeutigkeit der Darstellung hierdurch nicht verloren geht. Nach dieser Konvention wird  $\pi$  dann ganz einfach so aufgeschrieben:

$$\pi = (1 \ 2 \ 3)$$

## Untergruppen

Für viele Gruppen  $(M, \circ)$  gilt, dass sich die Menge  $M$  durch eine ihrer Teilmengen ersetzen lässt, ohne die Gruppeneigenschaften zu verlieren. Abbildung 2.17 demonstriert dies am Beispiel der symmetrischen Gruppe. Entnehmen wir aus  $S_3$  die Teilmenge

$$M' := \{(1,2,3), (2,3,1), (3,1,2)\},$$

so erhalten wir eine neue Gruppe mit nur drei Elementen. Die Komposition „ $\circ$ “ ist auf  $M'$  immer noch abgeschlossen, sodass wir tatsächlich eine Gruppe vor uns haben; in diesem Fall sogar eine kommutative. Gruppen, die wie hier, in eine größere Gruppe eingebettet sind, werden als Untergruppen bezeichnet:



### Definition 2.4 (Untergruppe)

Mit  $(M, \circ)$  und  $(M', \circ)$  seien zwei Gruppen gegeben.

- Ist  $M' \subseteq M$ , so nennen wir

$M'$  eine *Untergruppe* von  $M$  und

$M$  eine *Obergruppe* von  $M'$ .

- Ist  $M' \subset M$ , so nennen wir

$M'$  eine *echte Untergruppe* von  $M$  und

$M$  eine *echte Obergruppe* von  $M'$ .

Beachten Sie in dieser Definition, dass in  $(M, \circ)$  und  $(M', \circ)$  das gleiche Verknüpfungszeichen „ $\circ$ “ genannt ist. Das bedeutet, dass wir für zwei Elemente  $x, y \in M'$  immer das gleiche Ergebnis  $x \circ y$  erhalten, egal ob wir in der Verknüpfungstabelle von  $M$  oder in der Verknüpfungstabelle von  $M'$  nachschlagen. Eine Untergruppe unterscheidet sich von ihren Obergruppen also lediglich dadurch, dass sie potenziell weniger Elemente umfasst; die Verknüpfung bleibt dieselbe.

Die Untergruppeneigenschaft lässt sich anhand des *Untergruppenkriteriums* überprüfen. Dieses besagt, dass  $M'$  genau dann eine Untergruppe von  $M$  ist, wenn

- die Menge  $M'$  mindestens ein Element enthält,
- $M' \neq \emptyset$

$\circ$	(1,2,3)	(2,3,1)	(3,1,2)	(1,3,2)	(3,2,1)	(2,1,3)
(1,2,3)	(1,2,3)	(2,3,1)	(3,1,2)	(1,3,2)	(3,2,1)	(2,1,3)
(2,3,1)	(2,3,1)	(3,1,2)	(1,2,3)	(2,1,3)	(1,3,2)	(3,2,1)
(3,1,2)	(3,1,2)	(1,2,3)	(2,3,1)	(3,2,1)	(2,1,3)	(1,3,2)
(1,3,2)	(1,3,2)	(3,2,1)	(2,1,3)	(1,2,3)	(2,3,1)	(3,1,2)
(3,2,1)	(3,2,1)	(2,1,3)	(1,3,2)	(3,1,2)	(1,2,3)	(2,3,1)
(2,1,3)	(2,1,3)	(1,3,2)	(3,2,1)	(2,3,1)	(3,1,2)	(1,2,3)

**Abb. 2.17:** Die drei Permutationen  $(1,2,3)$ ,  $(2,3,1)$  und  $(3,1,2)$  bilden eine Untergruppe von  $S_3$ . Die Komposition führt aus dieser Menge nicht heraus und auch die anderen Gruppeneigenschaften sind weiterhin gültig. Da einige Permutationen in der betrachteten Teilmenge fehlen, sprechen wir nicht mehr länger von einer *symmetrischen Gruppe*, sondern nur noch ganz allgemein von einer *Permutationsgruppe*.

- die Verknüpfung mit inversen Elementen nicht aus  $M'$  hinausführt.

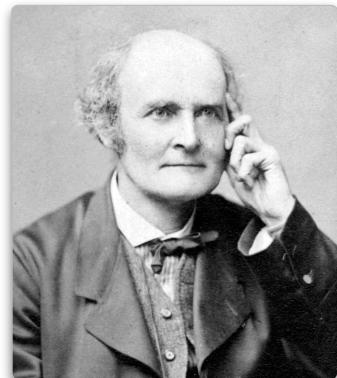
$$\text{☞ } x, y \in M' \Rightarrow x \circ y^{-1} \in M'$$

Der Begriff der Untergruppe ist ein bedeutender, denn es lässt sich zeigen, dass jede Gruppe zu einer Untergruppe einer symmetrischen Gruppe isomorph ist, sich also nur durch die Benennung der Elemente unterscheidet. Dies ist der Inhalt des berühmten *Satzes von Cayley*, der in seinen Grundzügen das erste Mal in einer Arbeit von Arthur Cayley aus dem Jahr 1854 bewiesen wurde (Abbildung 2.18) [16].

### 2.3.2 Körper

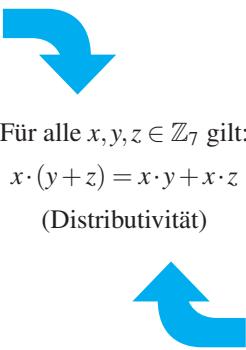
Anders als bei einer Gruppe, die aus einer Menge  $M$  und einer einzigen Verknüpfung „ $\circ$ “ gebildet wird, sind bei einem *Körper* immer zwei Verknüpfungen beteiligt. Um diese algebraische Struktur im Kern zu verstehen, werfen wir zunächst einen Blick auf Abbildung 2.19. Zu sehen sind die Tabellen zweier Verknüpfungen auf der Menge  $\mathbb{Z}_7$ , links die Addition modulo 7 und rechts die Multiplikation modulo 7. Nach unserem Ausflug in die Gruppentheorie ist die linke Tabelle für uns keine unbekannte mehr; im letzten Abschnitt haben wir herausgearbeitet, dass  $(\mathbb{Z}_7, +)$  eine kommutative Gruppe ist. Aber auch auf der rechten Seite können wir eine Gruppe identifizieren. Entfernen wir dort mit der 0 das neutrale Element von  $(\mathbb{Z}_7, +)$ , so erhalten wir die multiplikative Gruppe  $(\mathbb{Z}_7 \setminus \{0\}, \cdot)$ , die wir ebenfalls schon kennen.

Wichtig ist für uns die Tatsache, dass die beiden Gruppen nicht unabhängig voneinander sind. Die beiden Verknüpfungen „ $+$ “ und „ $\cdot$ “ sind über eine Eigenschaft verwoben, die uns aus Schulzeiten wohlvertraut



Arthur Cayley (1821 – 1895)

**Abb. 2.18:** Dem englischen Mathematiker Arthur Cayley haben wir wichtige Beiträge zur modernen Algebra zu verdanken. Auch der Begriff der Gruppe geht auf Cayley zurück, zumindest in seiner modernen Form [16]. Er war der Erste, der ihn in jenem abstrakten Sinne verwendete, den Sie in diesem Buch kennen gelernt haben: zur Beschreibung einer Menge und einer Verknüpfung, die zusammen die vier in Definition 2.2 geforderten Eigenschaften erfüllen.



+	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

Gruppe  $(\mathbb{Z}_7, +)$

·	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

Gruppe  $(\mathbb{Z}_7 \setminus \{0\}, \cdot)$

**Abb. 2.19:** Die Menge  $\mathbb{Z}_7$  bildet nicht nur mit  $,+^{\prime}$  eine kommutative Gruppe, sondern auch mit  $,\cdot^{\prime}$ , wenn wir das neutrale Element der ersten Gruppe herausnehmen. Die zusätzliche Gültigkeit des Distributivgesetzes macht  $(\mathbb{Z}_7, +, \cdot)$  zu einem *Körper*.

ist. Es gilt das *Distributivgesetz*  $x \cdot (y+z) = x \cdot y + x \cdot z$ , das wir zum Ausmultiplizieren eines geklammerten Ausdrucks verwenden.

Die beiden Gruppeneigenschaften sowie die Tatsache, dass die Verknüpfungen  $,+^{\prime}$  und  $,\cdot^{\prime}$  über das Distributivgesetz miteinander verwoben sind, macht  $(\mathbb{Z}_7, +, \cdot)$  zu jener algebraischen Struktur, die im Deutschen als *Körper* und im Englischen als *Field* bezeichnet wird.

### Definition 2.5 (Körper)

Eine Menge  $M$  mit den Verknüpfungen  $,+^{\prime}$  und  $,\cdot^{\prime}$ , geschrieben als  $(M, +, \cdot)$ , heißt *Körper*, wenn das Folgende gilt:

- $(M, +)$  ist eine kommutative Gruppe mit neutralem Element 0.
- $(M \setminus \{0\}, \cdot)$  ist eine kommutative Gruppe.
- $,+^{\prime}$  und  $,\cdot^{\prime}$  sind distributiv.

$$\text{☞ } x \cdot (y+z) = (x \cdot y) + (x \cdot z) \quad \text{für alle } x, y, z \in M$$

Beachten Sie, dass die Menge  $M \setminus \{0\}$  mit  $,\cdot^{\prime}$  eine kommutative Gruppe bilden muss, und nicht die Menge  $M$  selbst. Dennoch muss die Multiplikation  $,\cdot^{\prime}$  auch für die 0 definiert sein. In unserem Beispiel ergab die Multiplikation mit 0 immer den Wert 0, und es lässt sich zeigen, dass dies in jedem Körper der Fall sein muss.

Bevor wir diese Eigenschaft, neben einer Reihe weiterer Rechenregeln, formal herleiten, wollen wir noch weitere Beispiele ansehen:

### ■ Rationale, reelle und komplexe Zahlen

Für die klassische Mathematik besonders wichtig sind

- der Körper der rationalen Zahlen,   $(\mathbb{Q}, +, \cdot)$
- der Körper der reellen Zahlen,   $(\mathbb{R}, +, \cdot)$
- der Körper der komplexen Zahlen.   $(\mathbb{C}, +, \cdot)$

Dass die genannten Strukturen die Körpereigenschaften erfüllen, lässt sich leicht überprüfen:  $(\mathbb{Q}, +)$ ,  $(\mathbb{R}, +)$ ,  $(\mathbb{C}, +)$ ,  $(\mathbb{Q} \setminus \{0\}, \cdot)$ ,  $(\mathbb{R} \setminus \{0\}, \cdot)$  und  $(\mathbb{C} \setminus \{0\}, \cdot)$  sind kommutative Gruppen und die Verknüpfungen  $,+^\circ$  und  $,\cdot^\circ$  erfüllen zusammen das Distributivgesetz.

### ■ Restklassenkörper $(\mathbb{Z}_p, +, \cdot)$

In der Informations- und Codierungstheorie spielen die *Restklassenkörper*  $(\mathbb{Z}_p, +, \cdot)$  ( $p$  ist eine Primzahl) eine hervorgehobene Rolle. Auch hier lässt sich durch unsere geleistete Voraarbeit leicht einsehen, dass alle Körpereigenschaften erfüllt sind. Weiter oben haben wir sowohl  $(\mathbb{Z}_p, +)$  als auch  $(\mathbb{Z}_p \setminus \{0\}, \cdot)$  als Gruppen identifiziert, und  $,+^\circ$  und  $,\cdot^\circ$  verhalten sich auch dann distributiv zueinander, wenn wir die Operationen modulo  $p$  ausführen.

Ein spezieller Körper dieser Bauart ist uns bereits in Abbildung 2.19 begegnet. Es ist der Restklassenkörper  $(\mathbb{Z}_p, +, \cdot)$  für die Primzahl  $p = 7$ .

Der für uns wichtigste Körper dieser Bauart ist der Restklassenkörper  $(\mathbb{Z}_2, +, \cdot)$  in Abbildung 2.20. Er beruht auf einer zweielementigen Grundmenge und ist der kleinste Körper, den es gibt. In den nachfolgenden Kapiteln wird uns  $(\mathbb{Z}_2, +, \cdot)$  auf Schritt und Tritt begleiten.

Beachten Sie, dass  $(\mathbb{Z}_p, +, \cdot)$  nur dann die Körperaxiome erfüllt, wenn  $p$  eine Primzahl ist. Ist  $p$  faktorisierbar, so können wir in der Menge  $\mathbb{Z}_p$  mindestens ein Element  $x$  mit  $\text{ggT}(x, p) \neq 1$  finden, und nach Satz 2.1 gibt es für dieses Element kein multiplikatives Inverses (Abbildung 2.21). Damit ist gezeigt, dass  $(\mathbb{Z}_p, +, \cdot)$  dann und nur dann ein Körper ist, wenn wir für  $p$  eine Primzahl wählen.

### ■ Endliche Körper

Die Restklassenkörper  $(\mathbb{Z}_p, +, \cdot)$  sind spezielle *endliche Körper (finite fields)*, da ihre Grundmengen aus endlich vielen Elementen bestehen. Abbildung 2.22 zeigt, dass neben  $(\mathbb{Z}_p, +, \cdot)$  weitere Körper

+	0	1
0	0	1
1	1	0

.	0	1
0	0	0
1	0	1

Abb. 2.20: Der Körper  $(\mathbb{Z}_2, +, \cdot)$

mit einer endlichen Grundmenge existieren. Die dort abgebildete Struktur enthält 4 Elemente und erfüllt dennoch alle Körperaxiome aus Definition 2.5.

Endliche Körper haben in der Literatur verschiedene Namen. Zu Ehren des französischen Mathematikers Évariste Galois werden sie gerne als *Galoiskörper* bezeichnet (Abbildung 2.23). In manchen Fällen wird auch von *Galoisfeldern* gesprochen, in Anlehnung an den englischen Begriff *galois field*. Symbolisch haben sich für die Benennung eines endlichen Körpers mit  $n$  Elementen die Schreibweisen  $\mathbb{F}_n$  oder  $GF(n)$  etabliert.

+	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

.	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

Abb. 2.21: In  $(\mathbb{Z}_4, +, \cdot)$  gibt es für die Zahl 2 kein multiplikatives Inverses. Die Struktur verletzt das Gesetz der inversen Elemente und ist deshalb kein Körper.

Die prominente Rolle, die den endlichen Körpern über einer Grundmenge der Form  $\mathbb{Z}_m$  zukommt, unterstreicht der nachstehende Satz, den wir an dieser Stelle ohne formale Begründung akzeptieren wollen. Eine detaillierte Herleitung finden Sie z. B. in [107] oder [49]:

### Satz 2.4

- Genau zu den Primzahlpotenzen  $p^m$  ( $m \geq 1$ ) existieren endliche Körper mit  $p^m$  Elementen.
- Endliche Körper mit der gleichen Anzahl an Elementen sind zueinander isomorph.

Die Aussage dieses Satzes ist weitreichend. Sie attestiert, dass wir genau dann einen endlichen Körper finden können, wenn sich die Anzahl seiner Elemente in der Form  $p^m$  ( $p$  ist eine Primzahl) darstellen lässt, und er besagt im gleichen Atemzug, dass dieser Körper bis auf die Benennung der Symbole eindeutig bestimmt ist. Das bedeutet, dass wir in Abbildung 2.22, bei der Neudeinition der Verknüpfungen  $,+$  und  $\cdot$ , überhaupt keine Wahlmöglichkeit hatten. Der dort abgebildete Körper ist der einzige mit 4 Elementen, den es geben kann. Dass es überhaupt möglich war,  $,+$  und  $\cdot$  auf der Menge  $\mathbb{Z}_4$  passend zu definieren, folgt ebenfalls aus Satz 2.4, denn wir können die Zahl 4 in der Form  $p^m$  darstellen, mit der Primzahlbasis  $p = 2$  und dem Exponenten  $m = 2$ .

Der nächste Satz unterstreicht, dass wir in einem Körper in vertrauter Weise rechnen können:



### Satz 2.5 (Eigenschaften von Körpern)

In einem beliebigen Körper  $(M, +, \cdot)$  gilt:

- a)  $(y+z) \cdot x = (y \cdot x) + (z \cdot x)$  für alle  $x, y, z \in M$
- b)  $x \cdot 0 = 0 \cdot x = 0$  für alle  $x \in M$
- c)  $x \cdot (-y) = (-x) \cdot y = -(x \cdot y)$  für alle  $x, y \in M$
- d)  $x \cdot y = 0 \Rightarrow x = 0$  oder  $y = 0$  für alle  $x, y \in M$

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

.	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

Abb. 2.22: Galoiskörper  $(\mathbb{F}_4, +, \cdot)$

Beweis:

$$\begin{aligned} \text{a)} \quad (y+z) \cdot x &= x \cdot (y+z) \\ &= (x \cdot y) + (x \cdot z) \\ &= (y \cdot x) + (z \cdot x) \end{aligned}$$

$$\begin{aligned} \text{b)} \quad x \cdot 0 &= 0 + x \cdot 0 \\ &= (x \cdot x - x \cdot x) + x \cdot 0 \\ &= x \cdot x - (x \cdot x + x \cdot 0) \\ &= x \cdot x - x \cdot (x+0) \\ &= x \cdot x - x \cdot x = 0 \end{aligned}$$

$$\begin{aligned} 0 \cdot x &= 0 + 0 \cdot x \\ &= (x \cdot x - x \cdot x) + 0 \cdot x \\ &= x \cdot x - (x \cdot x + 0 \cdot x) \\ &= x \cdot x - (x+0) \cdot x \\ &= x \cdot x - x \cdot x = 0 \end{aligned}$$

c) Aus

$$x \cdot y + x \cdot (-y) = x \cdot (y + (-y)) = x \cdot 0 = 0$$

folgt, dass  $x \cdot (-y)$  das (nach Satz 2.3 eindeutig bestimmte) additive Inverse von  $x \cdot y$  ist, d. h., es ist

$$x \cdot (-y) = -(x \cdot y).$$

Analog folgt aus

$$x \cdot y + (-x) \cdot y = (x + (-x)) \cdot y = 0 \cdot y = 0,$$

dass  $(-x) \cdot y$  das additive Inverse von  $x \cdot y$  ist, d. h., es ist

$$(-x) \cdot y = -(x \cdot y).$$

- d) Die Behauptung folgt aus der Tatsache, dass  $\cdot$  auf  $M \setminus \{0\}$  abgeschlossen ist.  $\square$



Évariste Galois (1811 – 1832)

**Abb. 2.23:** Der französische Mathematiker Évariste Galois gehört zu den Begründern der Gruppentheorie. Ihm war es gelungen, die Ergebnisse von Abel über die Lösung von Polynomgleichungen höherer Grade zu einer allgemeinen Theorie weiterzuentwickeln, die wir heute als *Galois-Theorie* bezeichnen.

Genau wie Abel starb auch Galois in jungen Jahren. Sein letztes Manuskript, das er in einer schlaflosen Nacht vom 29. auf den 30. Mai 1832 hektisch zu Papier brachte, zieren die Worte „*je n'ai pas le temps*“, dt. „*mir fehlt die Zeit*“. Tatsächlich blieben Galois nur noch wenige Stunden bis zu seinem Tod. Er hatte sich auf ein Pistolenduell eingelassen, wohlwissend, dass sein Antagonist ein treffsicherer Schütze war.

Es war kurz nach Sonnenaufgang, als ein gezielter Schuss ein klaffendes Loch in Galois Bauchdecke riss. Nach mehreren Stunden wurde der Zurückgelassene von einem Bauer zufällig entdeckt und in ein nahe gelegenes Krankenhaus gebracht. Helfen konnte ihm dort niemand mehr. Am nächsten Tag erlag Galois seiner schweren Verletzung, im Alter von nur 20 Jahren.

Für die systematische Untersuchung von Körpern fehlen in unserem Repertoire noch wichtige Begriffe. Diese werden wir jetzt der Reihe nach einführen:



### Definition 2.6 (Unter- und Oberkörper, Körpererweiterung)

Mit  $(M, +, \circ)$  und  $(M', +, \circ)$  seien zwei Körper gegeben.

- Ist  $M' \subseteq M$ , so nennen wir

- ☞  $M'$  einen *Unterkörper* von  $M$ ,
- ☞  $M$  einen *Oberkörper* von  $M'$  und
- ☞ das Paar  $M'/M$  eine *Körpererweiterung*.

- Ist  $M' \subset M$ , so nennen wir

- ☞  $M'$  einen *echten Unterkörper* von  $M$  und
- ☞  $M$  einen *echten Oberkörper* von  $M'$  und
- ☞ das Paar  $M'/M$  eine *echte Körpererweiterung*.

- Enthält  $M$  keine echten Unterkörper, so nennen wir

- ☞  $M$  einen *Primkörper*.

Die Unterkörpereigenschaft lässt sich anhand des *Unterkörperkriteriums* überprüfen.  $M'$  ist genau dann ein Unterkörper von  $M$ , wenn

- die neutralen Elemente in  $M'$  enthalten sind,  
☞  $0 \in M', 1 \in M'$
- die Addition inverser Elemente nicht aus  $M'$  herausführt,  
☞  $x, y \in M' \Rightarrow x - y \in M'$
- die Multiplikation inverser Elemente nicht aus  $M'$  herausführt.  
☞  $x, y \in M', y \neq 0 \Rightarrow x \cdot y^{-1} \in M'$

Ebenfalls wichtig ist dieser Begriff:



### Definition 2.7 (Charakteristik)

Die *Charakteristik* eines Körpers  $M$  ist die kleinste Zahl  $c$  mit

$$\underbrace{1 + 1 + 1 + \dots + 1}_{c\text{-mal}} = 0$$

Die Charakteristik sei 0, falls kein solches  $c$  existiert.

Offenbar kann die Charakteristik eines Körpers nur dann 0 sein, wenn die Addition der 1 stets ein neues Element hervorbringt (Abbildung 2.24 oben) oder der Prozess irgendwann in eine Schleife gerät (Abbildung 2.24 Mitte). Eine Schleife ist aber nur dann möglich, wenn sie, wie in Abbildung 2.24 (unten) gezeigt, auf die 0 zurückführt. Andernfalls gäbe es zwei verschiedene Elemente  $x$  und  $y$  mit der Eigenschaft  $x+1 = y+1$ , was nicht sein kann. Das bedeutet, dass die Charakteristik eines endlichen Körpers immer größer als 0 sein muss und ausschließlich unendliche Körper eine Charakteristik von 0 aufweisen können.

Sammeln wir die Elemente, die das neutrale Element 1 hervorbringt, der Reihe nach auf, so erhalten wir mit

$$( \{1, 1+1, 1+1+1, \dots, \underbrace{1+1+\dots+1}_{c\text{-mal}} \}, \cdot, + ) \quad (2.9)$$

einen Unterkörper von  $M$ . Da jeder Unterkörper zwingend die 1 enthält und die Addition abgeschlossen sein muss, ist er der kleinstmögliche. Mit anderen Worten: (2.9) ist ein Primkörper.

Der Primkörper eines endlichen Körpers  $M$  ist immer eindeutig bestimmt und die Anzahl seiner Elemente entspricht der Charakteristik von  $M$ . Sehen wir genauer hin, so können wir über die Größe des Primkörpers eine noch tiefergehende Aussagen treffen. Hierzu nehmen wir an, die Charakteristik sei eine faktorisierbare Zahl, d.h.,  $c$  sei in der Form  $c = x \cdot y$  darstellbar mit  $x, y \geq 2$ . Dann wäre

$$\underbrace{1+1+1+\dots+1}_{c\text{-mal}} = (\underbrace{1+\dots+1}_{x\text{-mal}}) \cdot (\underbrace{1+\dots+1}_{y\text{-mal}}) = 0$$

und aus Satz 2.5 würde dann sofort

$$\underbrace{(1+\dots+1)}_{x\text{-mal}} = 0 \quad \text{oder} \quad \underbrace{(1+\dots+1)}_{y\text{-mal}} = 0$$

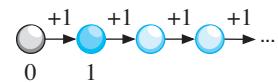
folgen, im Widerspruch zur Minimalitätseigenschaft der Charakteristik. Damit haben wir den folgenden Satz bewiesen:

### Satz 2.6

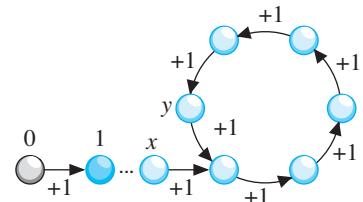
Die Charakteristik eines Körpers ist 0 oder eine Primzahl.

Wir wollen die Aussage dieses Satzes anhand der bisher gesehenen Körper überprüfen. Die Charakteristik des Restklassenkörpers  $(\mathbb{Z}_p, +, \cdot)$  ist

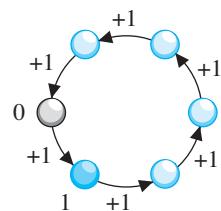
#### ■ Szenario 1



#### ■ Szenario 2



#### ■ Szenario 3



**Abb. 2.24:** Von den drei dargestellten Szenarien sind nur das erste und das letzte möglich. Würde die sukzessive Addition der 1 eine Schleife hervorbringen, wie sie in der Mitte dargestellt ist, so gäbe es zwei Elemente  $x$  und  $y$ , für die gleichzeitig  $x \neq y$  und  $x+1 = y+1$  gelten würden.

Für endliche Körper scheidet zudem die erste Variante aus, da nur eine begrenzte Zahl an Elementen zur Verfügung steht. In diesen Strukturen muss die sukzessive Addition von 1 irgendwann auf die 0 zurückführen.

$+$	0	1		
0	0	1		
1	1	0		
			2	3
			2	3
			3	2
			2	3
			3	1
			2	0
			3	0
			2	1
			1	0

$\cdot$	0	1		
0	0	0		
1	0	1		
			2	3
			0	0
			2	3
			2	1
			3	0
			3	2

**Abb. 2.25:** Extrahieren wir den Primkörper von  $(\mathbb{F}_4, +, \cdot)$ , so erhalten wir eine bekannte Struktur. Es ist der Restklassenkörper  $(\mathbb{Z}_2, +, \cdot)$ .

gleich  $p$ ; sie ist eine Primzahl und stimmt mit der Anzahl der Körperelemente überein. Das bedeutet, dass  $(\mathbb{Z}_p, +, \cdot)$  sein eigener Primkörper ist und keine echten Unterkörper besitzt.

Für den vierelementigen Körper aus Abbildung 2.22 ist klar, dass wir ein anderes Ergebnis erhalten müssen. Wir wissen bereits, dass die Charakteristik eine Primzahl ist und damit kleiner als 4 sein muss. Mit einem Blick auf die Verknüpfungstabelle können wir die Charakteristik auch sofort bestimmen. Da bereits die zweimalige Addition von 1 die 0 ergibt, besitzt der Körper die Charakteristik 2, und daraus folgt, dass er einen zweielementigen Primkörper in sich birgt. Abbildung 2.25 entlarvt den Primkörper als einen alten Bekannten: Es handelt sich um den Restklassenkörper  $(\mathbb{Z}_2, +, \cdot)$  aus Abbildung 2.20.

Tatsächlich lässt sich dieses Ergebnis weiter verallgemeinern. Jeder endliche Körper mit  $p^n$  Elementen besitzt die Charakteristik  $p$  und einen zu  $\mathbb{Z}_p$  isomorphen Primkörper. Da wir jeden Körper als eine Erweiterung seines Primkörpers ansehen dürfen, lässt sich dieses Ergebnis plakativ auch so formulieren: Die endlichen Körper sind die Körpererweiterungen der Restklassenkörper  $(\mathbb{Z}_p, +, \cdot)$ .

Ein wichtiger Punkt ist immer noch offen: Obwohl wir jetzt wissen, dass die endlichen Körper Erweiterungen der vertrauten Restklassenkörper sind, haben wir keine Kenntnis darüber, wie sie im Einzelnen aufgebaut sind. Die einzige Körpererweiterung, die wir bisher mit eigenen Augen gesehen haben, war durch die Verknüpfungstabelle aus Abbildung 2.22 gegeben. Dass die Tabelle quasi vom Himmel fiel, hatte einen einfachen Grund: Unsere mathematischen Hilfsmittel sind zum gegenwärtigen Zeitpunkt noch zu schwach, um ihren Aufbau systematisch zu begründen.

Dies wird sich schon bald ändern. Am Ende dieses Kapitels werden Sie verstehen, warum die Tabelle so aussah und nicht anders, und Sie werden in der Lage sein, die endlichen Körper systematisch zu konstruieren. Bis dahin ist es aber noch ein langer und an manchen Stellen beschwerlicher Weg, der uns das Gebiet der endlichen Körper erst einmal verlassen lässt. Er wird uns im nächsten Abschnitt zunächst zu den Ringstrukturen führen und anschließend die spezielle Struktur der Polynomringe entdecken lassen. Tatsächlich werden sich die Polynome als der Schlüssel zum Erfolg erweisen; über sie werden wir einen konstruktiven Zugang zu den für unsere Zwecke so wichtigen endlichen Körpern erhalten.

### 2.3.3 Ringe

In Abschnitt 2.3.2 haben wir erkannt, dass die Menge  $\mathbb{Z}_m$ , zusammen mit der Addition und Multiplikation modulo  $m$ , nur dann einen Körper bildet, wenn  $m$  eine Primzahl ist. In diesem Fall, und nur in diesem, besitzt jedes Element auch ein multiplikatives Inverses. Gut zu erkennen war diese Eigenschaft in Abbildung 2.21 an der Verknüpfungstabelle der Multiplikation in  $\mathbb{Z}_4$ . Von den Zahlen größer 0 besaßen nur die 1 und die 3, nicht aber die 2 ein multiplikatives Inverses.

Um auch solche Strukturen systematisch behandeln zu können, werden wir die strenge Definition des Körpers nun geringfügig abschwächen. Auf diese Weise erreichen wir eine algebraische Struktur, die in der Mathematik als *Ring* bezeichnet wird.



#### Definition 2.8 (Ring)

Eine Menge  $M$  mit den Verknüpfungen  $,+$  und  $,\cdot$ , geschrieben als  $(M, +, \cdot)$ , heißt *Ring*, wenn das Folgende gilt:

- $(M, +)$  ist eine kommutative Gruppe.
- $(M, \cdot)$  ist eine Halbgruppe.
- $,+$  und  $,\cdot$  sind links- und rechtsdistributiv.

$$\text{☞ } x \cdot (y + z) = (x \cdot y) + (x \cdot z) \quad \text{für alle } x, y, z \in M$$

$$\text{☞ } (y + z) \cdot x = (y \cdot x) + (z \cdot x) \quad \text{für alle } x, y, z \in M$$

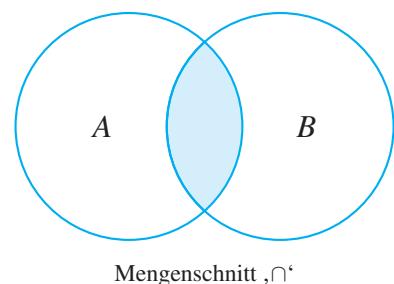
Wir vereinbaren zusätzlich:

- Ist  $,\cdot$  kommutativ, so ist  $(M, +, \cdot)$  ein *kommutativer Ring*.
- Enthält  $(M, \cdot)$  ein neutrales Element, so ist  $M$  ein *Ring mit 1*.

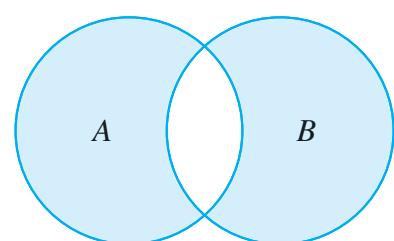
Schauen wir auch hier ein paar Beispiele an:

- $(\mathbb{Z}, +, \cdot)$ , der Ring der ganzen Zahlen

Die Menge der ganzen Zahlen bildet zusammen mit der gewöhnlichen Addition und der gewöhnlichen Multiplikation einen kommutativen Ring mit 1. Warum wir keinen Körper vor uns haben, ist leicht zu durchschauen.  $(\mathbb{Z} \setminus \{0\}, \cdot)$  müsste dann eine Gruppe sein und für jedes Element ein multiplikatives Inverses vorhalten. Außer für die 1, die zu sich selbst invers ist, ist der Kehrwert einer ganzen



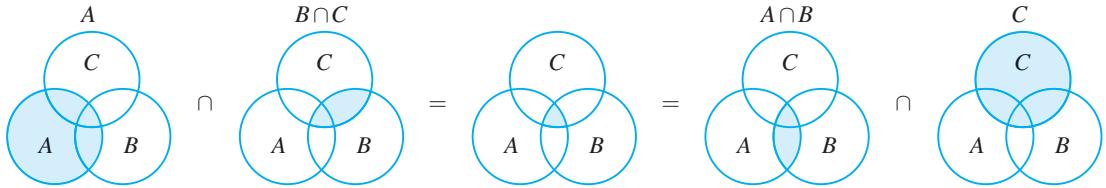
Mengenschnitt  $,\cap^c$



Symmetrische Differenz  $,\triangle^c$

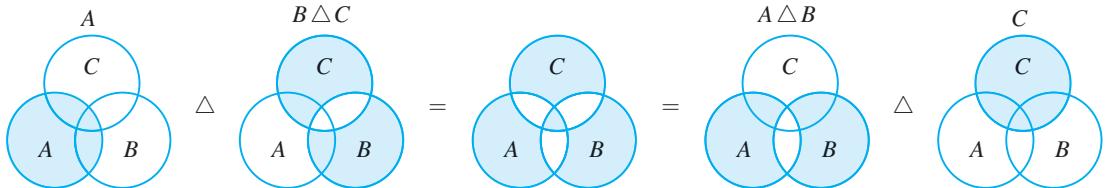
**Abb. 2.26:** Die Potenzmenge  $\mathfrak{P}(M)$  einer Menge  $M$  bildet zusammen mit der Schnittoperation  $,\cap^c$  und der symmetrischen Differenz  $,\triangle^c$  einen kommutativen Ring mit 1.

## Assoziativitat der Schnittoperation



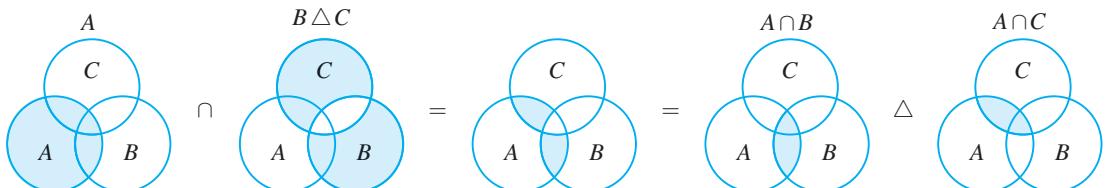
$$\text{☞ } A \cap (B \cap C) = (A \cap B) \cap C$$

## Assoziativitat der symmetrischen Differenz



$$\text{☞ } A \triangle (B \triangle C) = (A \triangle B) \triangle C$$

## Distributivitat von „∩“ und „△“



$$\text{☞ } A \cap (B \triangle C) = (A \cap B) \triangle (A \cap C)$$

**Abb. 2.27:** Die Mengenoperationen „ $\cap$ “ und „ $\triangle$ “ sind assoziativ und erfüllen das Distributivgesetz, das wir in der Definition der Ringstruktur gefordert haben.

Zahl aber selbst keine ganze Zahl.  $(\mathbb{Z} \setminus \{0\}, \cdot)$  ist somit keine Gruppe, sondern lediglich ein (kommutativer) Monoid.

■ Die endlichen Ringe  $(\mathbb{Z}_m, +, \cdot)$

Die Argumentation ist die gleiche wie bei  $(\mathbb{Z}, +, \cdot)$ . Die Menge  $\mathbb{Z}_m$  bildet zusammen mit der Multiplikation modulo  $m$  im Allgemeinen keine Gruppe, sondern lediglich einen kommutativen Monoid. Das bedeutet, dass  $(\mathbb{Z}_m, +, \cdot)$  zwar kein Korper ist, wohl aber ein kommutativer Ring mit 1. Ist  $m$  eine Primzahl, so wird  $(\mathbb{Z}_m, +, \cdot)$  aufgrund der Gruppeneigenschaft von  $(\mathbb{Z}_m \setminus \{0\}, \cdot)$  zu einem Korper.

■ Der Ring  $(\mathfrak{P}(M), \triangle, \cap)$

Hier ist  $M$  eine beliebige Trägermenge,  $\mathfrak{P}(M)$  die Potenzmenge von  $M$  (Menge aller Teilmengen von  $M$ ),  $\cap$  die Schnittmengenoperati-

on und  $\Delta$  die *symmetrische Differenz* (Abbildung 2.26):

$$A \Delta B := (A \setminus B) \cup (B \setminus A)$$

Wir wollen uns davon überzeugen, dass wirklich alle Ringeigenschaften erfüllt sind. Zunächst ist leicht zu erkennen, dass die Schnittoperation  $\cap$  kommutativ und assoziativ ist (Abbildung 2.27 oben) und mit  $M \in \mathfrak{P}(M)$  ein neutrales Element existiert; der Schnitt einer Menge  $A \in \mathfrak{P}(M)$  mit der Menge  $M$  liefert stets die Menge  $A$  zurück. Diese Eigenschaften machen  $(\mathfrak{P}(M), \cap)$  zu einer kommutativen Halbgruppe, in der ein neutrales Element existiert (Abelscher Monoid). Die symmetrische Differenz ist ebenfalls kommutativ und assoziativ (Abbildung 2.27 Mitte), und mit  $\emptyset$  existiert auch hier ein neutrales Element:

$$A \Delta \emptyset = (A \setminus \emptyset) \cup (\emptyset \setminus A) = A \cup \emptyset = A$$

Ferner gibt es zu jeder Menge  $A \in \mathfrak{P}(M)$  eine andere Menge mit der Eigenschaft, dass die symmetrische Differenz dieser beiden die leere Menge  $\emptyset$  ergibt. Dieses inverse Element ist die Menge  $A$  selbst:

$$A \Delta A = (A \setminus A) \cup (A \setminus A) = \emptyset \cup \emptyset = \emptyset$$

Damit ist  $(\mathfrak{P}(M), \Delta)$  als kommutative Gruppe identifiziert. Abbildung 2.27 (unten) zeigt, dass beide Verknüpfungen distributiv sind, und  $(\mathfrak{P}(M), \Delta, \cap)$  damit alle Eigenschaften eines kommutativen Rings mit 1 erfüllt.

#### ■ Der Ring der $n \times n$ -Matrizen $(\mathbb{K}^{n \times n}, +, \cdot)$

Ist  $\mathbb{K}$  ein Körper, so bildet  $\mathbb{K}^{n \times n}$ , die Menge der quadratischen Matrizen mit  $n$  Zeilen und  $n$  Spalten, einen Ring. Die Verknüpfungen  $+$  und  $\cdot$  sind die gewöhnliche Addition und Multiplikation für Matrizen. Zum Beispiel gilt für  $n = 2$ :

$$\begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} + \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{pmatrix} := \begin{pmatrix} a_{00} + b_{00} & a_{01} + b_{01} \\ a_{10} + b_{10} & a_{11} + b_{11} \end{pmatrix}$$

$$\begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \cdot \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{pmatrix} := \begin{pmatrix} a_{00}b_{00} + a_{01}b_{10} & a_{00}b_{01} + a_{01}b_{11} \\ a_{10}b_{00} + a_{11}b_{10} & a_{10}b_{01} + a_{11}b_{11} \end{pmatrix}$$

Die Menge  $\mathbb{K}^{n \times n}$  bildet zusammen mit der Addition eine kommutative Gruppe. Die Addition ist abgeschlossen, assoziativ und kommutativ; das neutrale Element ist die Nullmatrix (Abbildung 2.28), und das additive inverse Element einer Matrix entsteht, indem die Vorzeichen aller Matrixelemente invertiert werden. Die Multiplikation ist auf  $\mathbb{K}^{n \times n}$  ebenfalls abgeschlossen und assoziativ. Ferner existiert mit

■ Neutrales Element bezüglich  $,$   $+$

$$\begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix}$$

■ Neutrales Element bezüglich  $,$   $\cdot$

$$\begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix}$$

**Abb. 2.28:** Sowohl für die Addition als auch für die Multiplikation existieren neutrale Elemente.

$$E_2 := \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, E_3 := \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \dots$$

**Abb. 2.29:** Die Einheitsmatrizen folgen einem einheitlichen Schema. Sie enthalten auf der Hauptdiagonalen (dies sind die Elemente  $a_{ii}$ ) Einsen und sind ansonsten mit Nullen gefüllt.

$$\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ b_{10} & b_{11} \end{pmatrix}$$

**Abb. 2.30:** Die linke Matrix besitzt kein multiplikatives Inverses. Egal, womit wir sie multiplizieren: Wir erhalten links oben niemals eine 1 und damit niemals die Einheitsmatrix.

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 4 & 3 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 3 & 4 \\ 1 & 2 \end{pmatrix}$$

**Abb. 2.31:** Die Matrixmultiplikation ist im Allgemeinen nicht kommutativ.

der *Einheitsmatrix* ein Element, das sich bezüglich der Multiplikation neutral verhält (Abbildung 2.29). Dies macht  $(\mathbb{K}^{n \times n}, \cdot)$  aber noch nicht zu einer Gruppe. Hierzu müsste jede Matrix auch ein multiplikatives Inverses besitzen, was nicht der Fall ist (Abbildung 2.30). Damit ist  $(\mathbb{K}^{n \times n}, +, \cdot)$  ein Ring mit 1, aber kein Körper. Beachten Sie, dass  $(\mathbb{K}^{n \times n}, +, \cdot)$ , anders als die bisher betrachteten Beispiele, kein kommutativer Ring ist. Das Beispiel in Abbildung 2.31 zeigt, dass es bei der Matrixmultiplikation durchaus eine Rolle spielt, auf welcher Seite ein Faktor steht.

■ Die endlichen Ringe  $(\{0, 2, 4, \dots, 2n\}, +, \cdot)$

Die Menge der geraden Zahlen zwischen 0 und  $2n$  bildet für jede Zahl  $n \in \mathbb{N}$  einen kommutativen Ring, wenn die Verknüpfungen  $,$   $+$  und  $,$   $\cdot$  modulo  $2n + 2$  ausgeführt werden. Verantwortlich hierfür ist die Tatsache, dass die Summe und das Produkt zweier gerader Zahlen ebenfalls wieder eine gerade Zahl ergibt. Für  $n \neq 0$  ist diese Struktur die erste der bisher betrachteten, in der kein neutrales Element für die Multiplikation enthalten ist. Wir haben mit ihr einen Ring gefunden, der kein Ring mit 1 ist.

Das letzte Beispiel wollen wir noch etwas genauer betrachten. Die Menge  $\{0, 2, 4, \dots, 2n\}$  ist eine Teilmenge von  $\mathbb{Z}_{2n+2}$ , und weiter oben haben wir nachgewiesen, dass  $(\mathbb{Z}_m, +, \cdot)$  alle Ringaxiome erfüllt. Das bedeutet, dass wir einen Ring vor uns haben, der in einen größeren Ring eingebettet ist, und dies führt uns auf direktem Weg zum Begriff des Unterring (Abbildung 2.32):



### Definition 2.9 (Unter- und Oberring, Ringerweiterung)

Mit  $(M, +, \cdot)$  und  $(M', +, \cdot)$  seien zwei Ringe gegeben.

■ Ist  $M' \subseteq M$ , so nennen wir

- ☞  $M'$  einen *Unterring* von  $M$ ,
- ☞  $M$  einen *Oberring* von  $M'$  und
- ☞ das Paar  $M'/M$  eine *Ringerweiterung*.

■ Ist  $M' \subset M$ , so nennen wir

- ☞  $M'$  einen *echten Unterring* von  $M$  und
- ☞  $M$  einen *echten Oberring* von  $M'$  und
- ☞ das Paar  $M'/M$  eine *echte Ringerweiterung*.

■ Existiert in  $(M', \cdot)$  ein neutrales Element, so nennen wir

- ☞  $M'$  einen *Unterring mit 1*.

Umsortieren der Additionstabelle						
$+$	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

Umsortieren der Multiplikationstabelle						
$\cdot$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	2	3	4	5
2	0	2	4	0	2	4
3	0	3	0	3	0	3
4	0	4	2	0	4	2
5	0	5	4	3	2	1

Unterring von $(\mathbb{Z}_m, +, \cdot)$			
$+$	0	2	4
0	0	2	4
2	2	4	0
4	4	0	2
..	..	..	..
1	1	3	5
3	3	5	1
5	5	1	3
..	..	..	..
1	1	3	5
3	3	5	1
5	5	1	3
..	..	..	..
1	0	2	4
3	0	0	0
5	0	4	2
..	..	..	..
1	0	2	4
3	0	0	0
5	0	4	2

**Abb. 2.32:** Betrachten wir nur die positiven Zahlen innerhalb von  $\mathbb{Z}_m$ , so führt die Addition und die Multiplikation nicht heraus. Die Abgeschlossenheit macht diese Menge zu einem Unterring von  $(\mathbb{Z}_m, +, \cdot)$ , hier demonstriert für den Fall  $m = 6$ .

Die Unterringeigenschaft lässt sich anhand des *Unterringkriteriums* überprüfen. Dieses besagt, dass  $M'$  genau dann ein Unterring von  $M$  ist, wenn

- die Addition inverser Elemente nicht aus  $M'$  herausführt,  
👉  $x, y \in M' \Rightarrow x - y \in M'$
- die Multiplikation auf  $M'$  abgeschlossen ist.  
👉  $x, y \in M' \Rightarrow x \cdot y \in M'$

### 2.3.4 Ideale

Für unsere Beispieldmenge  $\{0, 2, 4, \dots, 2n\}$  war entscheidend, dass die Addition und die Multiplikation nicht aus den geraden Zahlen herausführt; erst durch diese Abgeschlossenheitseigenschaft wurde die Menge zu einem Unterring von  $(\mathbb{Z}, +, \cdot)$ . Tatsächlich gilt in unserem Beispiel noch mehr. Multiplizieren wir ein beliebiges Element aus  $\mathbb{Z}_m$ , ob gerade oder ungerade, mit einer geraden Zahl, so entsteht wiederum eine gerade Zahl. Mit anderen Worten: Die Multiplikation führt stets in die Menge der geraden Zahlen *hinein*. Diese verstärkte Abgeschlossenheitseigenschaft macht den Unterring der geraden Zahlen zu einer algebraischen Struktur, die einen eigenständigen Namen trägt.



#### Definition 2.10 (Ideal)

Sei  $(M, +, \cdot)$  ein Ring und  $M'$  ein Unterring von  $M$ .

- $M'$  heißt *Linksideal*,  
wenn aus  $m \in M$  und  $m' \in M'$  stets  $m \cdot m' \in M'$  folgt.
- $M'$  heißt *Rechtsideal*,  
wenn aus  $m \in M$  und  $m' \in M'$  stets  $m' \cdot m \in M'$  folgt.
- $M'$  heißt *Ideal*,  
wenn  $M'$  gleichzeitig ein Linksideal und ein Rechtsideal ist.

Der Begriff des Unterrings ist in der Literatur unterschiedlich definiert. In manchen Quellen wird nur dann von einem Unterring gesprochen, wenn das neutrale Element der Multiplikation darin enthalten ist. In diesem Fall ist ein Unterring das, was wir als *Unterring mit 1* bezeichnen. Der Unterschied klingt marginal, doch seine Konsequenzen sind beträchtlich.

Legen wir die alternative Definition zugrunde, so besitzt ein Ring ohne 1 keinen einzigen Unterring. Noch größere Konsequenzen ergeben sich für den Begriff des Ideals. Mit unserem Begriffsgerüst konnten wir Ideale als spezielle Unterringe definieren, die eine verstärkte Abgeschlossenheitseigenschaft erfüllen. Legen wir die alternative Definition zugrunde, dann wäre nicht mehr jedes Ideal ein Unterring. Tatsächlich würde nur noch eines der trivialen Ideale diese Eigenschaft erfüllen, da nur dieses die 1 enthalten kann. Ideale und Unterringe wären dann zueinander schief liegende Begriffe, die, anders als hier, in keiner Hierarchiebeziehung mehr stehen.

Jeder Ring  $(M, +, \cdot)$  hat mindestens zwei Ideale: die Menge  $\{0\}$  und die Menge  $M$ . Wir nennen sie die *trivialen Ideale*. Ein Ideal, das nicht gleich dem Ring selbst ist, bezeichnen wir als *echtes Ideal*. Die Unterscheidung in Links- und Rechtsideale ist nur für nichtkommutative Ringe von Bedeutung. Ist nämlich die Multiplikation kommutativ, so muss jedes Linksideal auch ein Rechtsideal sein und umgekehrt.

Beachten Sie, dass jedes Ideal auch ein Unterring ist, aber nur selten ein Unterring mit 1. Enthält nämlich ein Ideal die 1, so muss es nach Definition 2.10 auch alle Elemente enthalten, die wir durch Multiplikation eines Ringelements mit 1 erhalten können. Wegen der Neutralität der 1 sind dies ausnahmslos alle Ringelemente. Somit ist das einzige Ideal, das gleichzeitig ein Unterring mit 1 ist, der Ring selbst. Damit haben wir zugleich eine einfache Möglichkeit an der Hand, um echte Ideale zu charakterisieren: Ein Ideal eines Rings mit 1 ist genau dann echt, wenn es nicht die 1 enthält.

## 2.4 Endliche Körper

### 2.4.1 Polynomringe

Falls Sie mit der algebraischen Struktur des Rings bereits vertraut waren, hatten Sie mit hoher Wahrscheinlichkeit ein ganz bestimmtes Beispiel vermisst: den Polynomring  $\mathbb{K}[x]$ .

Ist  $\mathbb{K}$  ein Körper, so hat ein Polynom aus der Menge  $\mathbb{K}[x]$  die Form

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0 \quad (2.10)$$

mit  $a_n, a_{n-1}, \dots, a_1, a_0 \in \mathbb{K}$ . Die folgenden Beispiele verdeutlichen den Polynombegriff:

- $x^2 + 1$  ist ein Polynom aus  $\mathbb{Z}_2[x]$ .
- $2x^2 + x$  ist ein Polynom aus  $\mathbb{Z}_3[x]$ , aber nicht aus  $\mathbb{Z}_2[x]$ .
- $2x^2 + \frac{1}{3}x$  ist ein Polynom aus  $\mathbb{Q}[x]$ , aber nicht aus  $\mathbb{Z}_3[x]$ .
- $2x^2 + x + \sqrt{2}$  ist ein Polynom aus  $\mathbb{R}[x]$ , aber nicht aus  $\mathbb{Q}[x]$ .
- $(1+4i)x^2 + \sqrt{2}$  ist ein Polynom aus  $\mathbb{C}[x]$ , aber nicht aus  $\mathbb{R}[x]$ .

Ein wichtiger Begriff ist der *Grad (degree)* eines Polynoms  $p(x)$ , geschrieben als  $\deg p(x)$ . Er beschreibt den größten in  $p(x)$  vorkommenden Exponenten und lässt sich formal so definieren:

$$\deg p(x) := \max\{i \mid a_i \neq 0\} \quad (2.11)$$

Alle genannten Beispiele sind Polynome vom Grad 2. Vielleicht ist Ihnen aufgefallen, dass der Ausdruck (2.11) auf der rechten Seite gar nicht für alle Polynome definiert ist. Für das Nullpolynom degeneriert die Koeffizientenmenge zu  $\emptyset$ , sodass wir kein Maximum bestimmen können. Wir umgehen dieses Problem durch die gesonderte Vereinbarung  $\deg 0 := -\infty$ .

Die Addition und die Multiplikation von Polynomen folgt den gewöhnlichen Rechenregeln. Die Summe  $p(x) + q(x)$  und das Produkt  $p(x) \cdot q(x)$  lassen sich ausrechnen, indem die Koeffizienten komponentweise addiert bzw. die beiden Polynome wie gewohnt „ausmultipliziert“ werden (Abbildung 2.33). Beide Verknüpfungen sind sowohl kommutativ als auch assoziativ, und mit dem Polynom  $n(x) = 0$  bzw.

#### Polynomarithmetik in $\mathbb{R}[x]$

##### ■ Addition

$$(x^2 + x) + (x + 1) = x^2 + 2x + 1$$

##### ■ Multiplikation

$$\begin{array}{r} (x^2 + x) \cdot (x + 1) \\ \hline x^3 + x^2 \\ \quad x^2 + x \\ \hline x^3 + 2x^2 + x \end{array}$$

#### Polynomarithmetik in $\mathbb{Z}_2[x]$

##### ■ Addition

$$(x^2 + x) + (x + 1) = x^2 + 1$$

##### ■ Multiplikation

$$\begin{array}{r} (x^2 + x) \cdot (x + 1) \\ \hline x^3 + x^2 \\ \quad x^2 + x \\ \hline x^3 + x \end{array}$$

**Abb. 2.33:** Addition und Multiplikation von Polynomen

Achten Sie darauf, strikt zwischen Polynomen und deren zugehörigen *Polynomfunktionen* zu unterscheiden. Ein Polynom wird durch die Folge seiner Koeffizienten definiert. Folgerichtig sind



$$\begin{aligned} p(x) &:= 0 \\ q(x) &:= x^2 + x \end{aligned}$$

zwei verschiedene Polynome aus  $\mathbb{Z}_2[x]$ . Darüber hinaus können wir jedes Polynom aus der Menge  $\mathbb{Z}_2[x]$  als eine Funktion der Form

$$f : \mathbb{Z}_2 \rightarrow \mathbb{Z}_2$$

auffassen. Für unsere Beispieldpolynome sehen diese Funktionen so aus:

$$\begin{aligned} f_p : x &\mapsto 0 \\ f_q : x &\mapsto x^2 + x \end{aligned}$$

Für beide Funktionen erhalten wir immer den Wert 0, egal, welches Element aus  $\mathbb{Z}_2$  wir für die Zahl  $x$  auch einsetzen. Das bedeutet, dass die Polynomfunktionen zweier verschiedener Polynome nicht zwangsläufig verschieden sein müssen.

Eine Verwechslung ist ausgeschlossen, wenn Sie sich Polynome als eine endliche Folge von Koeffizienten vorstellen:

$$\begin{aligned} p(x) &\equiv (0,0,0) \\ q(x) &\equiv (1,1,0) \end{aligned}$$

In dieser Darstellung wird offensichtlich, warum wir zwei Polynome genau dann als gleich erachten, wenn ihre Koeffizienten identisch sind.

In Abschnitt 2.5 werden wir diese Sichtweise im Zusammenhang mit der Diskussion von Vektorräumen weiter vertiefen. Dort wird sich fast von selbst ergeben, dass ein Polynom vom Grad  $n$  zu einem Element eines Vektorraums der Dimension  $n+1$  wird, wenn die Koeffizienten als die Komponenten eines Vektors interpretiert werden.

$e(x) = 1$  existiert jeweils ein neutrales Element. Damit sind  $(\mathbb{K}[x], +)$  und  $(\mathbb{K}[x], \cdot)$  als Abel'sche Monoide identifiziert. Da für jedes Polynom der Form (2.10) mit  $-a_nx^n - a_{n-1}x^{n-1} - \dots - a_1x - a_0$  ein additives Inverses existiert, ist  $(\mathbb{K}[x], +)$  sogar eine Gruppe. Aus der Distributivität von  $,$   $+$  und  $\cdot$  folgt nun sofort der nachstehende Satz:



### Satz 2.7

Für jeden Körper  $\mathbb{K}$  ist  $\mathbb{K}[x]$  ein kommutativer Ring mit 1.

Ist  $\mathbb{K}[x]$  vielleicht sogar ein Körper? Die Antwort lautet Nein, da nicht für alle Polynome multiplikative Inverse existieren. Beispielsweise gibt es in  $\mathbb{R}[x]$  kein Polynom  $q(x)$ , das die Gleichung  $p(x) \cdot q(x) = 1$  löst, wenn der Grad von  $p$  größer als 1 ist. Kurzum:  $(\mathbb{K}[x], \cdot)$  ist im Allgemeinen keine Gruppe.

Obwohl wir wegen der fehlenden Inversen nicht uneingeschränkt dividieren können, lässt sich im Ring der Polynome zumindest eine *Division mit Rest* durchführen.

### Polynomdivision

Sind  $p(x)$  und  $q(x)$  zwei Polynome, so ist es die Aufgabe der *Polynomdivision*,  $p(x)$  in der Form

$$p(x) = q(x)s(x) + r(x) \quad (2.12)$$

darzustellen mit  $\deg r(x) < \deg q(x)$ . Eine solche Zerlegung ist immer möglich, und das Polynom  $s(x)$  und der *Divisionsrest*  $r(x)$  sind eindeutig bestimmt. Abbildung 2.34 demonstriert an mehreren Beispielen, wie sich die Polynomdivision systematisch durchführen lässt. In jeder Iteration wird ein Glied der Form  $a_i x^i$  so bestimmt, dass die Subtraktion von  $q(x) \cdot a_i x^i$  den führenden Teilterm des Dividenden verschwinden lässt, und dieser Schritt wird so lange wiederholt, bis kein passendes  $a_i x^i$  mehr gefunden werden kann. Die Division funktioniert in allen Polynomringen  $\mathbb{K}[x]$ , unabhängig von der Wahl des Körpers  $\mathbb{K}$ .

Lässt sich ein Polynom  $p(x)$  ohne Rest durch  $q(x)$  dividieren, so nennen wir  $q(x)$  einen *Teiler* von  $p(x)$  und schreiben  $q(x) \mid p(x)$ .

Das letzte Beispiel in Abbildung 2.34 hat gezeigt, dass  $x^2 + 1$  in  $\mathbb{Z}_2$  ein Teiler von  $x^3 + x^2 + x + 1$  ist. Ferner hat dieses Polynom den Teiler  $x + 1$  sowie die beiden *trivialen Teiler*  $x^3 + x^2 + x + 1$  und 1.

Polynomdivision in  $\mathbb{Q}[x]$ 

■ Beispiel 1:  $(x^6 + 1) : (x^5 + x^2 + x + 1)$

$$\begin{array}{r} x^6 \\ -x^6 - x^3 - x^2 - x \\ \hline -x^3 - x^2 - x + 1 \end{array} \quad \begin{array}{l} +1 = (x^5 + x^2 + x + 1) \cdot x \\ + (-x^3 - x^2 - x + 1) \end{array}$$

■ Beispiel 2:  $(x^5 + x^2 + x + 1) : (x^3 + x^2 + x - 1)$

$$\begin{array}{r} x^5 \\ -x^5 - x^4 - x^3 + x^2 \\ \hline -x^4 - x^3 + 2x^2 + x \\ x^4 + x^3 + x^2 - x \\ \hline 3x^2 \quad +1 \end{array} \quad \begin{array}{l} + x^2 + x + 1 = (x^3 + x^2 + x - 1) \cdot (x^2 - x) \\ +(3x^2 + 1) \end{array}$$

■ Beispiel 3:  $(x^3 + x^2 + x - 1) : (x^2 + \frac{1}{3})$

$$\begin{array}{r} x^3 + x^2 + x - 1 \\ -x^3 \quad -\frac{1}{3}x \\ \hline x^2 + \frac{2}{3}x - 1 \\ -x^2 \quad -\frac{1}{3} \\ \hline \frac{2}{3}x - \frac{4}{3} \end{array}$$

■ Beispiel 4:  $(x^2 + \frac{1}{3}) : (x - 2)$

$$\begin{array}{r} x^2 \quad +\frac{1}{3} \\ -x^2 + 2x \\ \hline 2x + \frac{1}{3} \\ -2x + 4 \\ \hline \frac{13}{3} \end{array} \quad \begin{array}{l} = (x - 2) \cdot (x + 2) + \frac{13}{3} \end{array}$$

Polynomdivision in  $\mathbb{Z}_2[x]$ 

■ Beispiel 1:  $(x^6 + 1) : (x^5 + x^2 + x + 1)$

$$\begin{array}{r} x^6 \\ -x^6 - x^3 - x^2 - x \\ \hline x^3 + x^2 + x + 1 \end{array} \quad \begin{array}{l} +1 = (x^5 + x^2 + x + 1) \cdot x + (x^3 + x^2 + x + 1) \\ + (-x^3 - x^2 - x + 1) \end{array}$$

■ Beispiel 2:  $(x^5 + x^2 + x + 1) : (x^3 + x^2 + x + 1)$

$$\begin{array}{r} x^5 \\ -x^5 - x^4 - x^3 - x^2 \\ \hline x^4 + x^3 \quad +x \\ -x^4 - x^3 - x^2 - x \\ \hline x^2 \quad +1 \end{array} \quad \begin{array}{l} + x^2 + x + 1 = (x^3 + x^2 + x + 1) \cdot (x^2 + x) \\ +(x^2 + 1) \end{array}$$

■ Beispiel 3:  $(x^3 + x^2 + x + 1) : (x^2 + 1)$

$$\begin{array}{r} x^3 + x^2 + x + 1 \\ -x^3 \quad -x \\ \hline x^2 \quad +1 \\ -x^2 \quad -1 \\ \hline 0 \end{array}$$

**Abb. 2.34:** Die Division von Polynomen funktioniert nach dem gleichen Prinzip wie die Division ganzer Zahlen. In mehreren Schritten wird der Dividend  $p(x)$  durch die Subtraktion eines Vielfachen von  $q(x)$  verringert. Der Faktor wird dabei so gewählt, dass der führende Summand des Dividenden verschwindet.

Sobald der Grad des Dividenden kleiner ist als der Grad von  $q(x)$ , kann kein passendes Vielfaches mehr gefunden werden, und das Verfahren terminiert. Der nun erreichte Wert des Dividenden ist der Divisionsrest  $r(x)$ .

In  $\mathbb{Z}_2$  ist die Polynomdivision besonders einfach. Hier gibt es nur zwei mögliche Koeffizienten (0 und 1), und zwischen der Subtraktion und der Addition eines Terms der Form  $x^i$  besteht wegen der Modulo-2-Rechnung kein Unterschied.

### Polynomzerlegung in $\mathbb{Z}_2[x]$

- Faktorisierung von  $x^3 + x^2 + x + 1$

$$\begin{aligned} & x^3 + x^2 + x + 1 \\ &= (x^2 + 1)(x + 1) \\ &= (x + 1)(x + 1)(x + 1) \end{aligned}$$

- Teiler sind ...

$$1, x + 1, x^2 + 1, x^3 + x^2 + x + 1$$

- Normierte Teiler sind ...

$$1, x + 1, x^2 + 1, x^3 + x^2 + x + 1$$

### Polynomzerlegung in $\mathbb{Q}[x]$

- Faktorisierung von  $x^3 + x^2 + x + 1$

$$\begin{aligned} & x^3 + x^2 + x + 1 \\ &= (x^2 + 1)(x + 1) \end{aligned}$$

- Teiler sind ...

$$\begin{aligned} & k, k(x + 1), k(x^2 + 1), k(x^3 + x^2 + x + 1) \\ & k \in \mathbb{Q} \setminus \{0\} \end{aligned}$$

- Normierte Teiler sind ...

$$1, x + 1, x^2 + 1, x^3 + x^2 + x + 1$$

**Abb. 2.35:** Faktorisierung von Polynomen.

Ist  $\mathbb{K}$  ein unendlicher Körper, so besitzen die Polynome aus  $\mathbb{K}[x]$  unendlich viele Teiler. Wir erhalten Sie durch die Multiplikation der normierten Teiler mit einem Körperelement. In  $\mathbb{Z}_2$  gibt es nur die Koeffizienten 0 und 1. Deshalb ist im Polynomring  $\mathbb{Z}_2[x]$  jeder Teiler immer auch ein normierter Teiler und umgekehrt.

Gibt es neben den gefundenen Teilern noch weitere? Die Antwort lautet Nein. Da wir in diesem Beispiel in  $\mathbb{Z}_2[x]$  rechnen, können wir  $x^2 + 1$  zwar noch weiter in  $(x + 1)(x + 1)$  zerlegen, aber den Faktor  $x + 1$  haben wir bereits. Insgesamt erhalten wir die 4 in Abbildung 2.35 (oben) zusammengefassten Teiler.

Ein anderes Ergebnis entsteht, wenn wir  $x^3 + x^2 + x + 1$  als ein Polynom aus der Menge  $\mathbb{Q}[x]$  oder  $\mathbb{R}[x]$  auffassen. In diesem Fall existieren unendlich viele Teiler, da wir aus jedem Teiler einen neuen generieren können, indem wir ihn mit einer von 0 verschiedenen rationalen oder reellen Zahl multiplizieren. Aus diesem Grund werden zumeist nur die sogenannten *normierten Teiler* bestimmt, deren führende Koeffizienten gleich 1 sind. Die Anzahl der normierten Teiler ist stets endlich, und für unser Beispieldpolynom erhalten wir in  $\mathbb{Q}[x]$  die gleichen normierten Teiler wie in  $\mathbb{Z}_2[x]$  (Abbildung 2.35 unten). Dass dies nicht immer der Fall ist, beweist z. B. das Polynom  $x^2 + 1$ . In  $\mathbb{Z}_2[x]$  besitzt es die normierten Teiler 1,  $x + 1$  und  $x^2 + 1$ . In  $\mathbb{Q}[x]$  sind die einzigen normierten Teiler die Polynome 1 und  $x^2 + 1$ .

Polynome, die neben den trivialen Teilern keine weiteren besitzen, sind von so großer Bedeutung, dass sie einen eigenen Namen tragen:



#### Definition 2.11 (Irreduzibles Polynom)

Ein Polynom  $p(x)$  heißt *irreduzibel*, wenn es sich nicht als Produkt von Polynomen niedrigerer Grade schreiben lässt.

Irreduzible Polynome lassen sich nicht zerlegen und spielen im Ring der Polynome damit die gleiche Rolle wie die Primzahlen im Ring der ganzen Zahlen. Tatsächlich gehen die Analogien zwischen den Polynomen und den ganzen Zahlen noch weiter. Etliche Eigenschaften, die wir über die Menge  $\mathbb{Z}$  herausgearbeitet haben, beruhen gar nicht auf der speziellen Beschaffenheit der ganzen Zahlen, sondern lediglich auf der Tatsache, dass die Ringaxiome erfüllt werden. Diese Eigenschaften gelten dann aber in jedem Ring und damit auch in  $\mathbb{K}[x]$ .

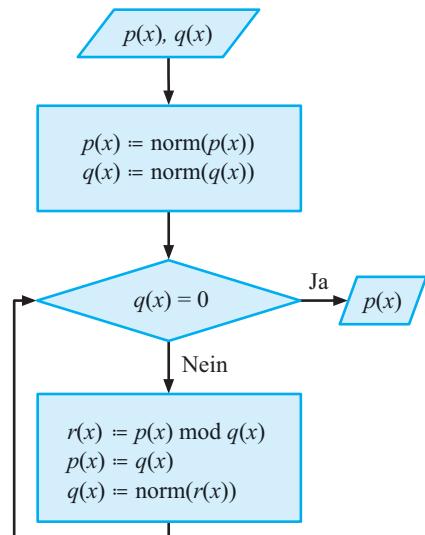
Zu den Ergebnissen, die wir eins zu eins auf Polynome übertragen können, gehört die Berechnung des größten gemeinsamen Teilers. Das Flussdiagramm aus Abbildung 2.37 zeigt, dass wir den euklidischen Algorithmus für Polynome wie gewohnt durchführen können, indem wir die beiden Operanden so lange wechselseitig dividieren, bis das Null-element erreicht wird. Der letzte von 0 verschiedene Divisionsrest ist dann der größte gemeinsame Teiler.

ggT-Berechnung in $\mathbb{Q}[x]$	ggT-Berechnung in $\mathbb{Z}_2[x]$
$\text{ggT}(x^6 + 1, x^5 + x^2 + x + 1)$  $x^6 + 1 = (x^5 + x^2 + x + 1) \cdot x + (-x^3 - x^2 - x + 1)$ $\text{ggT}(x^5 + x^2 + x + 1, x^3 + x^2 + x - 1)$  $x^5 + x^2 + x + 1 = (x^3 + x^2 + x - 1) \cdot (x^2 - x) + (3x^2 + 1)$ $\text{ggT}(x^3 + x^2 + x - 1, x^2 + \frac{1}{3})$  $x^3 + x^2 + x - 1 = (x^2 + \frac{1}{3})(x + 1) + (\frac{2}{3}x - \frac{4}{3})$ $\text{ggT}(x^2 + \frac{1}{3}, x - 2)$  $x^2 + \frac{1}{3} = (x - 2)(x + 2) + \frac{13}{3}$ $\text{ggT}(x - 2, 1)$  $x - 2 = 1 \cdot (x - 2) + 0$ $\text{ggT}(1, 0) = 1$	$\text{ggT}(x^6 + 1, x^5 + x^2 + x + 1)$  $x^6 + 1 = (x^5 + x^2 + x + 1) \cdot x + (x^3 + x^2 + x + 1)$ $\text{ggT}(x^5 + x^2 + x + 1, x^3 + x^2 + x + 1)$  $x^5 + x^2 + x + 1 = (x^3 + x^2 + x + 1) \cdot (x^2 + x) + (x^2 + 1)$ $\text{ggT}(x^3 + x^2 + x + 1, x^2 + 1)$  $x^3 + x^2 + x + 1 = (x^2 + 1)(x + 1) + 0$ $\text{ggT}(x^2 + 1, 0) = x^2 + 1$

**Abb. 2.36:** Berechnung des größten gemeinsamen Teilers zweier Polynome. In jedem Berechnungsschritt wird der erste durch den zweiten Operanden geteilt und der Rest bestimmt. Danach wird mit dem zweiten Operanden und dem normierten Rest weiter gerechnet. Das Verfahren bricht ab, sobald die beiden Polynome ohne Rest teilbar sind.

Abbildung 2.36 demonstriert den Ablauf am Beispiel der Polynome  $x^6 + 1$  und  $x^5 + x^2 + x + 1$ . Die Berechnung wird in zwei verschiedenen Polynomringen durchgeführt, links im Ring  $\mathbb{Q}[x]$  und rechts im Ring  $\mathbb{Z}_2[x]$ . In  $\mathbb{Q}[x]$  erhalten wir das Ergebnis 1, d. h., die Polynome sind in diesem Ring teilerfremd. In  $\mathbb{Z}_2[x]$  terminiert der Algorithmus früher und liefert als größten gemeinsamen Teiler das Polynom  $x^2 + 1$  zurück. Die Beispiele machen deutlich, dass die Irreduzibilität eines Polynoms keine absolute Eigenschaft ist; sie hängt davon ab, über welchem Körper wir die Polynomkoeffizienten interpretieren.

Die Beispiele zeigen aber noch etwas anderes: Auch wenn der euklidische Algorithmus im Ring der Polynome genauso funktioniert wie im Ring der ganzen Zahlen, ist er deutlich arbeitsintensiver. Dies liegt daran, dass wir in jedem Einzelschritt eine aufwendige Polynomdivision durchführen müssen, um den Divisionsrest zu erhalten. Glücklicherweise konnten wir für die Berechnung in Abbildung 2.36 auf Bekanntes zurückgreifen. Die Divisionen, die dort durchgeführt werden müssen, sind jene, mit denen wir in Abbildung 2.34 die allgemeine Arbeitsweise der Polynomdivision verdeutlicht haben.



**Abb. 2.37:** Der euklidische Algorithmus für Polynome

## 2.4.2 Konstruktion endlicher Körper

Mithilfe der Polynomdivision können wir die Addition und die Multiplikation in  $\mathbb{K}[x]$  modulo eines Polynoms durchführen, und damit in  $\mathbb{K}[x]$  auf ganz ähnliche Weise rechnen, wie wir es in der Menge  $\mathbb{Z}_m$  getan haben.

Für ein Polynom  $m(x)$  aus der Menge  $\mathbb{K}[x]$  definieren wir  $\mathbb{K}[x]_{m(x)}$  als die Menge der Reste, die bei der Division eines Polynoms aus  $\mathbb{K}[x]$  durch  $m(x)$  entstehen können.

$$\mathbb{K}[x]_{m(x)} := \{a_n x^n + \dots + a_1 x + a_0 \mid a_i \in \mathbb{K}, n < \deg m(x)\}$$

Beachten Sie, dass die Elemente von  $\mathbb{K}[x]_{m(x)}$  ausschließlich durch den Grad von  $m(x)$  bestimmt werden; die konkreten Werte der Koeffizienten spielen keine Rolle.

Wir wollen nun untersuchen, wie sich die Addition und die Multiplikation auf der Menge  $\mathbb{K}[x]_{m(x)}$  verhält. Zunächst ist klar, dass die Summe zweier Polynome aus  $\mathbb{K}[x]_{m(x)}$  wieder in  $\mathbb{K}[x]_{m(x)}$  liegt und die Polynomaddition damit abgeschlossen ist. Die Multiplikation ist auf der Menge  $\mathbb{K}[x]_{m(x)}$  ebenfalls abgeschlossen, wenn wir das Produkt  $p(x) \cdot q(x)$  durch den Modul  $m(x)$  teilen und als Ergebnis den Divisionsrest behalten. Anders als bei der Addition erhalten wir bei der Multiplikation für verschiedene Polynome  $m(x)$  verschiedene Ergebnisse. Die folgenden beiden Rechnungen demonstrieren dies am Beispiel der Polynome  $(x+1)$  und  $x$ , einmal multipliziert in  $\mathbb{Z}_2[x]_{x^2}$  und ein anderes Mal in  $\mathbb{Z}_2[x]_{x^2+x}$ :

- Multipliziert in  $\mathbb{Z}_2[x]_{x^2}$ :  $(x+1) \cdot (x) = x^2 + x \bmod x^2 = \textcolor{blue}{x}$
- Multipliziert in  $\mathbb{Z}_2[x]_{x^2+x}$ :  $(x+1) \cdot (x) = x^2 + x \bmod x^2 + x = \textcolor{blue}{0}$

Das erste Ergebnis erhalten wir aufgrund der Zerlegung

$$x^2 + x = x^2 \cdot 1 + \textcolor{blue}{x}$$

und das zweite aufgrund der Zerlegung

$$x^2 + x = (x^2 + x) \cdot 1 + \textcolor{blue}{0}$$

Zusammen mit der vereinbarten Addition und der vereinbarten Multiplikation bildet die Menge  $\mathbb{K}[x]_{m(x)}$  einen kommutativen Ring mit 1. Wir bezeichnen ihn als den *Restklassenring* modulo  $m(x)$ .

■ Addition in  $\mathbb{Z}_2[x]_{m(x)}$  mit  $\deg m(x) = 2$

+	0	1	$x$	$x+1$
0	0	1	$x$	$x+1$
1	1	0	$x+1$	$x$
$x$	$x$	$x+1$	0	1
$x+1$	$x+1$	$x$	1	0

+	00	01	10	11
00	00	01	10	11
01	01	00	11	10
10	10	11	00	01
11	11	10	01	00

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

■ Multiplikation in  $\mathbb{Z}_2[x]_{x^2}$

.	0	1	$x$	$x+1$
0	0	0	0	0
1	0	1	$x$	$x+1$
$x$	0	$x$	0	$x$
$x+1$	0	$x+1$	$x$	1

.	00	01	10	11
00	00	00	00	00
01	00	01	10	11
10	00	10	00	10
11	00	11	10	01

.	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

■ Multiplikation in  $\mathbb{Z}_2[x]_{x^2+1}$

.	0	1	$x$	$x+1$
0	0	0	0	0
1	0	1	$x$	$x+1$
$x$	0	$x$	$x$	0
$x+1$	0	$x+1$	$x+1$	0

.	00	01	10	11
00	00	00	00	00
01	00	01	10	11
10	00	10	01	11
11	00	11	11	00

.	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	1	3
3	0	3	3	0

■ Multiplikation in  $\mathbb{Z}_2[x]_{x^2+x}$

.	0	1	$x$	$x+1$
0	0	0	0	0
1	0	1	$x$	$x+1$
$x$	0	$x$	$x$	0
$x+1$	0	$x+1$	0	$x+1$

.	00	01	10	11
00	00	00	00	00
01	00	01	10	11
10	00	10	10	00
11	00	11	01	11

.	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	2	0
3	0	3	3	0

■ Multiplikation in  $\mathbb{Z}_2[x]_{x^2+x+1}$

.	0	1	$x$	$x+1$
0	0	0	0	0
1	0	1	$x$	$x+1$
$x$	0	$x$	$x+1$	1
$x+1$	0	$x+1$	1	$x$

.	00	01	10	11
00	00	00	00	00
01	00	01	10	11
10	00	10	11	01
11	00	11	01	10

.	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

Abb. 2.38: Von den vier gezeigten Polynomen erzeugt nur  $x^2 + x + 1$  die algebraische Struktur eines Körpers.

Abbildung 2.38 fasst die Verknüpfungstabellen aller Restklassenringe  $\mathbb{Z}_2[x]_{m(x)}$  zusammen, die mit den Polynomen  $m(x)$  vom Grad 2 gebildet werden können. Jede Tabelle ist in drei verschiedenen Notationen aufgeführt. Links sind die Polynome in der üblichen mathematischen Schreibweise vermerkt. In der Mitte und rechts sind sie durch ihre Koeffizientenvektoren repräsentiert, einmal in binärer und einmal in dezimaler Notation. Indem wir jedes Polynom eindeutig mit einer natürlichen Zahl assoziieren, gelangen wir zu Verknüpfungstabellen, die jetzt auch optisch an jene erinnern, die wir im Zusammenhang mit den natürlichen Zahlen kennen gelernt haben.

Die Tabellen, die wir für die Polynome  $x^2$  und  $x^2 + x + 1$  erhalten, sind alte Bekannte. Das Polynom  $x^2$  generiert die Multiplikationstabelle aus Abbildung 2.21 (unten). Das bedeutet, dass  $(\mathbb{Z}_2[x]_{x^2}, \cdot)$  und  $(\mathbb{Z}_4, \cdot)$  isomorphe Strukturen sind, sich also nur durch die Benennung ihrer Elemente unterscheiden. Noch interessanter ist der Restklassenring, den wir für das Polynom  $x^2 + x + 1$  erhalten. Die entstandenen Verknüpfungstabellen entsprechen jenen aus Abbildung 2.22 und bilden die Struktur eines Körpers. Als wir die Tabellen in Abbildung 2.22 einführten, hatten wir dies ohne Begründung getan. Sie fielen gewissermaßen vom Himmel, und wir hatten uns lediglich nachträglich davon überzeugt, dass wir einen Körper vor uns haben. Dies ist jetzt anders. Offenbar haben wir es geschafft, über die Polynomringe einen systematischen Zugang zu den endlichen Körpern zu erhalten.

Mit dem jetzigen Wissensstand wollen wir uns noch nicht zufrieden geben und einen zweiten Blick auf die Verknüpfungstabellen in Abbildung 2.38 werfen. Dieser zeigt, dass  $x^2 + x + 1$  das einzige Polynom ist, das die Struktur eines Körpers hervorbringt. Aber warum führt ausgegerechnet dieses Polynom zu einem Körper und die anderen nicht?

Um dieser Frage auf den Grund zu gehen, übertragen wir zunächst einen wichtigen Begriff auf Polynome, der uns von den ganzen Zahlen her geläufig ist:



### Definition 2.12 (Kongruenz von Polynomen)

Zwei Polynome  $p(x), q(x) \in \mathbb{K}[x]$  heißen *kongruent modulo  $m(x)$* , geschrieben als

$$p(x) \equiv q(x) \pmod{m(x)},$$

wenn ihre Division durch den Modul  $m(x)$  denselben Rest ergibt.

Demnach sind zwei Polynome kongruent, wenn sie sich um ein Vielfaches des Moduls  $m(x)$  voneinander unterscheiden:

$$p(x) \equiv q(x) \pmod{m(x)} \Leftrightarrow \begin{aligned} p(x) - q(x) &= k(x) \cdot m(x) \\ \text{für ein } k(x) &\in \mathbb{K}[x] \end{aligned} \quad (2.13)$$

Das Pendant zu (2.13) ist die Gleichung (2.5) aus Abschnitt 2.2, wo wir den gleichen Zusammenhang bei den ganzen Zahlen beobachten konnten. Direkt im Anschluss daran hatten wir uns mit der Lösung der Gleichungen  $x+y \equiv 0 \pmod{m}$  und  $x \cdot y \equiv 1 \pmod{m}$  befasst, d. h. mit der Frage, ob für eine Zahl  $x$  aus der Menge  $\mathbb{Z}_m$  ein additives bzw. ein multiplikatives Inverses existiert. Stellen wir die gleiche Frage für die Elemente aus  $\mathbb{K}[x]_{m(x)}$ , so kommen wir zu analogen Ergebnissen:

- Für jedes vorgelegte Polynom  $p(x) \in \mathbb{K}[x]_{m(x)}$  hat die Gleichung

$$p(x) + q(x) \equiv 0 \pmod{m(x)} \quad (2.14)$$

eine Lösung  $q(x) \in \mathbb{K}[x]_{m(x)}$ . Die Lösung ist eindeutig bestimmt und leicht zu benennen. Wir erhalten sie, indem wir in  $p(x)$  die Koeffizienten  $a_n, \dots, a_0$  durch ihre additiven Inversen  $-a_n, \dots, -a_0$  ersetzen.

- Für ein vorgelegtes Polynom  $p(x) \in \mathbb{K}[x]_{m(x)}$  ist die Gleichung

$$p(x) \cdot q(x) \equiv 1 \pmod{m(x)} \quad (2.15)$$

nur manchmal lösbar. Betrachten wir beispielsweise das Polynom  $x$  in  $\mathbb{Z}_2[x]_{x^2}$ , so erhalten wir als Ergebnis einer Multiplikation entweder  $x$  oder 0, aber niemals die 1. Kurzum: Für  $p(x) = x$  hat die Gleichung (2.15) in  $\mathbb{Z}_2[x]_{x^2}$  keine Lösung. Dagegen besitzt das Polynom  $(x+1)$  in  $\mathbb{Z}_2[x]_{x^2}$  sehr wohl ein multiplikatives Inverses. Wir können es ganz einfach durch die Multiplikation mit sich selbst auf die 1 zurückführen:

$$(x+1) \cdot (x+1) = x^2 + 1 \pmod{x^2} = 1$$

Anders ist die Situation in  $\mathbb{Z}_2[x]_{x^2+x+1}$ . Hier besitzt ausnahmslos jedes Polynom ein multiplikatives Inverses, und genau diese Eigenschaft ist es, die  $\mathbb{Z}_2[x]_{x^2+x+1}$  zu einem Körper macht.

Wir stehen hier vor der exakt gleichen Situation wie in Abschnitt 2.2. Dort hatten wir herausgefunden, dass die Menge  $\mathbb{Z}_m$  genau dann ein Körper ist, wenn wir für  $m$  eine Primzahl wählen. Ein zweiter Blick auf die Argumentation, die zu diesem Ergebnis führte, macht klar, dass wir ausschließlich auf die Eigenschaft von  $\mathbb{Z}_m$  zurückgegriffen haben, ein Ring zu sein. Das bedeutet, dass wir die Argumentationskette eins zu eins auf den Polynomring  $\mathbb{K}[x]_{m(x)}$  übertragen können und ein zu Satz 2.1 analoges Ergebnis erhalten:


**Satz 2.8**

Für jedes Polynom  $p(x) \in \mathbb{K}[x]_{m(x)}$  gilt:

$p(x) \cdot q(x) \equiv 1 \pmod{m(x)}$  hat eine Lösung  $\Leftrightarrow \text{ggT}(p(x), m(x)) = 1$

Im Falle einer Lösung ist  $q(x)$  eindeutig bestimmt.

Das bedeutet, dass der Polynomring  $\mathbb{K}[x]_{m(x)}$  genau dann zu einem Körper wird, wenn der Modul  $m(x)$  irreduzibel ist. Jetzt ist auch klar, warum von den vier Beispieldaten aus Abbildung 2.38 nur eines einen Körper hervorbringt: Die drei anderen sind in  $\mathbb{Z}_2[x]$  allesamt reduzibel:

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

Dass uns das Polynom  $x^2 + x + 1$  einen Körper mit genau 4 Elementen liefert, ist ebenfalls leicht einzusehen: In  $\mathbb{Z}_2[x]$  existieren genau 4 Polynome mit einem Grad kleiner als 2.

Diese Überlegung lässt sich mühelos verallgemeinern: Ist  $m(x)$  irreduzibel und  $\deg m(x) = n$ , so ist ein Polynom genau dann in der Menge  $\mathbb{Z}_p[x]_{m(x)}$  enthalten, wenn sein Grad kleiner als  $n$  ist und seine Koeffizienten aus  $\mathbb{Z}_p$  stammen. Ein solches Polynom wird durch einen Koeffizientenvektor mit  $n$  Einträgen definiert, und für jeden Eintrag kommen die Zahlen  $0, \dots, p-1$  in Frage. Das bedeutet, dass jedes irreduzible Polynom aus der Menge  $\mathbb{Z}_p[x]$  vom Grad  $n$  einen endlichen Körper mit  $p^n$  Elementen erzeugt.

Für  $p = 2$  und  $n = 8$  erhalten wir einen endlichen Körper, der in der Informatik eine besondere Rolle spielt. Er umfasst 256 Elemente und deckt damit exakt den Wertebereich eines Bytes ab. Um diesen Körper systematisch zu konstruieren, wird ein irreduzibles Polynom  $m(x)$  aus der Menge  $\mathbb{Z}_2[x]$  vom Grad 8 benötigt. Untersuchen wir alle in Frage kommenden Polynome der Reihe nach, so werden wir bei

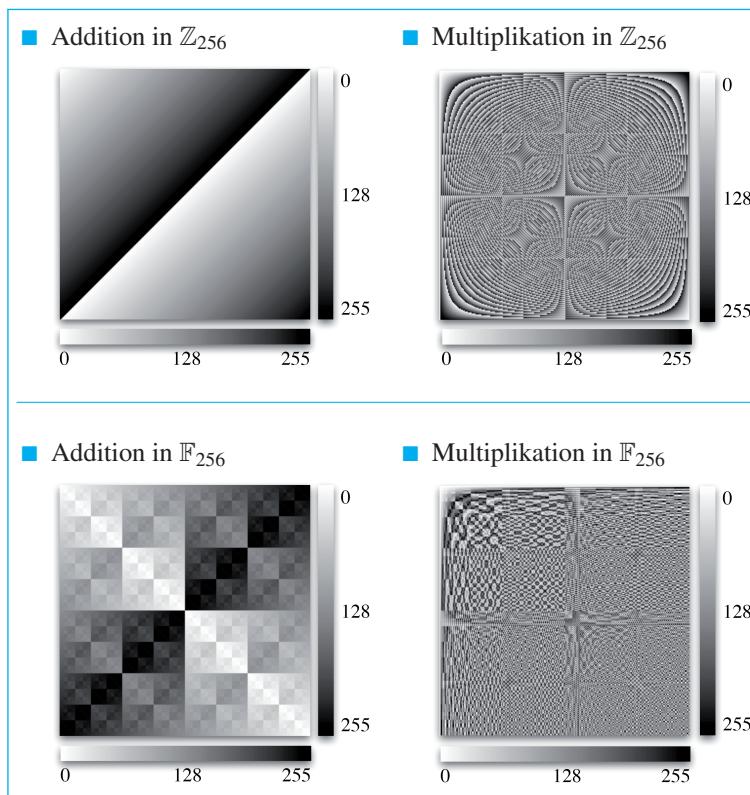
$$x^8 + x^4 + x^3 + x + 1$$

das erste Mal fündig. Das Polynom ist irreduzibel und führt damit zu einem Körper mit 256 Elementen.

Aber wie sieht dieser Körper im Detail aus? Bisher hatten wir uns einen Einblick in die Struktur eines endlichen Körpers stets durch das Aufstellen der Verknüpfungstabellen für  $, +'$  und  $, \cdot'$  verschafft. Würden

$x^8 + x^4 + x^3 + x + 1$ , das Polynom, auf dem die Verknüpfungstabellen in Abbildung 2.39 beruhen, ist ein prominentes. Es wird im bekannten Rijndael-Algorithmus verwendet, der nach den belgischen Kryptologen Vincent Rijmen und Joan Daemen benannt ist und die Grundlage für den *Advanced Encryption Standard*, kurz AES, bildet. Der Algorithmus wurde durch das National Institute of Standards and Technology (NIST) im Jahr 2001 als Nachfolger des *Data Encryption Standard*, kurz DES, verabschiedet und zählt zu den sichersten Verschlüsselungsalgorithmen, die wir heute kennen. Eine verständliche Einführung in das Gebiet der Kryptografie im Allgemeinen und den Rijndael-Algorithmus im Speziellen finden Sie in [28].





**Abb. 2.39:** In den nebenstehenden Bildern sind Zahlen durch Grauwerte dargestellt: 0 entspricht einem weißen und 255 einem schwarzen Pixel.

Oben: Die Bilder visualisieren die Addition und die Multiplikation auf der Menge  $\mathbb{Z}_{256}$ . Hier werden zwei Zahlen zunächst wie gewöhnlich miteinander verrechnet, danach wird das Ergebnis über die Modulo-Operation reduziert. Die Menge  $\mathbb{Z}_{256}$  bildet mit diesen Operationen keinen Körper, da nicht jede Zahl ein multiplikatives Inverses besitzt.

Unten: Die Abbildungen zeigen, wie die Addition und die Multiplikation umdefiniert werden müssen, um den Körperaxiomen zu genügen.

wir dies auch hier tun, so entstünden zwei unübersichtliche Tabellen mit jeweils 65.536 Elementen. Um der Zahlenmenge Herr zu werden, wollen wir versuchen, die Verknüpfungsstrukturen visuell zu erfassen. Dies gelingt am einfachsten, wenn wir die Zahlen von 0 bis 255 auf Grauwerte abbilden und das Ergebnis einer Addition und Multiplikation als hellere oder dunklere Punkte darstellen. Auf diese Weise wird die Verknüpfungstabelle zu einem 256 Pixel breiten und 256 Pixel hohen Bild.

Abbildung 2.39 zeigt vier Verknüpfungstabellen, die auf die beschriebene Weise aufbereitet wurden. Die beiden oberen Bilder visualisieren die gewöhnliche Addition und die Multiplikation auf der Menge  $\mathbb{Z}_{256}$ . Die unteren Bilder zeigen, wie wir die Addition und Multiplikation umdefinieren müssen, um die algebraische Struktur eines Körpers zu erhalten.

Seien Sie sich stets darüber bewusst, dass die Wahl des irreduziblen Polynoms  $m(x)$  ausschließlich die Multiplikation beeinflusst. Da die Addition in  $\mathbb{Z}_2[x]_{m(x)}$  der XOR-Verknüpfung der Koeffizienten entspricht, können niemals Polynome höherer Grade entstehen, und damit müssen wir auch niemals das Ergebnis mit der Modulo-Operation reduzieren. Bei der Multiplikation ist dies anders. Hier ist es nötig, das Produkt zweier Polynome nachträglich durch  $m(x)$  zu dividieren, sodass die Wahl des irreduziblen Polynoms einen direkten Einfluss auf das Ergebnis hat.

Nach dem bisher Gesagten ist klar, dass wir mit jedem irreduziblen Polynom aus  $\mathbb{Z}_2[x]$  vom Grad 8 einen endlichen Körper konstruieren können und davon existieren insgesamt 30. Die Abbildungen 2.40 und 2.41 zeigen alle 30 Polynome, zusammen mit den resultierenden Multiplikationstabellen.

Auch wenn die Körper für einzelne Polynome ein jeweils anderes Ergebnis liefern, scheinen sie aus etwas Abstand betrachtet ein Abbild der gleichen abstrakten Struktur zu sein. Warum dies so sein muss, hatten wir weiter oben bereits vorweggenommen. Satz 2.4 besagte, dass endliche Körper gleicher Größe zueinander isomorph sind, sich also lediglich in der Benennung ihrer Symbole unterscheiden. Damit ist klar, dass es nur eine untergeordnete Rolle spielt, welches irreduzible Polynom wir für die Körperkonstruktion am Ende verwenden. Wir erhalten jedes Mal eine isomorphe Struktur.

Zum Schluss wollen wir einen Satz beweisen, den wir in Kapitel 6 auf Seite 431 benötigen. Er macht eine Aussage über die Summen, die wir erhalten, wenn die Elemente eines endlichen Körpers bzw. Potenzen davon aufaddiert werden:



### Satz 2.9

Sei  $\mathbb{F}$  ein Körper mit  $n$  Elementen und  $k \in \{0, \dots, n-2\}$ . Dann gilt:

$$\sum_{\alpha \in \mathbb{F}} \alpha^k = 0$$

*Beweis:* Ist  $\gamma$  ein Körperelement, so sei  $\gamma\mathbb{F}$  die Menge

$$\gamma\mathbb{F} := \{\gamma\alpha \mid \alpha \in \mathbb{F}\}.$$

Ist  $\gamma$  nicht das Nullelement, so ist die Abbildung  $\alpha \mapsto \gamma\alpha$  eine Bijektion auf  $\mathbb{F}$ . Das bedeutet, dass die sukzessive Multiplikation von  $\gamma$  mit allen

Elementen aus  $\mathbb{F}$  erneut die Menge  $\mathbb{F}$  hervorbringt; es ist also  $\mathbb{F} = \gamma\mathbb{F}$ . Damit können wir die folgende Umformung vornehmen:

$$\sum_{\alpha \in \mathbb{F}} \alpha^k = \sum_{\alpha \in \gamma\mathbb{F}} \alpha^k = \sum_{\alpha \in \mathbb{F}} (\gamma\alpha)^k = \gamma^k \sum_{\alpha \in \mathbb{F}} \alpha^k$$

Daraus folgt, dass  $\sum_{\alpha \in \mathbb{F}} \alpha^k$  entweder gleich 0 ist oder jedes Element  $\gamma \in \mathbb{F} \setminus \{0\}$  die Beziehung  $\gamma^k = 1$  erfüllt, also eine Nullstelle des Polynoms  $x^k - 1$  ist. Letzteres ist aber unmöglich. Wenn ein Polynom  $n - 1$  verschiedene Nullstellen hat, muss sein Grad mindestens  $n - 1$  betragen. Folglich müsste das Polynom  $x^k - 1$  mindestens den Grad  $n - 1$  aufweisen, und dies steht im Widerspruch zu  $k \leq n - 2$ .  $\square$

### 2.4.3 Schnelles Rechnen in endlichen Körpern

Die vorangegangenen Abschnitte haben uns gezeigt, wie endliche Körper systematisch konstruiert werden können und wie sich die Elemente eines endlichen Körpers addieren und multiplizieren lassen. Eines unserer Kernergebnisse besagte, dass für jede Primzahl  $p$  und jede natürliche Zahl  $n \geq 1$  ein endlicher Körper mit  $p^n$  Elementen existiert, der bis auf Isomorphie eindeutig bestimmt ist. Den Fall  $n = 1$  hatten wir bereits in Abschnitt 2.3.2 mit den Körpern  $(\mathbb{Z}_p, +, \cdot)$  erledigt, und dort war das Rechnen ausgesprochen leicht: Zahlen werden ganz normal addiert oder multipliziert und das Ergebnis anschließend modulo  $p$  gerechnet. Für  $n \geq 2$  ist die Situation nicht mehr ganz so einfach. Um das Produkt zweier Elemente aus  $\mathbb{F}_{p^n}$  zu bilden, müssen wir zunächst eine Polynommultiplikation durchführen und anschließend den Rest bestimmen, der bei einer Polynomdivision durch den Modul entsteht. Der Mehraufwand ist beträchtlich und lässt das Rechnen in  $\mathbb{F}_{p^n}$  für  $n \geq 2$  auf den ersten Blick als schwierig erscheinen.

Wir wollen in diesem Abschnitt nach Möglichkeiten suchen, um die Multiplikation in endlichen Körpern zu beschleunigen. Wir beschränken uns dabei auf den endlichen Körper  $\mathbb{F}_{2^8}$ , da dieser die größte Praxisbedeutung besitzt. In Abschnitt 2.4.2 hatten wir gezeigt, dass wir die Elemente von  $\mathbb{F}_{2^8}$  mit den Polynomen aus der Menge  $\mathbb{Z}_2[x]_{m(x)}$  gleichsetzen dürfen, wenn wir für  $m(x)$  ein beliebiges irreduzibles Polynom aus  $\mathbb{Z}_2[x]$  vom Grad 8 wählen. Jedes Polynom aus  $\mathbb{Z}_2[x]_{m(x)}$  hat dann die Form

$$a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$$

mit  $a_i \in \{0, 1\}$ . Ein solches Polynom können wir eindeutig mit einer Zahl aus dem Intervall  $[0; 255]$  assoziieren, indem wir uns die Koeffizienten  $a_7a_6a_5a_4a_3a_2a_1a_0$  als die Ziffern einer 8 Bit breiten Binärzahl

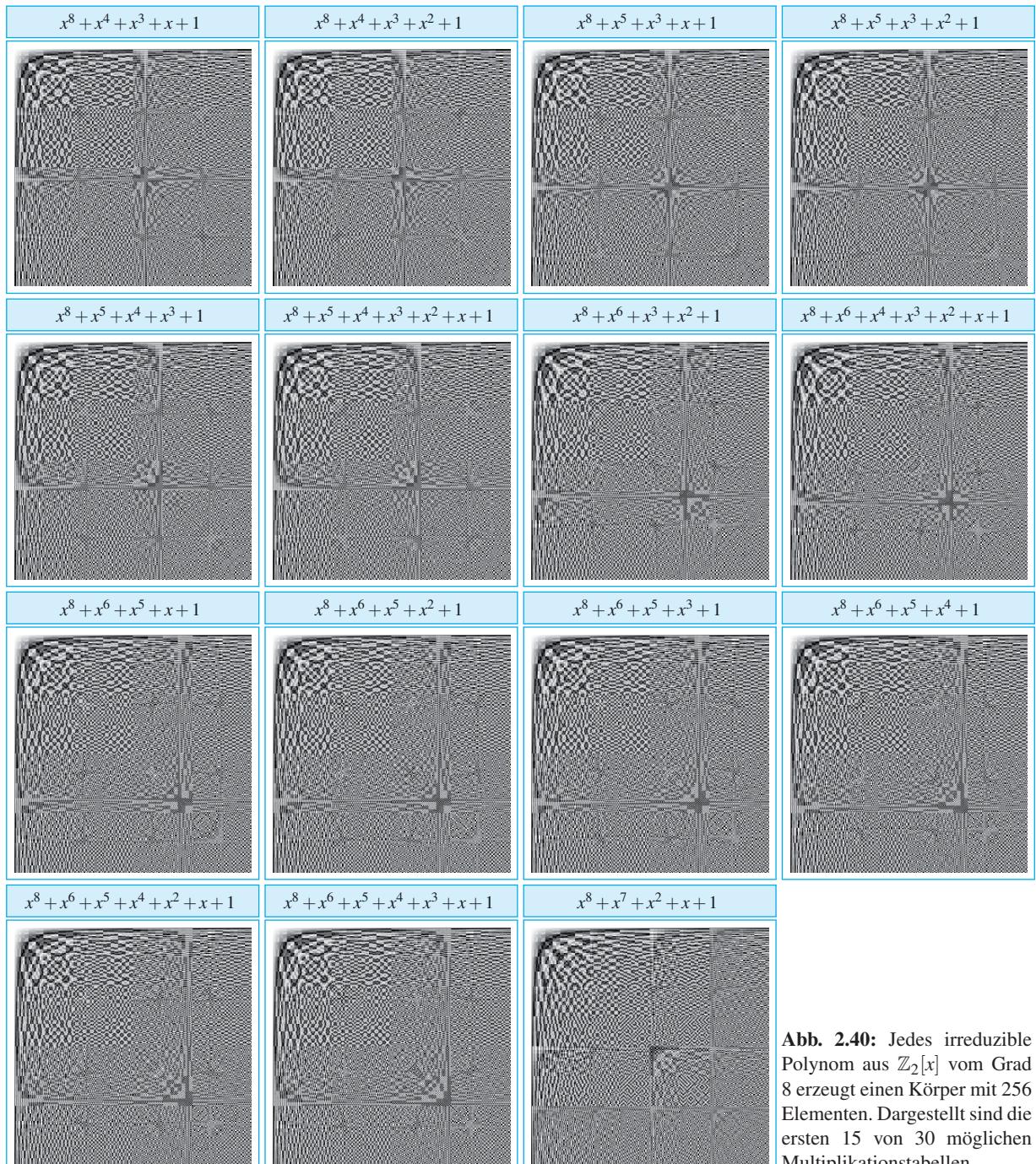


Das effiziente Rechnen in endlichen Körpern ist für die Praxis von großer Bedeutung.

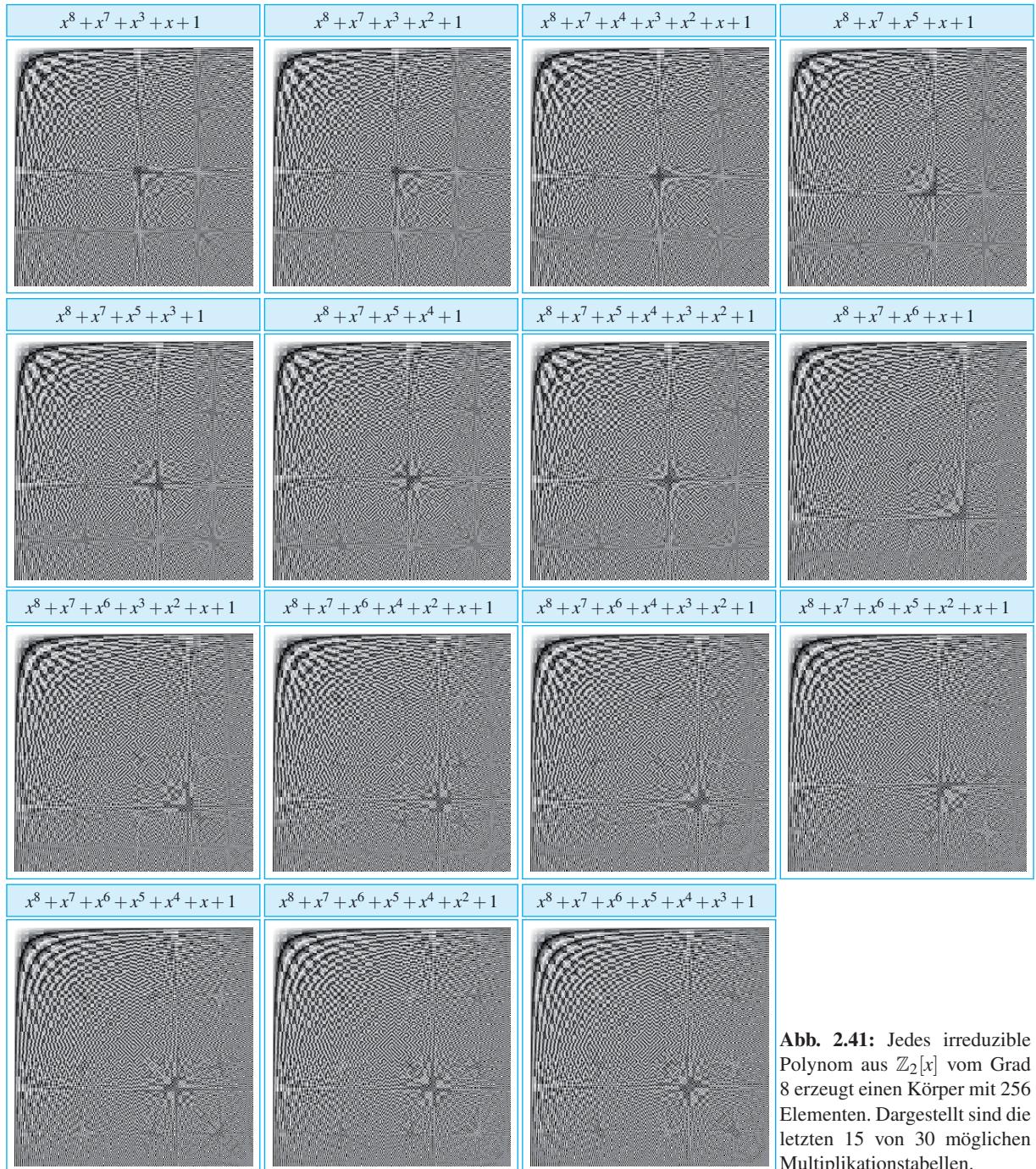
Viele technische Geräte, die wir im Alltag gerne und häufig benutzen, setzen Codierungsverfahren ein, die auf diesen Strukturen beruhen. Bekannte Beispiele sind die *Compact Disc* (CD), die *Digital Versatile Disc* (DVD) oder die *Blu-ray Disc* (BD). Alle drei verwenden eine ausgetüftelte Kombination von Korrekturverfahren, die das Abspielen von Datenträgern auch dann noch gewährleisten, wenn die Oberfläche beschmutzt oder durch Kratzer beschädigt wurde.

Eines dieser Verfahren ist die *Reed-Solomon-Codierung*, die den Datenstrom um zusätzliche Prüfbytes ergänzt. Berechnet werden die Prüfbytes mithilfe spezieller Polynome, die aus den Nutzdaten interpoliert werden. Die Polynome stammen aus der Menge  $\mathbb{F}_{256}[x]$ , d. h., die Koeffizienten sind Elemente des endlichen Körpers  $\mathbb{F}_{256}$ , den wir mittlerweile gut kennen. Beim Lesen eines optischen Datenträgers gibt es für das Abspielgerät alle Hände voll zu tun. Millionen und Abermillionen von Rechenschritten sind notwendig, um die benötigten Polynome zu interpolieren und fehlerhaft gelesene Datenbytes zu rekonstruieren.

In Abschnitt 6.4.5 werden wir uns mit der geschilderten Art der Codierung im Detail beschäftigen.



**Abb. 2.40:** Jedes irreduzible Polynom aus  $\mathbb{Z}_2[x]$  vom Grad 8 erzeugt einen Körper mit 256 Elementen. Dargestellt sind die ersten 15 von 30 möglichen Multiplikationstabellen.



**Abb. 2.41:** Jedes irreduzible Polynom aus  $\mathbb{Z}_2[x]$  vom Grad 8 erzeugt einen Körper mit 256 Elementen. Dargestellt sind die letzten 15 von 30 möglichen Multiplikationstabellen.

vorstellen. Noch kompakter lässt sich ein solches Polynom aufschreiben, wenn die 8 Binärkoeffizienten in eine zweiziffrige Hexadezimalzahl übersetzt werden:

- $x^7 + x^4 + x^2 + x + 1 = 1x^7 + 0x^6 + 0x^5 + 1x^4 + 0x^3 + 1x^2 + 1x + 1$
- 👉 10010111 (Binär)
- 👉 97 (Hexadezimal)
- $x^6 + x^4 + x^3 + x^2 + 1 = 0x^7 + 1x^6 + 0x^5 + 1x^4 + 1x^3 + 1x^2 + 0x + 1$
- 👉 01011101 (Binär)
- 👉 5D (Hexadezimal)

Legen wir die Binärdarstellung zugrunde, so ist die Addition in  $\mathbb{F}_{2^8}$  nichts anderes als die bitweise XOR-Verknüpfung der Operanden.

Die härtere Nuss haben wir bei der Multiplikation zu knacken. Zunächst ist klar, dass sich die Multiplikation zweier Polynome so aufschreiben lässt:

$$p(x)q(x) = \left( p(x) \sum_{i=7}^0 b_i x^i \right) \bmod m(x)$$

Die Summe können wir mit  $p(x)$  ausmultiplizieren und erhalten dann:

$$p(x)q(x) = \left( \sum_{i=7}^0 (p(x)b_i)x^i \right) \bmod m(x) \quad (2.16)$$

Der Ausdruck  $p(x)b_i$  beschreibt die Multiplikation des Polynoms  $p(x)$  mit dem Skalar  $b_i$ . Da für  $b_i$  nur die Werte 0 und 1 in Frage kommen, können wir  $p(x)b_i$  ganz einfach dadurch erhalten, indem wir die Koeffizienten von  $p(x)$  mit  $b_i$  UND-verknüpfen. Die anschließende Multiplikation mit  $x^i$  geht genauso einfach von der Hand: Sie entspricht einer bitweisen Verschiebung der Koeffizientenbits um  $i$  Stellen nach links. Rechnen wir das Produkt zweier Polynome anhand dieser Formel aus, so entsteht das typische Bild einer sich von links oben nach rechts unten erstreckenden Liste von Zwischenergebnissen, wie wir es von der händischen Multiplikation zweier Zahlen her kennen. Die linke Spalte in Abbildung 2.42 demonstriert die Multiplikation für die weiter oben genannten Beispieldaten.

Effizient ist diese Art der Multiplikation nicht. Auch wenn wir den Großteil der Berechnung durch einfache Bitoperationen auf den Koeffizientenvektoren erledigen können, müssen wir am Ende eine aufwendige Division durch das irreduzible Polynom  $m(x)$  durchführen.

Variante 1	Variante 2	Variante 3
$\underbrace{\left( \sum_{i=7}^0 a_i x^i \right)}_{p(x)} \cdot \underbrace{\left( \sum_{i=7}^0 b_i x^i \right)}_{q(x)} \bmod m(x)$ $= ((p(x) b_7) x^7 + (p(x) b_6) x^6 + \dots + (p(x) b_1) x + (p(x) b_0)) \bmod m(x)$	$\underbrace{\left( \sum_{i=7}^0 a_i x^i \right)}_{p(x)} \cdot \underbrace{\left( \sum_{i=7}^0 b_i x^i \right)}_{q(x)} \bmod m(x)$ $= ((\dots(p(x) b_7) x + p(x) b_6) x + \dots + p(x) b_1) x + p(x) b_0) \bmod m(x)$	$\underbrace{\left( \sum_{i=7}^0 a_i x^i \right)}_{p(x)} \cdot \underbrace{\left( \sum_{i=7}^0 b_i x^i \right)}_{q(x)} \bmod m(x)$ $= ((\dots(p(x) b_7 \bmod m(x)) x + p(x) b_6 \bmod m(x)) x + \dots + p(x) b_1 \bmod m(x)) x + p(x) b_0 \bmod m(x)$
<p>■ Multiplikation</p> $  \begin{array}{r}  10010111 \cdot 01011101 \\  00000000 \\  10010111 \\  00000000 \\  10010111 \\  10010111 \\  10010111 \\  00000000 \\  10010111 \\  \hline  010101011000011  \end{array}  $ <p>■ Division durch <math>m(x)</math></p> $  \begin{array}{r}  10101011000011 : 100011011 \\  100011011 \\  \hline  0100110100 \\  100011011 \\  \hline  00101111011 \\  100011011 \\  \hline  01100000  \end{array}  $	<p>■ Multiplikation</p> $  \begin{array}{r}  10010111 \cdot 01011101 \\  00000000 \\  10010111 \\  \hline  010010111 \\  00000000 \\  \hline  0100101110 \\  10010111 \\  \hline  01011001011 \\  10010111 \\  \hline  010100000001 \\  10010111 \\  \hline  01010100101 \\  00000000 \\  \hline  010101001010 \\  10010111 \\  \hline  010101011000011  \end{array}  $ <p>■ Division durch <math>m(x)</math></p> $  \begin{array}{r}  10101011000011 : 100011011 \\  100011011 \\  \hline  0100110100 \\  100011011 \\  \hline  00101111011 \\  100011011 \\  \hline  01100000  \end{array}  $	<p>■ Multiplikation modulo <math>m(x)</math></p> $  \begin{array}{r}  10010111 \cdot 01011101 \\  00000000 \\  10010111 \\  \hline  10010111 \\  00000000 \\  \hline  100101110 \\  100011011 \\  \hline  00110101 \\  10010111 \\  \hline  11111101 \\  10010111 \\  \hline  101101101 \\  100011011 \\  \hline  01110110 \\  10010111 \\  \hline  01111011 \\  00000000 \\  \hline  11110110 \\  10010111 \\  \hline  101111011 \\  100011011 \\  \hline  01100000  \end{array}  $ <p>(☞ <math>x^m = m</math>)</p>

Abb. 2.42: Händische Multiplikation in  $\mathbb{F}_{2^8}$

## Addition

```
uint8_t add(uint8_t a,
            uint8_t b)
{
    return a ^ b;
}
```

## Subtraktion

```
uint8_t sub(uint8_t a,
            uint8_t b)
{
    return a ^ b;
}
```

## Multiplikation

```
uint8_t mul(uint8_t a,
            uint8_t b,
            uint16_t m)
{
    uint16_t r = 0;
    unsigned i;

    for(i = 0; i < 8; i++) {
        r <= 1;
        if (b & 0x80)
            r ^= a;
        if (r & 0x100)
            r ^= m;
        b <= 1;
    }
    return r;
}
```

**Abb. 2.43:** Schnelle Galoiskörper-Arithmetik, implementiert in der Programmiersprache C

Wir wollen versuchen, die aufwendige Polynomdivision zu vermeiden, und schreiben dazu Gleichung (2.16) ein wenig um. Indem wir das Polynom

$$q(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$$

entsprechend dem *Horner-Schema*

$$q(x) = (((...((b_7x + b_6)x + b_5)x + b_4)x + b_3)x + b_2)x + b_1)x + b_0$$

entwickeln und rekursiv mit  $p(x)$  ausmultiplizieren, erhalten wir die Formel, die in Abbildung 2.42 in der mittleren Spalte abgedruckt ist. Auf den ersten Blick scheint der Unterschied nur marginal zu sein. Anstatt alle Zwischenergebnisse ganz am Ende zu summieren, wird jetzt nach jedem Zwischenschritt addiert. Rechnen wir das Produkt händisch aus, so ist diese Variante sogar noch aufwendiger als die erste.

Der Vorteil dieser Variante ist, dass sie sich zu einem effizienten Verfahren weiterentwickeln lässt. Dies gelingt, indem die Division durch  $m(x)$  nicht mehr am Ende ausgeführt wird, sondern jedes Mal, wenn wir ein neues Zwischenergebnis erhalten.

Es scheint, als würde dies den Aufwand weiter erhöhen, da wir die Modulo-Reduktion jetzt viel häufiger durchführen müssen. Erst auf den zweiten Blick wird deutlich, warum wir dennoch effizienter rechnen können als vorher: Als Zwischenergebnisse können nur noch Polynome mit einem Grad kleiner oder gleich  $n$  entstehen, und in beiden Fällen können wir die Modulo-Berechnung effizient durchführen. Ist der Grad kleiner als  $n$ , so müssen wir nichts tun. Ist der Grad gleich  $n$ , so können wir den Divisionsrest ganz einfach durch die Subtraktion von  $m(x)$  erhalten.

Die Subtraktion in  $\mathbb{Z}_2$  ist identisch mit der Addition, und diese ist wiederum identisch mit der XOR-Verknüpfung. Das bedeutet, dass wir in jedem Rechenschritt lediglich eine zusätzliche XOR-Verknüpfung mit  $m(x)$  durchführen müssen, sobald als Zwischenergebnis ein Polynom vom Grad  $n$  entsteht. Auf diese Weise können wir auf die kostspielige Polynomdivision am Ende verzichten und erhalten als Ergebnis den Algorithmus zur Galoiskörper-Multiplikation in Abbildung 2.43.

## Logarithmieren in endlichen Körpern

Mit den bisher angestellten Überlegungen haben wir es geschafft, die Komplexität der Galoiskörper-Multiplikation deutlich zu reduzieren. Der im letzten Abschnitt erarbeitete Algorithmus ist kaum langsamer als die gewöhnliche Multiplikation zweier Binärzahlen und versetzt uns in die Lage, effizient in endlichen Körpern zu rechnen. In diesem Abschnitt werden Sie sehen, dass sich die Berechnung mit einem einfachen Trick noch weiter beschleunigen lässt.

Die Optimierung basiert auf dem bekannten Zusammenhang

$$\log_b(x \cdot y) = \log_b x + \log_b y \quad (2.17)$$

aus der Logarithmenrechnung. Die Formel besagt, dass der Logarithmus des Produkts  $x \cdot y$  das Gleiche ist wie die Summe der Logarithmen von  $x$  und  $y$ . Diese Gesetzmäßigkeit ist von praktischem Nutzen: Sie ermöglicht es, das Produkt zweier Zahlen  $x$  und  $y$  zu berechnen, ohne eine einzige Multiplikation auszuführen. Drei Schritte sind hierfür notwendig:

- Logarithmieren
  - ☞ Berechne  $\log_b x$  und  $\log_b y$
- Addieren
  - ☞ Berechne  $\log_b x + \log_b y$
- Potenzieren
  - ☞ Berechne  $b^{\log_b x + \log_b y}$

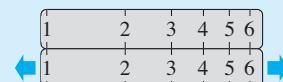
Das Ergebnis ist das Produkt  $x \cdot y$ . Unter der Annahme, dass sich die Logarithmusfunktion und die Potenzfunktion effizient berechnen lassen, haben wir eine erhebliche Vereinfachung erreicht. Über die Formel (2.17) ist es uns gelungen, die Multiplikation auf die viel einfachere Addition zurückzuführen.

Können wir die Idee auf Polynome übertragen und die Multiplikation dort in ähnlicher Weise auf die Addition zurückführen? Bevor wir diese Frage beantworten können, müssen wir zunächst klären, was das Logarithmieren einer Zahl genau bedeutet. In Worten ausgedrückt verbirgt sich hinter dem Logarithmieren einer Zahl  $x$  die Aufgabe, eine Zahl  $y$  mit der Eigenschaft

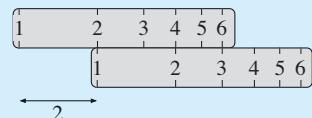
$$b^y = x$$



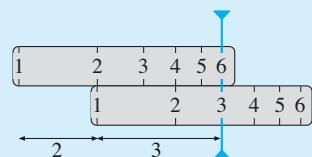
Der Trick, mit dem wir die Multiplikation auf die Addition zurückführen, ist keinesfalls neu. Er entspricht jenem, den Generationen von Mathematikern dazu verwendet haben, um Zahlen mithilfe eines *Rechenschiebers* zu multiplizieren. Ein solches Gerät bestand aus zwei identischen logarithmischen Skalen, von denen eine frei verschoben werden konnte:



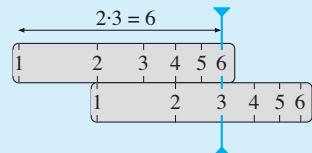
Um das Produkt zweier Zahlen zu berechnen, musste die verschiebbare Skala – die sogenannte *Zunge* – um den Wert des ersten Faktors versetzt werden. Für die Berechnung von  $2 \cdot 3$  sah dies so aus:



Danach wurde eine bewegliche Markierung – der sogenannte *Läufer* – über den zweiten Faktor geschoben.



Jetzt ließ sich das Ergebnis an der ersten Skala direkt ablesen:



Aus heutiger Sicht ist der Rechenschieber ein Relikt einer anderen Zeit. Bereits wenige Jahre nach der Erfindung des elektronischen Taschenrechners war er aus den hiesigen Rechenstuben fast vollständig verschwunden.

$x$	(02)
$\downarrow \cdot x \bmod x^8 + x^4 + x^3 + x + 1$	
$x^2$	(04)
$\downarrow \cdot x \bmod x^8 + x^4 + x^3 + x + 1$	
$x^3$	(08)
$\downarrow \cdot x \bmod x^8 + x^4 + x^3 + x + 1$	
$x^4$	(10)
$\downarrow \cdot x \bmod x^8 + x^4 + x^3 + x + 1$	
$x^5$	(20)
$\downarrow \cdot x \bmod x^8 + x^4 + x^3 + x + 1$	
$x^6$	(40)
$\downarrow \cdot x \bmod x^8 + x^4 + x^3 + x + 1$	
$x^7$	(80)
$\downarrow \cdot x \bmod x^8 + x^4 + x^3 + x + 1$	
$x^4 + x^3 + x + 1$	(1B)
$\downarrow \cdot x \bmod x^8 + x^4 + x^3 + x + 1$	
$x^5 + x^4 + x^2 + x$	(36)
$\downarrow \cdot x \bmod x^8 + x^4 + x^3 + x + 1$	
$x^6 + x^5 + x^3 + x^2$	(6C)
$\downarrow \cdot x \bmod x^8 + x^4 + x^3 + x + 1$	
$\dots$	(D8) (AB) (4D) (9A) (2F) (5E) (BC) (63)
	(C6) (97) (35) (6A) (D4) (B3) (7D) (FA)
	(EF) (C5) (91) (39) (72) (E4) (D3) (BD)
	(61) (C2) (9F) (25) (4A) (94) (33) (66)
	(CC) (83) (1D) (3A) (74) (E8) ...
$\downarrow \cdot x \bmod x^8 + x^4 + x^3 + x + 1$	
$x^7 + x^6 + x^3 + x + 1$	(CB)
$\downarrow \cdot x \bmod x^8 + x^4 + x^3 + x + 1$	
$x^7 + x^3 + x^2 + 1$	(8D)
$\downarrow \cdot x \bmod x^8 + x^4 + x^3 + x + 1$	
1	(01)
$\downarrow \cdot x \bmod x^8 + x^4 + x^3 + x + 1$	
$x$	(02)

**Abb. 2.44:** Das Polynom  $g(x) = x$  generiert lediglich 51 Körperelemente von  $\mathbb{Z}_2[x]_{x^8+x^4+x^3+x+1}$ . Als Basis des diskreten Logarithmus scheidet es daher aus.

zu finden. Die Zahl  $b$  ist die Basis der verwendeten Logarithmusfunktion. Übertragen wir den Zusammenhang auf Polynome, so tritt an die Stelle von  $x$  ein Polynom  $p(x)$  und an die Stelle der Basis  $b$  ein Polynom  $g(x)$ , das wir als *Generatorpolynom* bezeichnen. Der Logarithmus von  $p(x)$  zur Basis  $g(x)$  wäre dann eine Zahl  $y$  mit der Eigenschaft

$$g(x)^y = p(x). \quad (2.18)$$

Die Zahl  $y$  heißt der *diskrete Logarithmus* des Polynoms  $p(x)$ .

Würde sich ein  $y$  mit dieser Eigenschaft immer finden lassen, dann ließe sich das Produkt von  $p(x)$  und  $q(x)$  ganz einfach so ausrechnen:

$$p(x) \cdot q(x) = g(x)^{y_1} \cdot g(x)^{y_2} = g(x)^{y_1+y_2} \quad (2.19)$$

Hierin sind  $y_1$  und  $y_2$  die diskreten Logarithmen von  $p(x)$  und  $q(x)$ .

Auf den ersten Blick kommt Gleichung (2.19) reichlich schwerfällig daher, doch bei genauerer Betrachtung wird klar, dass wir mit ihr am Ziel sind. Rechnen wir z. B. in  $\mathbb{F}_{2^8}$ , dann existieren so wenige Elemente, dass wir das Logarithmieren und das Potenzieren durch einen simplen Tabellenzugriff erledigen können. Konkret benötigen wir hierfür die folgenden beiden Tabellen mit jeweils 256 Einträgen:

### ■ Logarithmentabelle

Diese Tabelle enthält für jedes der 256 Polynome  $p(x)$  den Wert  $y$  aus Gleichung (2.18).

### ■ Potenzentabelle

Diese Tabelle speichert an der  $i$ -ten Position das Polynom  $g(x)^i$ , d. h. die  $i$ -te Potenz des Generatorpolynoms.

Mit Hilfe der beiden Tabellen lässt sich die Multiplikation auf drei hintereinander auszuführende Elementaroperationen reduzieren:

### ■ Nachschlagen

☞ Extrahiere  $y_1$  und  $y_2$  aus der Logarithmentabelle

### ■ Addieren

☞ Berechne  $y_1 + y_2$

### ■ Nachschlagen

☞ Lese das Feld  $y_1 + y_2$  der Potenzentabelle aus

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	00	19	01	32	02	1a	c6	4b	c7	1b	68	33	ee	df	03	
1	64	04	e0	0e	34	8d	81	ef	4c	71	08	c8	f8	69	1c	c1
2	7d	c2	1d	b5	f9	b9	27	6a	4d	e4	a6	72	9a	c9	09	78
3	65	2f	8a	05	21	0f	e1	24	12	f0	82	45	35	93	da	8e
4	96	8f	db	bd	36	d0	ce	94	13	5c	d2	f1	40	46	83	38
5	66	dd	fd	30	bf	06	8b	62	b3	25	e2	98	22	88	91	10
6	7e	6e	48	c3	a3	b6	1e	42	3a	6b	28	54	fa	85	3d	ba
7	2b	79	0a	15	9b	9f	5e	ca	4e	d4	ac	e5	f3	73	a7	57
8	af	58	a8	50	f4	ea	d6	74	4f	ae	e9	d5	e7	e6	ad	e8
9	2c	d7	75	7a	eb	16	0b	f5	59	cb	5f	b0	9c	a9	51	a0
a	7f	0c	f6	6f	17	c4	49	ec	d8	43	1f	2d	a4	76	7b	b7
b	cc	bb	3e	5a	fb	60	b1	86	3b	52	a1	6c	aa	55	29	9d
c	97	b2	87	90	61	be	dc	fc	bc	95	cf	cd	37	3f	5b	d1
d	53	39	84	3c	41	a2	6d	47	14	2a	9e	5d	56	f2	d3	ab
e	44	11	92	d9	23	20	2e	89	b4	7c	b8	26	77	99	e3	a5
f	67	4a	ed	de	c5	31	fe	18	0d	63	8c	80	c0	f7	70	07

**Tab. 2.1:** Logarithmentabelle für die schnelle Galoiskörper-Multiplikation. Die markierten Felder enthalten die diskreten Logarithmen der Polynome aus Abbildung 2.45.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	01	03	05	0f	11	33	55	ff	1a	2e	72	96	a1	f8	13	35
1	5f	e1	38	48	d8	73	95	a4	f7	02	06	0a	1e	22	66	aa
2	e5	34	5c	e4	37	59	eb	26	6a	be	d9	70	90	ab	e6	31
3	53	f5	04	0c	14	3c	44	cc	4f	d1	68	b8	d3	6e	b2	cd
4	4c	d4	67	a9	e0	3b	4d	d7	62	a6	f1	08	18	28	78	88
5	83	9e	b9	d0	6b	bd	dc	7f	81	98	b3	ce	49	db	76	9a
6	b5	c4	57	f9	10	30	50	f0	0b	1d	27	69	bb	d6	61	a3
7	fe	19	2b	7d	87	92	ad	ec	2f	71	93	ae	e9	20	60	a0
8	fb	16	3a	4e	d2	6d	b7	c2	5d	e7	32	56	fa	15	3f	41
9	c3	5e	e2	3d	47	c9	40	c0	5b	ed	2c	74	9c	bf	da	75
a	9f	ba	d5	64	ac	ef	2a	7e	82	9d	bc	df	7a	8e	89	80
b	9b	b6	c1	58	e8	23	65	af	ea	25	6f	b1	c8	43	c5	54
c	fc	1f	21	63	a5	f4	07	09	1b	2d	77	99	b0	cb	46	ca
d	45	cf	4a	de	79	8b	86	91	a8	e3	3e	42	c6	51	f3	0e
e	12	36	5a	ee	29	7b	8d	8c	8f	8a	85	94	a7	f2	0d	17
f	39	4b	dd	7c	84	97	a2	fd	1c	24	6c	b4	c7	52	f6	01

**Tab. 2.2:** Potenzentabelle für die schnelle Galoiskörper-Multiplikation. Das markierte Feld enthält das Multiplikationsergebnis für das Beispiel aus Abbildung 2.45.

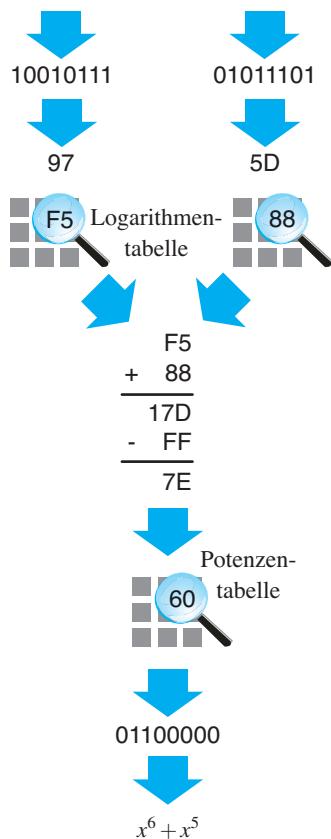
Nach all dem Gesagten dürfen wir nicht vergessen, dass unser Trick nur dann funktioniert, wenn sich jedes Polynom auch wirklich logarithmieren lässt. Um dies zu gewährleisten, müssen wir ein Polynom finden, mit dem sich alle Körperelemente nacheinander aufzählen lassen. Konkret hat ein solches Polynom  $g(x)$  die Eigenschaft, dass in der Folge

$$g(x), g(x) \cdot g(x), g(x) \cdot g(x) \cdot g(x), g(x) \cdot g(x) \cdot g(x) \cdot g(x), \dots \quad (2.20)$$

ausnahmslos alle von 0 verschiedenen Körperelemente vorkommen. Fehlt in dieser Folge auch nur ein Einziges, so ist das Polynom  $g(x)$  für unsere Zwecke ungeeignet.

Gibt es überhaupt Polynome mit der von uns benötigten Eigenschaft? Als Beispiel legen wir erneut den Körper  $\mathbb{Z}_2[x]_{x^8+x^4+x^3+x+1}$  zugrunde und berechnen die Folge (2.20) zunächst einmal für das Polynom  $g(x) = x$  (Hexadezimal 02). Multiplizieren wir das Polynom, wie in Abbildung 2.44 gezeigt, sukzessive mit sich selbst, so werden insgesamt 51 Körperelemente erzeugt. Die nächste Multiplikation führt auf das Element  $x$  zurück und der Zyklus wiederholt sich von vorne. Da wir ein Generatorelement benötigen, das alle 255 von 0 verschiedenen Körperelemente erzeugt, kommt das Polynom als Basis für den diskreten Logarithmus nicht in Frage.

$$x^7 + x^4 + x^2 + x + 1 \quad x^6 + x^4 + x^3 + x^2 + 1$$



**Abb. 2.45:** Mithilfe des diskreten Logarithmus lässt sich die Multiplikation in endlichen Körpern durch wenige Tabellenzugriffe erledigen.

Trotzdem haben wir Glück, denn Polynome mit der geforderten Eigenschaft existieren tatsächlich! Wir müssen auch gar nicht lange suchen, denn bereits das Polynom

$$g(x) = x + 1$$

erfüllt unseren Zweck. Es generiert die Körperelemente in der Reihenfolge, wie sie in Tabelle 2.2 zu sehen ist. Aus Platzgründen ist die Tabelle zweidimensional angelegt. Die Zeilennummer definiert die oberen vier Bits und die Spaltennummer die unteren vier Bits des Exponenten. Demnach erhalten wir die Folge (2.20), indem wir die Tabelleneinträge in der uns vertrauten Leserichtung von links nach rechts und von oben nach unten aufsagen.

Tabelle 2.1 enthält die diskreten Logarithmen zur Basis  $g(x) = x + 1$ . Die Zeilen- und die Spaltennummer definieren die oberen bzw. die unteren 4 Bits des Polynoms  $p(x)$  und der Tabelleneintrag enthält den passenden Wert für  $y$ , mit dem sich Gleichung (2.18) lösen lässt. Da das Polynom  $x + 1$  sämtliche von 0 verschiedenen Körperelemente erzeugt, ist die Funktion für alle Polynome ungleich 0 definiert.

Abbildung 2.45 zeigt, wie wir die weiter oben durchgeführte Multiplikation von

$$x^7 + x^4 + x^2 + x + 1 \quad \text{und} \quad x^6 + x^4 + x^3 + x^2 + 1$$

auf die geschilderte Weise berechnen können. Wahrscheinlich ist Ihnen bei der Betrachtung des Beispiels aufgefallen, dass bei der Addition der diskreten Logarithmen ein Überlauf entstanden ist. Hätten wir diesen missachtet, so wäre auf ein Element außerhalb der Exponententabelle zugegriffen worden. Da sich die Exponenten nach jeweils 255 Elementen wiederholen, lässt sich ein Überlauf ganz einfach durch die Subtraktion von FF kompensieren. Als Ergebnis erhalten wir die Tabellenposition 7E, und genau dort finden wir in der Potenzentabelle das erwartete Ergebnis wieder. Es ist das Polynom  $x^6 + x^5$ , repräsentiert durch den Koeffizientenvektor 60.

## Hardware-Implementierung

Die Galoiskörper-Multiplikation lässt sich noch weiter beschleunigen, wenn wir sie nicht in Software, sondern in Hardware implementieren. Viele Spezialprozessoren aus dem Bereich der digitalen Signalverarbeitung sind mit speziellen Arithmetikeinheiten für das Rechnen in endlichen Körpern ausgestattet und damit in der Lage, eine Multiplikation in einem einzigen Takt auszuführen.

Abbildung 2.46 zeigt, wie eine solche Multiplikationseinheit aussehen kann. Die Schaltung stammt aus [25] und arbeitet nach dem gleichen Prinzip wie der Multiplikationsalgorithmus aus Abbildung 2.43.

Als Eingabe nimmt der Multiplizierer die Eingabevektoren

$$\begin{aligned} & (a_3, a_2, a_1, a_0) \\ & (b_3, b_2, b_1, b_0) \\ & (hb_4, hb_3, hb_2, hb_1, hb_0) \\ & (m_4, m_3, m_2, m_1, m_0) \end{aligned}$$

entgegen und berechnet daraus den Ausgabevektor

$$(c_3, c_2, c_1, c_0)$$

An die Eingänge  $a_i, b_i$  werden die Koeffizienten der zu multiplizierenden Polynome  $p(x)$  und  $q(x)$  und an  $m_i$  die Koeffizienten des irreduziblen Polynoms  $m(x)$  angelegt. Die zusätzlichen Eingänge  $hb_i$  dienen dazu, den Leitkoeffizienten von  $m(x)$  zu markieren.  $hb$  steht für *highest bit*:

$$hb_i = \begin{cases} 1 & \text{falls } i = \deg m(x) \\ 0 & \text{falls } i \neq \deg m(x) \end{cases}$$

An den Ausgangsleitungen  $c_0, c_1, \dots$  können die Koeffizienten des Produktpolynoms  $p(x) \cdot q(x)$  abgelesen werden, gerechnet in  $\mathbb{Z}_2[x]_{m(x)}$ .

Intern besteht der Multiplizierer aus einem mehrzeiligen Array von Multiplikationszellen, das von den Eingabewerten von oben nach unten durchlaufen wird. Vergleichen wir die Hardwareschaltung mit der Software-Implementierung aus Abbildung 2.43, so entspricht jede Zeile des Hardware-Arrays einer Iteration der For-Schleife.

Jede Zeile des Arrays wird durch mehrere Multiplikationszellen gebildet, deren Aufbau ebenfalls in Abbildung 2.46 erklärt ist. Jede Zelle besitzt sechs Eingangsleitungen ( $a_{in}, b_{in}, c_{in}, hb_{in}, m_{in}$  und  $y_{in}$ ) und fünf Ausgangsleitungen ( $a_{out}, b_{out}, c_{out}, hb_{out}$  und  $m_{out}$ ). Das UND-Gatter rechts oben berechnet das Produkt der Koeffizienten  $a_i$  und  $b_i$  und das XOR-Gatter darunter addiert das Ergebnis auf. Beide Gatter zusammen erfüllen damit die gleiche Funktion wie der Befehl

```
if (b & 0x80)
    r ^= a;
```

in der Software-Implementierung. Das nachgeschaltete XOR-Gatter subtrahiert bei Bedarf das Polynom  $m(x)$ . Ausgelöst wird die Subtraktion durch eine Überlauflogik, die aufgrund der Werte von  $hb_{in}, y_{in}$  und

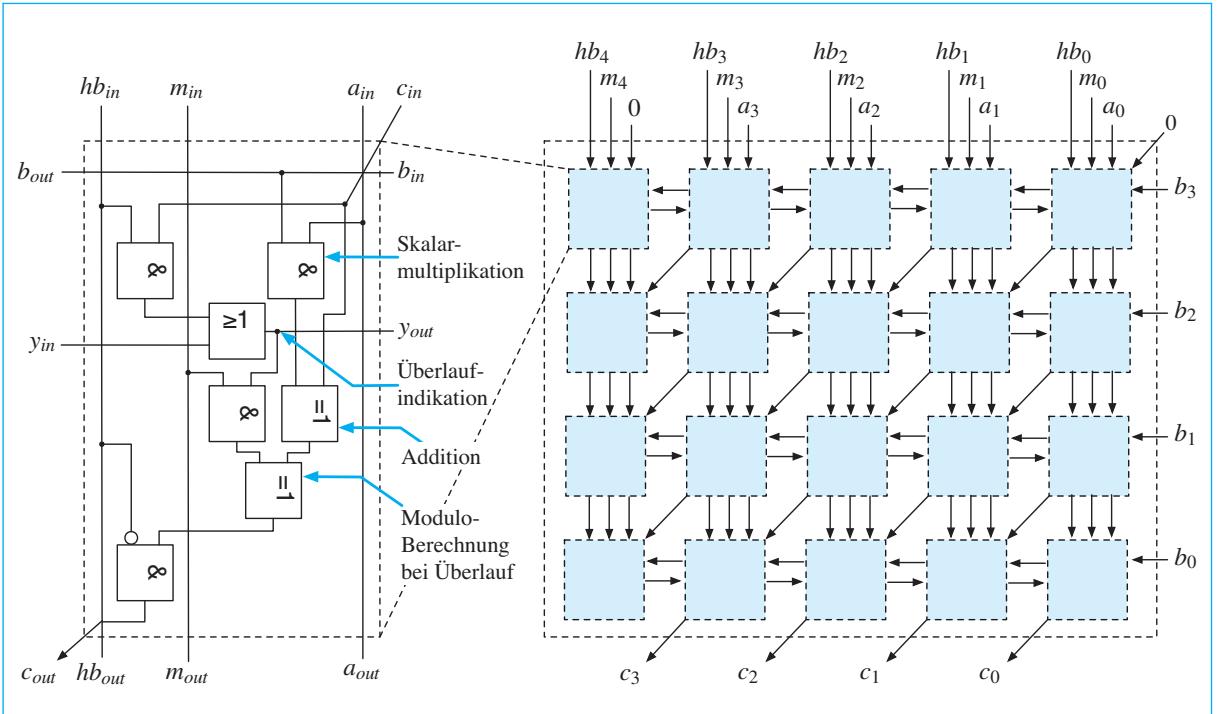


Abb. 2.46: Galoiskörper-Multiplizierer (nach [25])

$c_{in}$  feststellt, ob sie durchgeführt werden muss oder nicht. Insgesamt erfüllt das nachgeschaltete XOR-Gatter damit die gleiche Funktion wie folgendes Programmfragment:

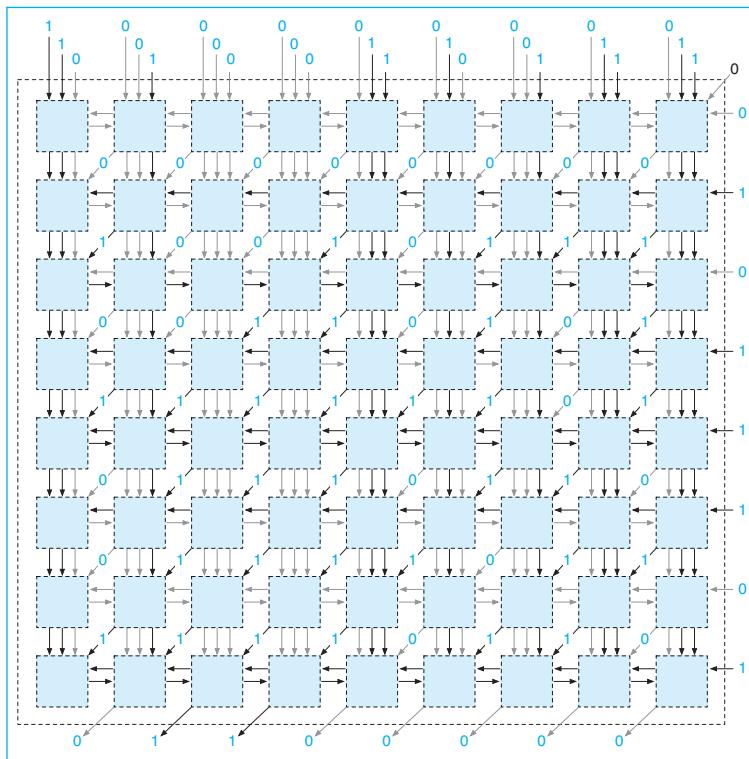
```
if (r & 0x100)
    r ^= m;
```

Die im Programm zusätzlich enthaltene Schiebeoperation

```
r <= 1;
```

erfordert in der Hardwareschaltung keinen Aufwand. Sie wird automatisch ausgeführt, da die Bits spaltenversetzt von einer Zeile in die nächste weitergegeben werden.

Abbildung 2.47 zeigt, wie sich der Multiplizierer in Aktion verhält. Die Eingangsleitungen sind mit den Koeffizienten der oben eingeführten Beispieldaten  $x^7 + x^4 + x^2 + x + 1$  und  $x^6 + x^4 + x^3 + x^2 + 1$  sowie dem irreduziblen Polynom  $x^8 + x^4 + x^3 + x + 1$  belegt. Unter dieser Ein-



**Abb. 2.47:** Hardware-Implementierung der Galoiskörper-Multiplikation

$$\begin{array}{r}
 10010111 \cdot 01011101 \\
 00000000 \\
 10010111 \\
 \hline
 10010111 \\
 00000000 \\
 \hline
 100101110 \\
 100011011 \\
 \hline
 00110101 \\
 10010111 \\
 \hline
 11111101 \\
 10010111 \\
 \hline
 101101101 \\
 100011011 \\
 \hline
 01110110 \\
 10010111 \\
 \hline
 01111011 \\
 00000000 \\
 \hline
 11110110 \\
 10010111 \\
 \hline
 101111011 \\
 100011011 \\
 \hline
 01100000
 \end{array}$$

(☞  $x^m = m$ )

(☞  $x^m = m$ )

(☞  $x^m = m$ )

gabe produziert die Schaltung mit dem Bitmuster 01100000 das gleiche Ergebnis wie unsere händischen Rechnungen aus Abbildung 2.42.

Ein Blick auf die berechneten Zwischenergebnisse zeigt, dass die Hardwareschaltung intern nach demselben Berechnungsschema arbeitet wie die dritte Variante aus Abbildung 2.42. Die in jeder Zeile generierten Bits stimmen mit den händisch berechneten Zwischenergebnissen eins zu eins überein.

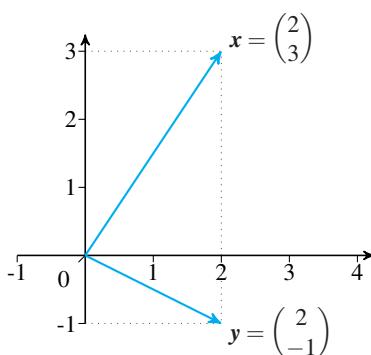


Abb. 2.48: Pfeildarstellung von Vektoren

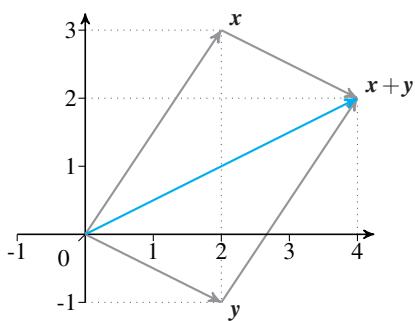


Abb. 2.49: Geometrische Interpretation der Vektoraddition

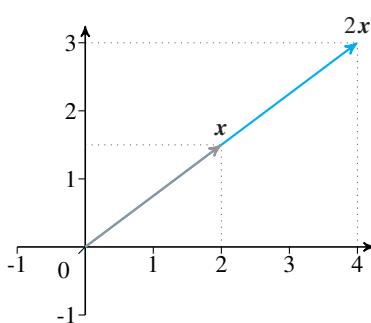


Abb. 2.50: Geometrische Interpretation der Vektormultiplikation

## 2.5 Vektorräume

In diesem Abschnitt wenden wir uns einem zentralen Begriff aus dem Gebiet der linearen Algebra zu: dem *Vektorraum*. Zu den bekanntesten Vertretern dieser Struktur gehört der Vektorraum  $\mathbb{R}^2$ , der in der analytischen Geometrie für die Beschreibung der euklidischen Ebene verwendet wird. In diesem Vektorraum gibt es zwei verschiedene Arten von Objekten:

### ■ Vektoren

Ein Vektor  $x \in \mathbb{R}^2$  ist eine geordnete Zusammenfassung zweier reeller Zahlen  $x_0$  und  $x_1$  und wird gewöhnlich in einer der nachstehenden Formen angegeben:

$$\begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \text{ oder } (x_0 \ x_1)$$

Wird die erste Schreibweise gewählt, so sprechen wir von einem *Spaltenvektor*, andernfalls von einem *Zeilenvektor*. Geometrisch lassen sich die Vektoren aus  $\mathbb{R}^2$  durch Richtungspfeile darstellen (Abbildung 2.48).

Auf der Menge der Vektoren ist eine Addition definiert. In  $\mathbb{R}^2$  lässt sich die Summe zweier Vektoren durch die separate Addition der Vektorkomponenten berechnen:

$$x + y = \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} + \begin{pmatrix} y_0 \\ y_1 \end{pmatrix} := \begin{pmatrix} x_0 + y_0 \\ x_1 + y_1 \end{pmatrix}$$

Die Addition lässt sich auch geometrisch deuten. Sie entspricht der Verkettung zweier Richtungspfeile (Abbildung 2.49).

### ■ Skalare

In  $\mathbb{R}^2$  lassen sich Vektoren nicht nur addieren, sondern auch mit reellen Zahlen multiplizieren. Das Produkt  $c \cdot x$  entsteht, indem die Vektorkomponenten von  $x$  separat mit der reellen Zahl  $c$  multipliziert werden:

$$c \cdot x = c \cdot \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} := \begin{pmatrix} c \cdot x_0 \\ c \cdot x_1 \end{pmatrix}$$

Geometrisch können wir die Multiplikation als die Streckung ( $c > 1$ ) oder die Stauchung ( $c < 1$ ) eines Vektors um den Faktor  $c$  interpretieren (Abbildung 2.50).

Gruppeneigenschaften von ,+‘	Zusätzliche Eigenschaften
Kommutativgesetz $\begin{pmatrix} x_0 \\ x_1 \end{pmatrix} + \begin{pmatrix} y_0 \\ y_1 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \end{pmatrix} + \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}$	Multiplikatives Assoziativgesetz $c \cdot \left( d \cdot \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \right) = (c \cdot d) \cdot \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}$
Additives Assoziativgesetz $\left( \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} + \begin{pmatrix} y_0 \\ y_1 \end{pmatrix} \right) + \begin{pmatrix} z_0 \\ z_1 \end{pmatrix} = \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} + \left( \begin{pmatrix} y_0 \\ y_1 \end{pmatrix} + \begin{pmatrix} z_0 \\ z_1 \end{pmatrix} \right)$	Distributivgesetze $c \cdot \left( \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} + \begin{pmatrix} y_0 \\ y_1 \end{pmatrix} \right) = c \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} + c \begin{pmatrix} y_0 \\ y_1 \end{pmatrix}$ $(c+d) \cdot \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = c \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} + d \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}$
Neutrales additives Element $\mathbf{0} + \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}$	Neutrales multiplikatives Element $1 \cdot \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}$
Additive Inverse $\begin{pmatrix} x_0 \\ x_1 \end{pmatrix} + \begin{pmatrix} -x_0 \\ -x_1 \end{pmatrix} = \mathbf{0}$	

Abb. 2.51: Rechenregeln in  $\mathbb{R}^2$ 

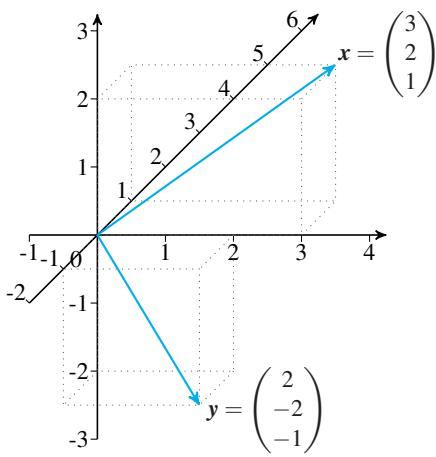
In  $\mathbb{R}^2$  besteht zwischen den Skalaren und den Vektoren eine große Ähnlichkeit, da beide auf dem Begriff der reellen Zahl aufbauen. Zwingend ist dieser Zusammenhang nicht. In anderen Räumen können Vektoren und Skalare völlig unterschiedliche Objekte sein, so dass wir beide Objekttypen gedanklich sauber voneinander trennen müssen.

In  $\mathbb{R}^2$  gelten viele der uns bekannten Rechenregeln. Zum einen ist die Vektoraddition kommutativ und assoziativ. Zusätzlich existiert mit dem Nullvektor

$$\mathbf{0} := \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

ein neutrales Element und wir können durch die Addition des komponentenweise negierten Vektors  $-\mathbf{x}$  jeden beliebigen Vektor  $\mathbf{x}$  auf den Nullvektor zurückführen. Mit  $(\mathbb{R}^2, +)$  haben wir damit eine wohlvertraute Struktur vor uns: eine kommutative Gruppe.

Die Multiplikation mehrerer Skalare mit einem Vektor ist ebenfalls assoziativ und die Multiplikation der reellen Zahl 1 mit einem beliebigen



**Abb. 2.52:** Die geometrische Entsprechung des Vektorraums  $\mathbb{R}^3$  ist der (dreidimensionale) euklidische Raum.

Vektor  $x$  ergibt wieder das Element  $x$ . Im Zusammenspiel folgen die Operationen  $,+$  und  $,\cdot$  ebenfalls einem bekannten Schema: Sie erfüllen die beiden in Abbildung 2.51 gezeigten Distributivgesetze.

Die am Beispiel von  $\mathbb{R}^2$  herausgearbeiteten Eigenschaften wollen wir als Grundlage nehmen, um Vektorräume formal zu definieren.



### Definition 2.13 (Vektorraum)

Sei  $\mathbb{K}$  ein Körper. Eine Menge  $V$  bildet mit den beiden Abbildungen

$$,+^{\prime} : V \times V \rightarrow V \quad \text{und} \quad ,\cdot^{\prime} : \mathbb{K} \times V \rightarrow V$$

einen *Vektorraum* über  $\mathbb{K}$ , oder kurz einen  $\mathbb{K}$ -Vektorraum, wenn die folgenden Eigenschaften gelten:

- $(V, +)$  ist eine kommutative Gruppe.

- $,+^{\prime}$  und  $,\cdot^{\prime}$  erfüllen die Distributivgesetze.

$c \cdot (x+y) = c \cdot x + c \cdot y$  für alle  $c \in \mathbb{K}, x, y \in V$

$(c+d) \cdot x = c \cdot x + d \cdot x$  für alle  $c, d \in \mathbb{K}$  und  $x \in V$

- $,\cdot^{\prime}$  ist assoziativ.

$c \cdot (d \cdot x) = (c \cdot d) \cdot x$  für alle  $c, d \in \mathbb{K}, x \in V$

- 1 ist ein neutrales multiplikatives Element.

$1 \cdot x = x$  für alle  $x \in V$

Geht der Körper  $\mathbb{K}$  aus dem Kontext hervor, so wird das Präfix gerne weggelassen und nur noch von einem *Vektorraum* gesprochen.

Die nachstehend aufgeführten Beispiele geben weitere Einblicke in den Aufbau und die Struktur von Vektorräumen.

- Die Menge  $\mathbb{R}^n$  bildet nicht nur für  $n = 2$ , sondern für alle natürlichen Zahlen  $n \geq 1$  einen Vektorraum. Beispielsweise ist  $\mathbb{R}^3$  die mathematische Beschreibung des euklidischen Raums (Abbildung 2.52).

Für  $n \leq 3$  können wir uns die Vektoren aus  $\mathbb{R}^n$  als Richtungspfeile vorstellen, für  $n > 3$  versagt dagegen unsere menschliche Anschauungskraft. Das pure Rechnen in diesen Räumen ist aber nicht schwerer als in  $\mathbb{R}^2$  oder  $\mathbb{R}^3$ , da alle Vektoren dem gleichen systematischen Aufbau folgen: Ein Vektor aus  $\mathbb{R}^n$  wird aus genau  $n$  reellen Zahlen

gebildet, die wir in der gewohnten Spalten- oder Zeilenschreibweise notieren können:

$$\begin{pmatrix} x_0 \\ x_1 \\ \dots \\ x_{n-1} \end{pmatrix} \text{ oder } (x_0 \ x_1 \ \dots \ x_{n-1}) \text{ mit } x_0, x_1, \dots, x_{n-1} \in \mathbb{R}$$

- Ersetzen wir in der Definition von  $\mathbb{R}^n$  die Menge der reellen Zahlen durch einen beliebigen Körper  $\mathbb{K}$ , so erhalten wir die Vektorräume  $\mathbb{K}^n$ . In der Codierungstheorie spielen die Vektorräume über endlichen Körpern eine wichtige Rolle, allen anderen voran die Räume  $\mathbb{Z}_2^n$  (Abbildungen 2.53 und 2.54). Die Vektoren aus diesen Räumen können wir ebenfalls in der vertrauten Spalten- oder Zeilenschreibweise notieren. Zu beachten ist dabei, dass die Vektorkomponenten aus dem Körper  $\mathbb{Z}_2$  stammen und deshalb nur noch die Werte 0 und 1 vorkommen können.
- $\mathbb{K}^{m \times n}$ , die Menge aller Matrizen mit  $m$  Zeilen und  $n$  Spalten über dem Körper  $\mathbb{K}$ , bildet einen  $\mathbb{K}$ -Vektorraum, wenn die Addition zweier Matrizen komponentenweise ausgeführt wird, genauso wie die Multiplikation mit einem Skalar  $c \in \mathbb{K}$ . Zum Beispiel gilt in  $\mathbb{K}^{3 \times 2}$ :

$$\begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \\ a_{20} & a_{21} \end{pmatrix} + \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \\ b_{20} & b_{21} \end{pmatrix} := \begin{pmatrix} a_{00} + b_{00} & a_{01} + b_{01} \\ a_{10} + b_{10} & a_{11} + b_{11} \\ a_{20} + b_{20} & a_{21} + b_{21} \end{pmatrix}$$

$$c \cdot \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \\ a_{20} & a_{21} \end{pmatrix} := \begin{pmatrix} c \cdot a_{00} & c \cdot a_{01} \\ c \cdot a_{10} & c \cdot a_{11} \\ c \cdot a_{20} & c \cdot a_{21} \end{pmatrix}$$

Dass wir einen Vektorraum vor uns haben, lässt sich leicht überprüfen. Zunächst bildet die Menge der  $m \times n$ -Matrizen zusammen mit der Matrixaddition eine kommutative Gruppe. Des Weiteren erfüllt die komponentenweise Multiplikation mit einem Skalar die geforderten Assoziativ- und Distributivgesetze, und die Multiplikation mit dem Skalar 1 verhält sich neutral.

Als Spezialfälle erhalten wir mit  $\mathbb{K}^{n \times 1}$  und  $\mathbb{K}^{1 \times n}$  zwei Vektorräume, die wir schon kennen. Beide sind isomorph zu  $\mathbb{K}^n$  und bringen sogar die uns vertraute Notation hervor. Schreiben wir eine Matrix aus  $\mathbb{K}^{n \times 1}$  in gewohnter Weise auf, so erhalten wir einen Spaltenvektor. Für die Matrizen aus  $\mathbb{K}^{1 \times n}$  gilt etwas ganz Ähnliches. Sie degenerieren zu Vektoren in Zeilenschreibweise.

$$\mathbb{Z}_2^2 = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\}$$

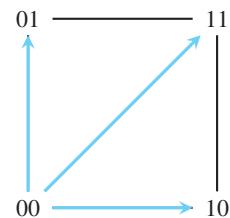


Abb. 2.53: Der Vektorraum  $\mathbb{Z}_2^2$

$$\mathbb{Z}_2^3 = \left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right\}$$

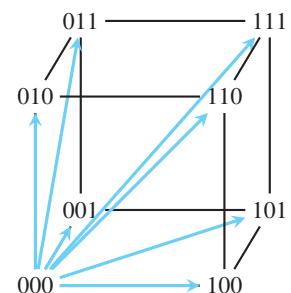


Abb. 2.54: Der Vektorraum  $\mathbb{Z}_2^3$

Hat ein Vektorraum ein endliches Erzeugendensystem, so nennen wir ihn *endlich erzeugbar*. Hierunter fallen alle der oben genannten Beispiele, bis auf den Polynomraum  $\mathbb{K}[x]$ , für den ausschließlich unendliche Erzeugendensysteme existieren.  
Ein solches Erzeugendensystem lässt sich schnell finden:

$$\{1, x, x^2, x^3, x^4, \dots\}$$

Es ist leicht einzusehen, dass der Raum der Polynome von keiner endlichen Menge aufgespannt werden kann. In einer endlichen Menge müssten wir lediglich alle Terme der Form  $x^n$  der Reihe nach durchsehen und uns den größten vorkommenden Exponenten merken. Bezeichnen wir diesen Exponenten mit

$$\max n,$$

so könnten wir mit

$$x^{(\max n)+1}$$

sofort ein Polynom angeben, das sich nicht erzeugen ließe.



■ Weiter oben haben wir dargelegt, dass  $(\mathbb{K}[x], +)$  eine kommutative Gruppe ist. Definieren wir die Multiplikation eines Polynoms mit einem Skalar  $c \in \mathbb{K}$ , wie üblich, als

$$c \cdot (a_n x^n + \dots + a_1 x + a_0) := (ca_n) x^n + \dots + (ca_1) x + (ca_0),$$

so bildet die Menge  $\mathbb{K}[x]$  einen Vektorraum. Auch hier ist leicht einzusehen, dass alle in Definition 2.13 geforderten Eigenschaften erfüllt sind. Die Multiplikation mit einem Skalar ist assoziativ und distributiv, und die Multiplikation der 1 mit einem beliebigen Polynom  $p(x)$  ergibt  $p(x)$ .

Jeder Vektorraum lässt sich durch die Angabe eines *Erzeugendensystems* beschreiben. Hierbei handelt es sich um eine ausgewählte Menge von Vektoren, mit denen sich sämtliche Elemente eines Vektorraums darstellen lassen. Das bedeutet konkret: Ist  $\mathbf{x}$  ein Vektor eines  $\mathbb{K}$ -Vektorraums  $V$  und  $E$  ein Erzeugendensystem von  $V$ , so können wir  $\mathbf{x}$  in der Form

$$\mathbf{x} = \lambda_1 \mathbf{v}_1 + \lambda_2 \mathbf{v}_2 + \dots + \lambda_n \mathbf{v}_n \quad (2.21)$$

schreiben, mit  $\lambda_1, \lambda_2, \dots, \lambda_n \in \mathbb{K}$  und  $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n \in E$ . Die rechte Seite von (2.21) bezeichnen wir als *Linearkombination* von  $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$ .

Für den Vektorraum  $\mathbb{R}^2$  können wir ein Erzeugendensystem sofort angeben:

$$E_1 := \left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\} \quad (2.22)$$

Diese Menge erfüllt unseren Zweck, da wir damit jeden Vektor aus  $\mathbb{R}^2$  mit Leichtigkeit in die geforderte Form (2.21) bringen können. Es ist

$$\mathbf{x} = \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = x_0 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + x_1 \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Da sich ausnahmslos jeder Vektor aus  $\mathbb{R}^2$  als Linearkombination der gewählten Vektoren darstellen lässt, sagen wir, dass diese Vektoren den Raum  $\mathbb{R}^2$  *aufspannen* oder *erzeugen*:



### Definition 2.14 (Erzeugendensystem)

Die Menge  $E \subseteq V$  ist ein *Erzeugendensystem* des Vektorraums  $V$ , wenn sich jeder Vektor  $\mathbf{x} \in V$  als *Linearkombination* endlich vieler Vektoren aus  $E$  aufschreiben lässt:

$$\mathbf{x} = \lambda_1 \cdot \mathbf{v}_1 + \lambda_2 \cdot \mathbf{v}_2 + \dots + \lambda_n \cdot \mathbf{v}_n$$

Vergessen Sie nicht, dass ein Erzeugendensystem durchaus unendlich viele Elemente beinhalten kann, eine Linearkombination aber immer nur aus endlich vielen Summanden besteht.

Sehen wir unser oben gewähltes Beispiel genauer an, so wird schnell klar, dass das betrachtete Erzeugendensystem nicht das einzige ist, das den entsprechenden Vektorraum aufspannt. Den Raum  $\mathbb{R}^2$  können wir genauso gut mit den Vektoren der folgenden Menge erzeugen (Abbildung 2.55 unten):

$$E_2 := \left\{ \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \end{pmatrix} \right\}$$

Ein weiteres Erzeugendensystem ist dieses hier:

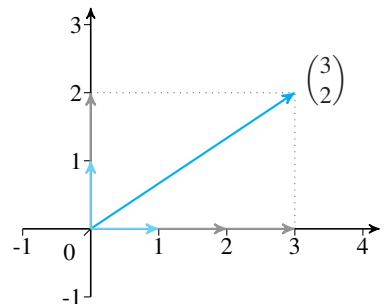
$$E_3 := \left\{ \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \end{pmatrix}, \begin{pmatrix} 4 \\ 2 \end{pmatrix} \right\}$$

Obwohl auch  $E_3$  den Vektorraum  $\mathbb{R}^2$  aufspannt, geht eine wichtige Eigenschaft verloren.  $E_1$  und  $E_2$  sind *minimal*, da kein Erzeugendensystem für  $\mathbb{R}^2$  existiert, das weniger als 2 Vektoren umfasst. Dagegen enthält die Menge  $E_3$  mit  $\binom{2}{1}$  und  $\binom{4}{2}$  zwei Vektoren, die sich als Linearkombination der anderen darstellen lassen. Es ist

$$\begin{pmatrix} 2 \\ 1 \end{pmatrix} = 0 \cdot \begin{pmatrix} -1 \\ 0 \end{pmatrix} + \frac{1}{2} \cdot \begin{pmatrix} 4 \\ 2 \end{pmatrix} \quad \text{und} \quad \begin{pmatrix} 4 \\ 2 \end{pmatrix} = 0 \cdot \begin{pmatrix} -1 \\ 0 \end{pmatrix} + 2 \cdot \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

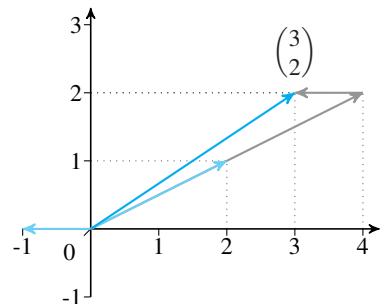
Eine Menge mit dieser Eigenschaft heißt *linear abhängig*. Lässt sich kein Vektor als Linearkombination der anderen darstellen, so sprechen wir von einer *linear unabhängigen* Menge. Die folgende Definition bringt den Unterschied auf elegante Weise zum Ausdruck: Sie führt die Eigenschaft einer Menge, linear abhängig oder linear unabhängig zu sein, auf die Anzahl der Möglichkeiten zurück, den Nullvektor darzustellen.

■ Basis  $E_1 = \left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\}$



$$\begin{pmatrix} 3 \\ 2 \end{pmatrix} = 3 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + 2 \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

■ Basis  $E_2 = \left\{ \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \end{pmatrix} \right\}$



$$\begin{pmatrix} 3 \\ 2 \end{pmatrix} = 2 \begin{pmatrix} 2 \\ 1 \end{pmatrix} + 1 \begin{pmatrix} -1 \\ 0 \end{pmatrix}$$

**Abb. 2.55:** Zwei Erzeugendensysteme des Vektorraums  $\mathbb{R}^2$



### Definition 2.15 (Lineare Unabhängigkeit)

Die Vektoren  $x_1, x_2, \dots, x_n$  eines  $\mathbb{K}$ -Vektorraums heißen *linear unabhängig*, wenn die Gleichung

$$\lambda_1 x_1 + \lambda_2 x_2 + \dots + \lambda_n x_n = \mathbf{0}$$

nur die triviale Lösung  $\lambda_1 = \lambda_2 = \dots = \lambda_n = 0$  besitzt.

- $B_1 = \{M_1, M_2, M_3, M_4\}$  mit

$$M_1 = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \quad M_2 = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

$$M_3 = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, \quad M_4 = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} = \sum_{i=1}^4 \lambda_i M_i$$

mit  $\lambda_1 = a_{00}$

$\lambda_2 = a_{01}$

$\lambda_3 = a_{10}$

$\lambda_4 = a_{11}$

- $B_2 = \{N_1, N_2, N_3, N_4\}$  mit

$$N_1 = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \quad N_2 = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$$

$$N_3 = \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}, \quad N_4 = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} = \sum_{i=1}^4 \lambda_i N_i$$

mit  $\lambda_1 = a_{00}$

$\lambda_2 = a_{01} - a_{00}$

$\lambda_3 = a_{10} - a_{01}$

$\lambda_4 = a_{11} - a_{10}$

**Abb. 2.56:** Zwei Basen des Vektorraums  $\mathbb{R}^{2 \times 2}$ .  $B_1$  ist die *Standardbasis*.

Dass die Menge  $E_3$  linear abhängig ist, können wir jetzt ganz formal begründen. Die Gleichung

$$\lambda_1 \begin{pmatrix} 2 \\ 1 \end{pmatrix} + \lambda_2 \begin{pmatrix} -1 \\ 0 \end{pmatrix} + \lambda_3 \begin{pmatrix} 4 \\ 2 \end{pmatrix} = \mathbf{0}$$

besitzt mit  $\lambda_1 = 2$ ,  $\lambda_2 = 0$  und  $\lambda_3 = -1$  eine nichttriviale Lösung.

Ist eine lineare abhängige Menge von Vektoren gegeben, so lassen sich daraus stets Elemente entfernen, ohne den aufgespannten Vektorraum zu verändern. Beispielsweise können wir in unserer Menge  $E_3$  auf einen der Vektoren  $\begin{pmatrix} 2 \\ 1 \end{pmatrix}$  oder  $\begin{pmatrix} 4 \\ 2 \end{pmatrix}$  verzichten. Beachten Sie dabei, dass wir nur solche Vektoren herausnehmen dürfen, die sich als Linearkombination der verbleibenden Vektoren schreiben lassen. Würden wir aus  $E_3$  den Vektor  $\begin{pmatrix} -1 \\ 0 \end{pmatrix}$  entfernen, so ließen sich nicht mehr alle Vektoren aus  $\mathbb{R}^2$  erzeugen.

Linear unabhängige Erzeugendensysteme sind so wichtig, dass sie einen eigenen Namen tragen:



### Definition 2.16 (Basis)

Es sei  $V$  ein Vektorraum. Eine Teilmenge  $B \subset V$  heißt *Basis* von  $V$ , wenn sie die folgenden beiden Eigenschaften aufweist:

- $B$  erzeugt  $V$ .
- $B$  ist linear unabhängig.

Von den drei betrachteten Erzeugendensystemen von  $\mathbb{R}^2$  waren  $E_1$  und  $E_2$  linear unabhängig und  $E_3$  linear abhängig. Folgerichtig sind  $E_1$  und  $E_2$  Basen des  $\mathbb{R}^2$ , nicht aber  $E_3$ .

Unter den unendlich vielen Basen des  $\mathbb{R}^2$  nimmt die Menge  $E_1$  eine Sonderstellung ein. Sie besitzt als einzige die willkommene Eigenschaft, dass die Werte für  $\lambda_1, \lambda_2, \dots, \lambda_n$  aus Definition 2.14 mit den Vektorkomponenten identisch sind und daher nicht aufwendig ausgerechnet werden müssen. Aufgrund dieser besonderen Eigenschaft wird  $E_1$  als die *Standardbasis* von  $\mathbb{R}^2$  bezeichnet. Auch in vielen anderen Vektorräumen existieren Standardbasen, wie Abbildung 2.56 am Beispiel des Vektorraums  $\mathbb{R}^{2 \times 2}$  belegt.

Basen besitzen die Eigenschaft, dass sie stets die gleiche Anzahl an Elementen enthalten. Das bedeutet für den Raum  $\mathbb{R}^2$ , dass eine Basis stets

2 Elemente umfasst, egal welche Vektoren darin tatsächlich enthalten sind. Für andere Vektorräume gilt das Gleiche. Eine Basis des  $\mathbb{R}^3$  umfasst stets 3 Vektoren und allgemein gilt, dass eine Basis des  $\mathbb{K}^n$  genau  $n$  Elemente enthält. Diese Eigenschaft macht die Mächtigkeit der Basis zu einer abstrakten Kenngröße eines Vektorraums.



### Definition 2.17 (Dimension)

Sei  $B$  eine Basis des Vektorraums  $V$ .

- Ist  $B$  eine endliche Menge, so sprechen wir von einem *endlich dimensionalen Vektorraum* und setzen

$$\dim V := |B|$$

- Ist  $B$  eine unendliche Menge, so sprechen wir von einem *unendlich dimensionalen Vektorraum* und setzen

$$\dim V := \infty$$

## Untervektorräume

Bei den bisher diskutierten Strukturen waren wir häufig in der Lage, in sich abgeschlossene Teilmengen zu identifizieren. Dies führte uns auf direktem Weg zu den Begriffen der Untergruppe, des Unterkörpers und des Unterring. Vektorräume machen hier keine Ausnahme:



### Definition 2.18 (Untervektorraum)

Jede Teilmenge eines  $\mathbb{K}$ -Vektorraums  $V$ , die selbst ein  $\mathbb{K}$ -Vektorraum ist, heißt *Untervektorraum* von  $V$ .

Untervektorräume lassen sich anhand des *Unterraumkriteriums* erkennen. Dieses besagt, dass  $V'$  genau dann ein Untervektorraum von  $V$  ist, wenn

- $V'$  den Nullvektor enthält,  
☞  $\mathbf{0} \in V'$
- die Vektoraddition abgeschlossen ist,  
☞  $x, y \in V' \Rightarrow x + y \in V'$

- die Multiplikation mit Skalaren abgeschlossen ist.

$$\text{☞ } c \in \mathbb{K}, x \in V' \Rightarrow c \cdot x \in V'$$

Wichtige Untervektorräume sind diese hier:

- Ist  $B$  eine Basis des Vektorraums  $\mathbb{K}^n$ , so spannt jede Teilmenge  $B' \subseteq B$  einen Untervektorraum von  $\mathbb{K}^n$  auf. Für  $B' = B$  und  $B' = \emptyset$  erhalten wir die beiden *trivialen Untervektorräume*: den Vektorraum selbst und die Menge  $\{\mathbf{0}\}$ , die ausschließlich den Nullvektor enthält.

Ist  $B'$  eine echte Teilmenge, bestehend aus  $n$  Vektoren, so erhalten wir einen echten Untervektorraum der Dimension  $n$ . In  $\mathbb{R}^3$  lassen sich Untervektorräume intuitiv visualisieren. Entfernen wir aus einer Basis des  $\mathbb{R}^3$  einen Vektor, so spannen die verbleibenden zwei Vektoren eine Ebene auf, die den Ursprung schneidet. Damit können wir uns jeden Untervektorraum der Dimension 2 als eine solche Ebene vorstellen, und auf der anderen Seite gilt, dass jede Ebene durch den Ursprung einen Untervektorraum von  $\mathbb{R}^3$  beschreibt.

Entfernen wir einen weiteren Vektor aus der Basis, so spannt das verbleibende Element eine Gerade durch den Ursprung auf. Umgekehrt gilt, dass jede Gerade durch den Ursprung einen echten Untervektorraum von  $\mathbb{R}^2$  beschreibt, und damit auch einen echten Untervektorraum von  $\mathbb{R}^n$  für  $n > 2$ .

- Polynome

Es sei  $\mathbb{K}[x]_{<n}$  die Menge aller Polynome über dem Körper  $\mathbb{K}$  mit einem Grad kleiner als  $n$ :

$$\mathbb{K}[x]_{<n} := \{p(x) \in \mathbb{K}[x] \mid \deg p(x) < n\}$$

Identifizieren wir den Vektor

$$\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-2} \\ a_{n-1} \end{pmatrix}$$

mit dem Polynom  $a_0x^{n-1} + a_1x^{n-2} + \dots + a_{n-2}x + a_{n-1}$ , so entsteht eine Eins-zu-eins-Beziehung zwischen dem Vektorraum aus  $\mathbb{K}^n$  und den Polynomen aus  $\mathbb{K}[x]_{<n}$ . Die Addition zweier Polynome aus  $\mathbb{K}[x]_{<n}$  entspricht der Addition zweier Vektoren aus  $\mathbb{K}^n$  und die Multiplikation eines Polynoms mit einem Skalar führt zu dem gleichen Ergebnis wie die Multiplikation eines Vektors mit einem Skalar. Das

bedeutet, dass wir mit  $\mathbb{K}[x]_{<n}$  einen  $n$ -dimensionalen Vektorraum über der Menge  $\mathbb{K}$  vor uns haben.

Mit dem erworbenen Wissen über endliche Körper können wir aus dem letzten Beispiel ein interessantes Ergebnis ableiten. Da wir jedes Element eines endlichen Körpers mit  $p^n$  Elementen als ein Polynom aus der Menge  $\mathbb{Z}_p[x]_{<n}$  ansehen dürfen, gilt der folgende Satz:


**Satz 2.10**

$\mathbb{F}_{p^n}$  ist ein  $\mathbb{Z}_p$ -Vektorraum der Dimension  $n$ .

An dieser Stelle wollen wir unser Augenmerk auf wichtige Spezialfälle der Vektorräume  $\mathbb{K}^n$  richten: die Räume  $\mathbb{Z}_2^n$ . Für  $n = 3$  erhalten wir den weiter oben diskutierten Vektorraum aus Abbildung 2.54. Da in  $\mathbb{Z}_2^3$  nur endlich viele Elemente enthalten sind, können wir seine Untervektorräume der Reihe nach auflisten. Sehen wir von den trivialen Lösungen ab (diese sind der Vektorraum selbst sowie die Menge  $\{\mathbf{0}\}$ ), so bleiben die 12 Unterräume aus Abbildung 2.57 übrig.

Die eindimensionalen Untervektorräume lassen sich durch Geraden veranschaulichen, die vom Ursprung ausgehen und zu einer der anderen Würfecken führen. Ersetzen wir eine Gerade durch eine Ebene, so entsteht ein zweidimensionaler Untervektorraum. Damit scheint im Vektorraum  $\mathbb{Z}_2^n$  der gleiche Zusammenhang zu gelten wie im vertrauten euklidischen Raum  $\mathbb{R}^3$ . Eindimensionale Unterräume werden durch Geraden und zweidimensionale Unterräume durch Ebenen gebildet.

Sehen wir genauer hin, so scheint es mit der Menge

$$\left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \right\} \quad (2.23)$$

einen Untervektorraum zu geben, der sich dieser Regel widersetzt. Dass wir uns auch diesen Raum als eine Ebene vorstellen können, macht Abbildung 2.58 deutlich. Die geometrische Interpretation gelingt, weil wir jeden Unterraum von  $\mathbb{Z}_2^3$  als die Projektion eines Unterraums von  $\mathbb{Z}^3$  ansehen können. Ersetzen wir nämlich in einem beliebigen Unterraum von  $\mathbb{Z}^3$  die Vektoren entsprechend dem Schema

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} \mapsto \begin{pmatrix} x_0 \bmod 2 \\ x_1 \bmod 2 \\ x_2 \bmod 2 \end{pmatrix},$$

Untervektorraum $U_1$	Untervektorraum $U_2$	Untervektorraum $U_3$	Untervektorraum $U_4$
$\left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \right\}$	$\left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \right\}$	$\left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \right\}$	$\left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right\}$
Untervektorraum $U_5$	Untervektorraum $U_6$	Untervektorraum $U_7$	Untervektorraum $U_8$
$\left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \right\}$	$\left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \right\}$	$\left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \right\}$	$\left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \right\}$
Untervektorraum $U_9$	Untervektorraum $U_{10}$	Untervektorraum $U_{11}$	Untervektorraum $U_{12}$
$\left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \right\}$	$\left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \right\}$	$\left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \right\}$	$\left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right\}$

**Abb. 2.57:** Die nichttrivialen Untervektorräume von  $\mathbb{Z}_2^3$ . Von den 12 abgebildeten Untervektorräumen sind die Räume  $U_5$  und  $U_{12}$  besonders wichtig. In Abschnitt 3.4.3 werden wir offenlegen, welche Rolle sie in der Codierungstheorie spielen.

so erhalten wir einen der Unterräume aus Abbildung 2.57. Auch den Vektorraum (2.23) können wir auf diese Weise erhalten. Er ist die Projektion des Untervektorräums, der von den Vektoren

$$\left\{ \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \right\} \quad (2.24)$$

aufgespannt wird.

## 2.5.1 Generatormatrizen

Untervektorräume von  $\mathbb{K}^n$  besitzen die Eigenschaft, dass wir ihre Elemente mit einer *Generatormatrix* erzeugen können. Um den Aufbau einer solchen Matrix zu verstehen, müssen wir uns lediglich daran erinnern, dass sich jedes Element  $x$  eines  $n$ -dimensionalen  $\mathbb{K}$ -Vektorraums als eine Linearkombination von  $n$  Basisvektoren aufschreiben lässt:

$$x = \lambda_1 v_1 + \lambda_2 v_2 + \dots + \lambda_n v_n \quad (2.25)$$

Die Faktoren  $\lambda_1, \lambda_2, \dots, \lambda_n$  sind Elemente des Skalarkörpers  $\mathbb{K}$  und für jeden vorgelegten Vektor  $x$  eindeutig bestimmt.

Schreiben wir diese Gleichung für den Untervektorraum (2.23) auf, so erhalten wir mit

$$x = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \lambda_1 \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} + \lambda_2 \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

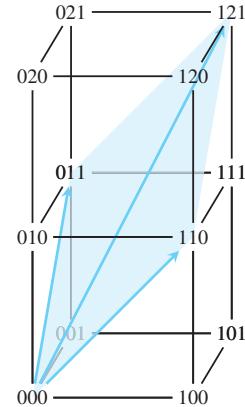
eine Gleichung, die sich genauso gut mithilfe einer Matrix formulieren lässt:

$$x = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix}$$

Verwenden wir anstelle der Spaltenschreibweise die Zeilenschreibweise, dann lautet diese Gleichung so:

$$x = (x_0 \ x_1 \ x_2) = (\lambda_1 \ \lambda_2) \cdot \underbrace{\begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}}_{\text{Generatormatrix } G}$$

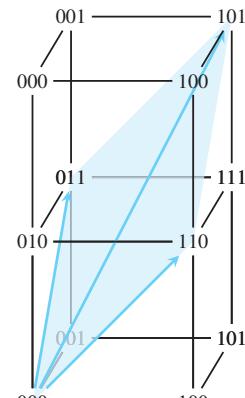
Die Matrix  $G$  wird als *Generatormatrix* bezeichnet, da die Multiplikation mit den möglichen Kombinationen von  $\lambda_1$  und  $\lambda_2$  sämtliche Elemente des Untervektorräums hervorbringt (Abbildung 2.59).



Ausschnitt aus dem Vektorraum

$$\left\{ \lambda_0 \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} + \lambda_1 \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \mid \lambda_0, \lambda_1 \in \mathbb{Z} \right\}$$

Projektion



$$\left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \right\}$$

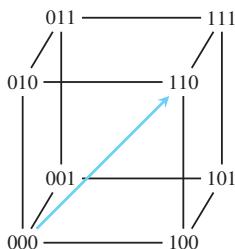
**Abb. 2.58:** Projizieren wir einen Untervektorraum von  $\mathbb{Z}^3$  auf die Menge  $\mathbb{Z}_2^3$ , so erhalten wir einen Untervektorraum von  $\mathbb{Z}_2^3$ . Auf diese Weise können wir jeden zweidimensionalen Untervektorraum von  $\mathbb{Z}_2^3$  mit einer Ebene identifizieren und damit geometrisch deuten.

■  $\lambda_1 = 0, \lambda_2 = 0$

$$(0 \ 0) \cdot \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} = (0 \ 0 \ 0)$$

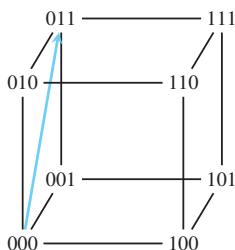
■  $\lambda_1 = 0, \lambda_2 = 1$

$$(0 \ 1) \cdot \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} = (1 \ 1 \ 0)$$



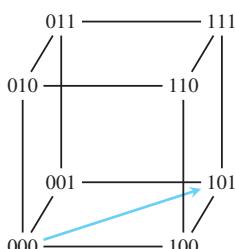
■  $\lambda_1 = 1, \lambda_2 = 0$

$$(1 \ 0) \cdot \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} = (0 \ 1 \ 1)$$



■  $\lambda_1 = 1, \lambda_2 = 1$

$$(1 \ 1) \cdot \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} = (1 \ 0 \ 1)$$



**Abb. 2.59:** Durch die Multiplikation der Generatormatrix mit den möglichen Kombinationen von  $\lambda_1$  und  $\lambda_2$  lassen sich alle Elemente des Vektorraums erzeugen.

Der Aufbau der Generatormatrix ist damit geklärt. Wir erhalten sie, indem wir die Basisvektoren zeilenweise untereinander schreiben.

Da die meisten Vektorräume unendlich viele Basen besitzen, ist die Generatormatrix nur selten eindeutig bestimmt. Dennoch können wir sie in eine normierte Form bringen, indem wir in den linken Spalten die Werte der Einheitsmatrix erzeugen. Für unser Beispiel sieht die Matrix dann so aus:

$$G = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

Wir bezeichnen  $G$  als eine Generatormatrix in *systematischer* oder *reduzierter Form*. Abbildung 2.60 zeigt, dass der erzeugte Untervektorraum unverändert bleibt.

Die Generatormatrix enthält eine Fülle von Informationen, und bereits an ihrer äußeren Form lassen sich wichtige Kenngrößen des erzeugten Vektorraum ablesen: Eine Matrix aus der Menge  $\mathbb{K}^{m \times n}$  erzeugt einen  $m$ -dimensionalen Untervektorraum von  $\mathbb{K}^n$ .

Als Beispiel betrachten wir die folgende Generatormatrix, die uns in diesem Buch gleich mehrfach wiederbegegnen wird:

$$G = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \quad (2.26)$$

$G$  besitzt 4 Zeilen und 7 Spalten und definiert einen 4-dimensionalen Untervektorraum von  $\mathbb{Z}_2^7$ .

## 2.5.2 Orthogonalräume

In jedem Vektorraum sind mindestens zwei Verknüpfungen definiert: die Addition zweier Vektoren sowie die Multiplikation eines Vektors mit einem Skalar. In vielen Vektorräumen existiert eine zusätzliche Verknüpfung,  $\cdot$ :  $V \times V \rightarrow \mathbb{K}$ , die zwei Vektoren auf ein Element des Skalar-Körpers  $\mathbb{K}$  abbildet.

Die Operation  $\cdot$  wird als *inneres Produkt* bezeichnet und ist im Vektorraum  $\mathbb{K}^n$  beispielsweise so definiert:

$$\mathbf{x} \cdot \mathbf{y} := \sum_{i=0}^{n-1} x_i \cdot y_i$$

Betrachten wir die speziellen Räume  $\mathbb{R}^2$  oder  $\mathbb{R}^3$ , so lässt sich das innere Produkt über die äquivalente Formel

$$\mathbf{x} \cdot \mathbf{y} = |\mathbf{x}| \cdot |\mathbf{y}| \cdot \cos \angle(\mathbf{x}, \mathbf{y}) \quad (2.27)$$

berechnen, wobei  $|\mathbf{x}|$  und  $|\mathbf{y}|$  die Längen der Vektoren  $\mathbf{x}$  bzw.  $\mathbf{y}$  und  $\angle(\mathbf{x}, \mathbf{y})$  den Winkel zwischen  $\mathbf{x}$  und  $\mathbf{y}$  beschreibt (Abbildung 2.61). Formel (2.27) zeigt, wie wir das inneren Produkt geometrisch deuten können. Aus ihr folgt beispielsweise, dass zwei Vektoren genau dann orthogonal zueinander sind, wenn das Produkt  $\mathbf{x} \cdot \mathbf{y}$  den Wert 0 ergibt:

$$\mathbf{x} \perp \mathbf{y} \Leftrightarrow \mathbf{x} \cdot \mathbf{y} = 0$$

Damit sind wir in der Lage, den zu einem Vektorraum  $V \subseteq \mathbb{K}^n$  gehörenden *Orthogonalraum*  $V^\perp$  zu definieren:

$$V^\perp := \{\mathbf{y} \in \mathbb{K}^n \mid \mathbf{x} \cdot \mathbf{y} = 0 \text{ für alle } \mathbf{x} \in V\}$$

In Worten ausgedrückt umfasst  $V^\perp$  genau jene Vektoren aus  $\mathbb{K}^n$ , die auf sämtlichen Vektoren von  $V$  senkrecht stehen.

Wir wollen versuchen, mehr über die Struktur des Orthogonalraums zu erfahren. Zunächst halten wir fest, dass  $V^\perp$  niemals leer sein kann, da der Nullvektor  $\mathbf{0}$  auf allen Vektoren senkrecht steht und damit in  $V^\perp$  enthalten sein muss. Ferner gilt für zwei Vektoren  $\mathbf{x}, \mathbf{y} \in V^\perp$  und jedes Körperelement  $c \in \mathbb{K}$ , dass auch der Summenvektor  $\mathbf{x} + \mathbf{y}$  sowie der skalierte Vektor  $c \cdot \mathbf{x}$  auf den Elementen von  $V$  senkrecht stehen. Das bedeutet, dass die Menge  $V^\perp$  das Unterraumkriterium erfüllt und damit selbst ein Vektorraum ist.

Mithilfe der Generatormatrix eines Vektorraums  $V$  sind wir in der Lage, den Orthogonalraum  $V^\perp$  sogar systematisch zu berechnen. Die folgenden Eigenschaften weisen uns den Weg:

- Ein Vektor ist genau dann zu allen Vektoren eines Vektorraums  $V$  orthogonal, wenn er zu den Vektoren einer Basis orthogonal ist.
- Die Zeilen einer Generatormatrix bilden eine Basis von  $V$ .

Das bedeutet nichts anderes, als dass die Elemente von  $V^\perp$  die Lösungen der nachstehenden Gleichung sind:

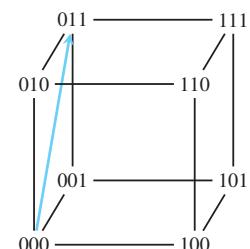
$$G \cdot \mathbf{y}^T = \mathbf{0} \quad (2.28)$$

■  $\lambda_1 = 0, \lambda_2 = 0$

$$(0 \ 0) \cdot \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} = (0 \ 0 \ 0)$$

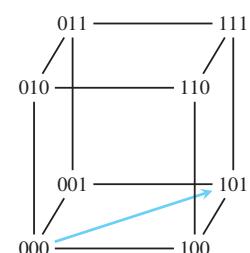
■  $\lambda_1 = 0, \lambda_2 = 1$

$$(0 \ 1) \cdot \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} = (0 \ 1 \ 1)$$



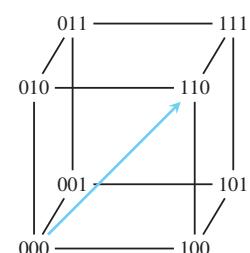
■  $\lambda_1 = 1, \lambda_2 = 0$

$$(1 \ 0) \cdot \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} = (1 \ 0 \ 1)$$



■  $\lambda_1 = 1, \lambda_2 = 1$

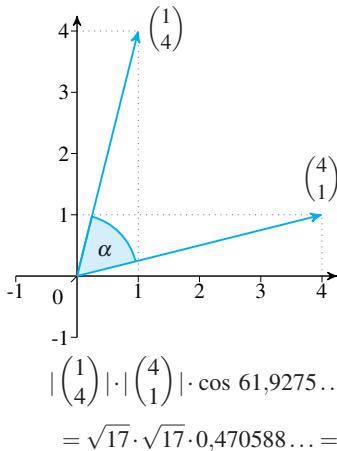
$$(1 \ 1) \cdot \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} = (1 \ 1 \ 0)$$



**Abb. 2.60:** Ist in der Generatormatrix die Einheitsmatrix enthalten, so sprechen wir von einer Generatormatrix in *systematischer* oder *reduzierter Form*.

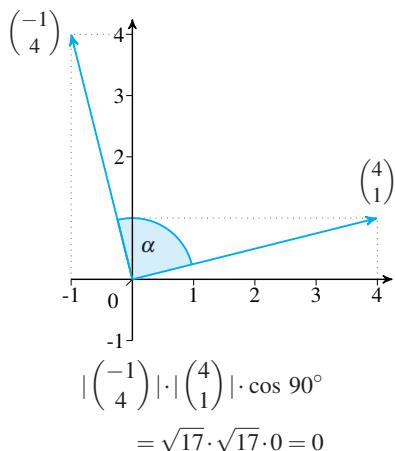
## Beispiel 1

$$\begin{pmatrix} 1 \\ 4 \end{pmatrix} \cdot \begin{pmatrix} 4 \\ 1 \end{pmatrix} = 1 \cdot 4 + 4 \cdot 1 = 8$$



## Beispiel 2

$$\begin{pmatrix} -1 \\ 4 \end{pmatrix} \cdot \begin{pmatrix} 4 \\ 1 \end{pmatrix} = -1 \cdot 4 + 4 \cdot 1 = 0$$



**Abb. 2.61:** Berechnung des inneren Produkts über die Längen und den relativen Winkel zweier Vektoren

Für unseren Beispielvektorraum (2.26) lautet diese Gleichung so:

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_6 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Als Erstes ersetzen wir die Generatormatrix durch ihre systematische Form:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_6 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Aus dieser Darstellung können wir die Lösungen direkt ablesen:

$$\begin{aligned} y_0 &= y_5 + y_6 \\ y_1 &= y_4 + y_6 \\ y_2 &= y_4 + y_5 \\ y_3 &= y_4 + y_5 + y_6 \end{aligned}$$

Als Ergebnis haben wir einen dreidimensionalen Lösungsraum erhalten, der durch die drei unabhängigen Variablen  $y_4, y_5, y_6$  und die vier abhängigen Variablen  $y_0, y_1, y_2, y_3$  gegeben ist.

In anderer Schreibweise können wir das Ergebnis so ausdrücken:

$$\left\{ \begin{pmatrix} y_5 + y_6 \\ y_4 + y_6 \\ y_4 + y_5 \\ y_4 + y_5 + y_6 \\ y_4 \\ y_5 \\ y_6 \end{pmatrix} \mid y_4, y_5, y_6 \in \mathbb{Z}_2 \right\} \quad (2.29)$$

Dies ist der Orthogonalraum des von  $G$  erzeugten Vektorraums.

Aus der Art und Weise, wie sich  $V^\perp$  berechnen lässt, können wir einen Rückschluss auf die Dimension des Orthogonalraums ziehen. Besteht die Generatormatrix  $G$  aus  $m$  Zeilen und  $n$  Spalten, so ist die Lösung der Gleichung (2.28) durch  $n - m$  freie Variablen  $v_m, \dots, v_{n-1}$  bestimmt. Indem wir die freien Variablen nacheinander mit den Wertekombinationen  $(1, 0, 0, \dots, 0), (0, 1, 0, \dots, 0), \dots$  belegen, können wir daraus  $n - m$  linear unabhängige Vektoren generieren.

Alles in allem gilt damit der folgende Satz:


**Satz 2.11**

Der Orthogonalraum  $V^\perp$  eines  $m$ -dimensionalen Vektorraums  $V \subset \mathbb{K}^n$  ist ein Untervektorraum von  $\mathbb{K}^n$  der Dimension  $n - m$ .

Der Orthogonalraum (2.29) ist also ein dreidimensionaler Unterraum, der von der folgenden Basis erzeugt wird:

$$\left\{ \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \right\}$$

Mit dieser Basis in Händen können wir für den Orthogonalraum  $V^\perp$  ebenfalls eine Generatormatrix angeben:

$$G^\perp = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \quad (2.30)$$

Vergleichen wir die Matrix  $G^\perp$  mit der systematischen Form von  $G$ , so wird deutlich, dass wir  $G^\perp$  daraus direkt ableiten können. Wir müssen lediglich die Spalten, die rechts neben der Einheitsmatrix stehen, als Zeilen notieren und das Ergebnis anschließend um die Einheitsmatrix ergänzen (Abbildung 2.62).

Die Generatormatrix von  $V^\perp$  ist für das Arbeiten mit dem Vektorraum  $V$  von praktischem Nutzen, denn mit ihr können wir feststellen, ob ein Vektor zu  $V$  gehört oder nicht. Da ein Vektor  $x$  genau dann zu  $V$  gehört, wenn er orthogonal zu allen Vektoren aus  $V^\perp$  ist, gilt die folgende Beziehung:

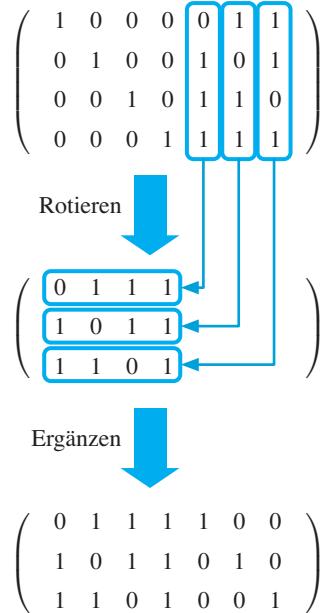
$$x \in V \Leftrightarrow G^\perp \cdot x^T = \mathbf{0} \quad (2.31)$$

Aufgrund dieser Eigenschaft besitzen die Generatormatrizen des Orthogonalraums  $V^\perp$  einen zusätzlichen Namen: Sie werden als *Kontrollmatrizen* von  $V$  bezeichnet.

Aufgrund der Beziehungen  $V^{\perp\perp} = V$  und  $G^{\perp\perp} = G$  können wir (2.31) auch so schreiben:

$$x \in V^\perp \Leftrightarrow G \cdot x^T = \mathbf{0}$$

Die Eigenschaft, eine Kontrollmatrix zu sein, ist offenbar symmetrisch: Über die Matrizen  $G^\perp$  und  $G$  kontrollieren sich die Vektorräume  $V$  und  $V^\perp$  gegenseitig.



**Abb. 2.62:** Liegt die Generatormatrix eines Vektorraums  $V$  in systematischer Form vor, so lässt sie sich auf direktem Weg in eine Generatormatrix des Orthogonalraums  $V^\perp$  umformen.

## 2.6 Übungsaufgaben

**Aufgabe 2.1**

Welche Kongruenzen sind richtig?

**Webcode 2048**

- a)  $7 \equiv 1 \pmod{3}$
- c)  $16 \equiv 2 \pmod{2}$
- e)  $16 \equiv 2 \pmod{2}$
- g)  $0 \equiv -7 \pmod{7}$
- b)  $1 \equiv 16 \pmod{2}$
- d)  $21 \equiv 14 \pmod{7}$
- f)  $0 \equiv 7 \pmod{7}$
- h)  $0 \equiv 0 \pmod{1}$

**Aufgabe 2.2**

**Webcode 2051**

Bestimmen Sie für die nachstehenden Zahlen den größten gemeinsamen Teiler (ggT). Benutzen Sie hierzu den optimierten euklidischen Algorithmus aus Abbildung 2.8. Spielt die Reihenfolge der Operanden zu Beginn der Berechnung eine Rolle?

$$\begin{array}{lll}
 \text{ggT}(60, 38) & \text{ggT}(16, 26) & \text{ggT}(8, 13) \\
 = \text{ggT}(\quad, \quad) & = \text{ggT}(\quad, \quad) & = \text{ggT}(\quad, \quad) \\
 = \text{ggT}(\quad, \quad) & = \text{ggT}(\quad, \quad) & = \text{ggT}(\quad, \quad) \\
 = \text{ggT}(\quad, \quad) & = \text{ggT}(\quad, \quad) & = \text{ggT}(\quad, \quad) \\
 = \text{ggT}(\quad, \quad) & = \text{ggT}(\quad, \quad) & = \text{ggT}(\quad, \quad) \\
 = \text{ggT}(\quad, \quad) & = \text{ggT}(\quad, \quad) & = \text{ggT}(\quad, \quad) \\
 = \text{ggT}(\quad, \quad) & = \text{ggT}(\quad, \quad) & = \text{ggT}(\quad, \quad) \\
 = \text{ggT}(\quad, \quad) & = \text{ggT}(\quad, \quad) & = \text{ggT}(\quad, \quad) \\
 = \quad & = \quad & = \quad
 \end{array}$$

**Aufgabe 2.3**

- a) Bestimmen Sie die nachstehenden Mengen:

**Webcode 2083**

$$\mathbb{Z}_{10}^* = \{\quad, \quad, \quad, \quad\}$$

$$\mathbb{Z}_{11}^* = \{\quad, \quad, \quad, \quad, \quad, \quad, \quad, \quad, \quad, \quad\}$$

$$\mathbb{Z}_{12}^* = \{\quad, \quad, \quad, \quad\}$$

$$\mathbb{Z}_{13}^* = \{\quad, \quad, \quad\}$$

- b) Beweisen Sie, dass die Mengen  $\mathbb{Z}_m^*$  paarweise verschieden sind.

Bestimmen Sie das multiplikative Inverse von ...

- a) 3 in  $\mathbb{Z}_4^*$       b) 3 in  $\mathbb{Z}_5^*$       c) 4 in  $\mathbb{Z}_5^*$       d) 3 in  $\mathbb{Z}_7^*$       e) 6 in  $\mathbb{Z}_7^*$

**Aufgabe 2.4**


**Webcode**  
**2122**

Welche der nachstehenden Paare bilden eine (kommutative) Gruppe?

- a)  $(\mathbb{N}, +)$       c)  $(\mathbb{Z}, +)$       e)  $(\mathbb{Q}, +)$       g)  $(\mathbb{R}, +)$       i)  $(\mathbb{C}, +)$   
 b)  $(\mathbb{N} \setminus \{0\}, \cdot)$       d)  $(\mathbb{Z} \setminus \{0\}, \cdot)$       f)  $(\mathbb{Q} \setminus \{0\}, \cdot)$       h)  $(\mathbb{R} \setminus \{0\}, \cdot)$       j)  $(\mathbb{C} \setminus \{0\}, \cdot)$

**Aufgabe 2.5**


**Webcode**  
**2142**

In dieser Aufgabe geht es um die Hieroglyphen-Menge

$$M := \{\text{Egyptian Hieroglyphs}\}.$$

Mit „ $\circ_1$ “ bis „ $\circ_4$ “ sind auf  $M$  vier verschiedene Verknüpfungen definiert:

$\circ_1$						

$\circ_2$						

$\circ_3$						

$\circ_4$						

Für welche Werte von  $i$  bildet  $(M, \circ_i)$  eine (kommutative) Gruppe?

**Aufgabe 2.7****Webcode  
2212**

In Abschnitt 2.3.1 haben wir gezeigt, dass  $(\mathbb{Z}_m^*, \cdot)$  für alle natürlichen Zahlen  $m \geq 2$  eine Gruppe bildet, wenn die Multiplikation modulo  $m$  durchgeführt wird. Nachstehend sehen Sie die Verknüpfungstabelle für den Fall  $m = 15$ :

.	1	2	4	7	8	11	13	14
1	1	2	4	7	8	11	13	14
2	2	4	8	14	1	7	11	13
4	4	8	1	13	2	14	7	11
7	7	14	13	4	11	2	1	8
8	8	1	2	11	4	13	14	7
11	11	7	14	2	13	1	8	4
13	13	11	7	1	14	8	4	2
14	14	13	11	8	7	4	2	1

Wie viele Untergruppen können Sie in der Verknüpfungstabelle erkennen?

**Aufgabe 2.8****Webcode  
2248**

Das Untergruppenkriterium aus Abschnitt 2.3.1 werden Sie in der Literatur manchmal in einer anderen Form wiederfinden. In einer alternativen Formulierung besagt es, dass  $M'$  genau dann eine Untergruppe von  $M$  ist, wenn

- das neutrale Element enthalten ist,   $e \in M'$
- die Verknüpfung  $\circ$  auf  $M'$  abgeschlossen ist,   $x, y \in M' \Rightarrow x \circ y \in M'$
- zu jedem Element aus  $M'$  ein inverses Element in  $M'$  existiert.   $x \in M' \Rightarrow x^{-1} \in M'$

Zeigen Sie, dass es sich hierbei um ein äquivalentes Kriterium handelt.

**Aufgabe 2.9****Webcode  
2283**

Sei  $M$  eine Gruppe und  $N$  eine endliche Untergruppe von  $M$ . Für jedes Element  $g \in N$  nennen wir die Menge

$$gN := \{gx \mid x \in N\}$$

eine Nebenklasse von  $N$ . Zeigen Sie,

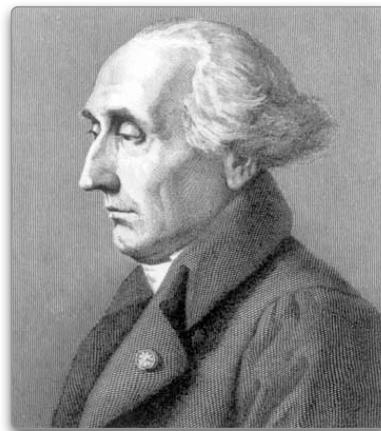
- dass jede Nebenklasse von  $N$  genauso viele Elemente enthält wie  $N$  und
- dass die Nebenklassen von  $N$  paarweise disjunkt sind.

In diesem Kapitel haben Sie zahlreiche Beispiele für endliche Gruppen und deren Untergruppen kennen gelernt. Vielleicht ist Ihnen aufgefallen, dass zwischen der Anzahl der Elemente einer Gruppe und der Anzahl der Elemente ihrer Untergruppen ein interessanter Zusammenhang besteht. Für alle betrachteten Beispiele galt, dass die Anzahl der Elemente einer Untergruppe die Anzahl der Elemente der Gruppe teilt.

Dass wir dieses Phänomen in allen endlichen Gruppen beobachten können, ist der Inhalt des *Satzes von Lagrange*, eines wichtigen Theorems der Gruppentheorie. Der italienische Mathematiker Joseph-Louis de Lagrange hat ihn im Jahr 1770 publiziert, allerdings in einer völlig anderen Formulierung [54, 77]. Die Begriffe der Gruppe und der Untergruppe hielten erst viel später Einzug in die Mathematik.

Beweisen Sie den Satz von Lagrange, indem Sie auf die Ergebnisse der vorherigen Teilaufgabe zurückgreifen.

Joseph-Louis de Lagrange  
(1736 – 1813)



Welche der folgenden Tabellen geben die additive und die multiplikative Verknüpfung eines Körpers bzw. eines Rings wieder?

$+$	○	○	○	$\cdot$	○	○
○	○	○	○	○	○	○
○	○	○	○	○	○	○

$+$	○	○	○	$\cdot$	○	○
○	○	○	○	○	○	○
○	○	○	○	○	○	○

$+$	○	○	○	$\cdot$	○	○
○	○	○	○	○	○	○
○	○	○	○	○	○	○

$+$	○	○	○	$\cdot$	○	○
○	○	○	○	○	○	○
○	○	○	○	○	○	○

$+$	○	○	○	$\cdot$	○	○
○	○	○	○	○	○	○
○	○	○	○	○	○	○

$+$	○	○	○	$\cdot$	○	○
○	○	○	○	○	○	○
○	○	○	○	○	○	○

### Aufgabe 2.10



Webcode

2350

### Aufgabe 2.11



Webcode

2358

---

**Aufgabe 2.12** Auf der Menge  $\mathbb{R}^2$  sei die Multiplikation  $, \cdot'$  folgendermaßen definiert:



**Webcode  
2370**

$$(x_1, y_1) \cdot (x_2, y_2) := (x_1 \cdot x_2 - y_1 \cdot y_2, x_1 \cdot y_2 + y_1 \cdot x_2)$$

- a) Welche besondere Rolle spielen die Elemente  $(1,0)$  und  $\left(\frac{x}{x^2+y^2}, \frac{-y}{x^2+y^2}\right)$ ?
- b) Ist  $(\mathbb{R}^2, +, \cdot')$  ein Körper?

---

**Aufgabe 2.13** Welche der folgenden Rechenregeln sind auf der Menge der komplexen Zahlen gültig?



**Webcode  
2410**

- a)  $(x_2 + y_2i + x_3 + y_3i) \cdot (x_1 + y_1i) = (x_2 + y_2i) \cdot (x_1 + y_1i) + (x_3 + y_3i) \cdot (x_1 + y_1i)$
- b)  $(x_1 + y_1i) \cdot 0 = 0 \cdot (x_1 + y_1i) = 0$
- c)  $(x_1 + y_1i) \cdot (- (x_2 + y_2i)) = (- (x_1 + y_1i)) \cdot (x_2 + y_2i) = - ((x_1 + y_1i) \cdot (x_2 + y_2i))$
- d) Aus  $(x_1 + y_1i) \cdot (x_2 + y_2i) = 0$  folgt  $x_1 + y_1i = 0$  oder  $x_2 + y_2i = 0$

---

**Aufgabe 2.14**



**Webcode  
2442**

Mit  $V$  sei eine nichtleere Menge und mit  $, \sqcap' : V \times V \rightarrow V$  und  $, \sqcup' : V \times V \rightarrow V$  seien zwei Verknüpfungen auf dieser Menge gegeben. Das Tripel  $(V, \sqcap, \sqcup)$  ist eine *boolesche Algebra*, wenn die folgenden vier *Huntington'schen Axiome* erfüllt sind:

■ Kommutativgesetze

$$\begin{aligned} x \sqcap y &= y \sqcap x \\ x \sqcup y &= y \sqcup x \end{aligned}$$

■ Distributivgesetze

$$\begin{aligned} x \sqcap (y \sqcup z) &= (x \sqcap y) \sqcup (x \sqcap z) \\ x \sqcup (y \sqcap z) &= (x \sqcup y) \sqcap (x \sqcup z) \end{aligned}$$

■ Neutrale Elemente

Es existieren  $1, 0$  mit

$$\begin{aligned} x \sqcap 1 &= x \\ x \sqcup 0 &= x \end{aligned}$$

■ Inverse Elemente

Für alle  $x$  existiert ein  $x^{-1}$  mit

$$\begin{aligned} x \sqcap x^{-1} &= 0 \\ x \sqcup x^{-1} &= 1 \end{aligned}$$

Welche der folgenden Aussagen sind richtig?

- a)  $(V, \sqcup)$  ist eine Gruppe.
- b)  $(V, \sqcap)$  ist eine Gruppe.
- c)  $(V, \sqcup, \sqcap)$  ist ein Körper.
- d)  $(V, \sqcup, \sqcap)$  ist ein Ring.

Auf Seite 102 haben wir die Charakteristik eines Körpers definiert. Hinter diesem Begriff verbarg sich die kleinste Zahl  $c$  mit

$$\underbrace{1 + 1 + 1 + \dots + 1}_{c\text{-mal}} = 0.$$

**Aufgabe 2.15**
**Webcode  
2447**

- a) Bestimmen Sie die Charakteristik des Polynomkörpers  $\mathbb{Z}_3[x]_{x^2+1}$ .

- b) Eine interessante Beobachtung ist diese hier: Wenn wir nicht die 1, sondern ein beliebiges Körperelement  $c$ -mal aufaddieren, erhalten wir ebenfalls die 0 zurück. Verifizieren Sie die Aussage für den oben eingeführten Körper, indem Sie die nachstehende Tabelle ergänzen:

$p(x)$	$p(x) + p(x)$	$p(x) + p(x) + p(x)$
0		
1		
2		
$x$		
$x + 1$		
$x + 2$		
$2x$		
$2x + 1$		
$2x + 2$		

- c) Erklären Sie, warum wir diese Beobachtung in jedem Körper machen können.

$(\mathbb{Z}, +, \cdot)$  und  $(\mathbb{Q}, +, \cdot)$  sind kommutative Ringe mit 1. Welche Aussagen sind richtig?

- |   |   |
|---|---|
| a) $\mathbb{Z}$ ist ein Unterring von $\mathbb{Q}$ .  | d) $2\mathbb{Z}$ ist ein Ideal von $\mathbb{Z}$ .         |
| b) $\mathbb{Z}$ ist ein Ideal von $\mathbb{Q}$ .      | e) $2\mathbb{Z} + 1$ ist ein Unterring von $\mathbb{Z}$ . |
| c) $2\mathbb{Z}$ ist ein Unterring von $\mathbb{Z}$ . | f) $2\mathbb{Z} + 1$ ist ein Ideal von $\mathbb{Z}$ .     |

**Aufgabe 2.16**
**Webcode  
2517**

Hinweis: Die Menge  $2\mathbb{Z}$  bezeichnet die Menge der geraden ganzen Zahlen und  $2\mathbb{Z} + 1$  die Menge der ungeraden ganzen Zahlen.

**Aufgabe 2.17****Webcode  
2556**

Betrachten Sie die folgende alternative Definition eines Ringes mit 1:

Eine Menge  $M$  mit den Verknüpfungen  $,$   $+^{'}$  und  $,$   $\cdot^{'}$ , geschrieben als  $(M, +, \cdot)$ , heißt *Ring mit 1*, wenn Folgendes gilt:

- $(M, +)$  ist eine Gruppe.
- $(M, \cdot)$  ist eine Halbgruppe mit einem neutralen Element.
- Für alle  $x, y, z \in M$  ist  $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$  und  $(y + z) \cdot x = (y \cdot x) + (z \cdot x)$ .

Im Gegensatz zu Definition 2.8 fehlt hier die Forderung, dass  $(M, +)$  kommutativ ist. Ist die Definition falsch?

**Aufgabe 2.18****Webcode  
2569**

Für welche Werte von  $y$  bildet

$$(\{p \mid p \in \mathbb{K}[x] \wedge p(0) = y\}, +, \cdot)$$

einen Unterring oder ein Ideal von  $\mathbb{K}[x]$ ?

**Aufgabe 2.19****Webcode  
2622**

Eine algebraische Struktur heißt *nullteilerfrei*, wenn die Gleichung  $x \cdot y = 0$  nur dann lösbar ist, falls einer der beiden Faktoren gleich 0 ist, wenn also gilt:

$$x \cdot y = 0 \Rightarrow x = 0 \text{ oder } y = 0$$

Zeigen Sie, dass Körper nullteilerfrei sein müssen, Ringe dagegen nicht.

**Aufgabe 2.20****Webcode  
2674**

Analysieren Sie die folgende Aussage:

$$\text{„Das Polynom } p \text{ ist irreduzibel} \Leftrightarrow p \text{ hat keine Nullstellen“}$$

Ist die Aussage immer richtig? Falls nicht, gilt sie für Polynome spezieller Grade?

**Aufgabe 2.21****Webcode  
2692**

In Abschnitt 2.4.1 haben wir erarbeitet, wie sich der größte gemeinsame Teiler zweier Polynome berechnen lässt. Eine Besonderheit des Algorithmus aus Abbildung 2.37 war es, die Polynome zu Beginn und nach jedem ausgeführten Rechenschritt zu normieren. In dieser Aufgabe werden Sie sehen, dass dies ausschließlich aus Komfortgründen geschah.

Berechnen Sie den größten gemeinsamen Teiler der Polynome

$$\begin{aligned} p(x) &:= x^6 + x^5 + \frac{1}{2}x^4 + \frac{1}{2}x^3 + 4x^2 + 4x, \\ q(x) &:= 4x^3 + 4x^2 + 2x + 2, \end{aligned}$$

indem Sie die unten stehenden Berechnungssequenzen ergänzen. Führen Sie die Berechnung zweimal aus, einmal mit normierten und einmal mit nicht normierten Polynomen.

a) Rechnung mit normierten Polynomen

$$\begin{aligned} &\text{ggT}\left( \boxed{x^6 + x^5 + \frac{1}{2}x^4 + \frac{1}{2}x^3 + 4x^2 + 4x}, \boxed{x^3 + x^2 + \frac{1}{2}x + \frac{1}{2}} \right) \\ &= \text{ggT}\left( \boxed{\phantom{x^6 + x^5 + \frac{1}{2}x^4 + \frac{1}{2}x^3 + 4x^2 + 4x}}, \boxed{\phantom{x^3 + x^2 + \frac{1}{2}x + \frac{1}{2}}} \right) \\ &= \text{ggT}\left( \boxed{\phantom{x^6 + x^5 + \frac{1}{2}x^4 + \frac{1}{2}x^3 + 4x^2 + 4x}}, \boxed{\phantom{x^3 + x^2 + \frac{1}{2}x + \frac{1}{2}}} \right) \\ &= \text{ggT}\left( \boxed{\phantom{x^6 + x^5 + \frac{1}{2}x^4 + \frac{1}{2}x^3 + 4x^2 + 4x}}, \boxed{\phantom{x^3 + x^2 + \frac{1}{2}x + \frac{1}{2}}} \right) \\ &= \boxed{\phantom{x^6 + x^5 + \frac{1}{2}x^4 + \frac{1}{2}x^3 + 4x^2 + 4x}} \end{aligned}$$

b) Rechnung mit nicht normierten Polynomen

$$\begin{aligned} &\text{ggT}\left( \boxed{x^6 + x^5 + \frac{1}{2}x^4 + \frac{1}{2}x^3 + 4x^2 + 4x}, \boxed{4x^3 + 4x^2 + 2x + 2} \right) \\ &= \text{ggT}\left( \boxed{\phantom{x^6 + x^5 + \frac{1}{2}x^4 + \frac{1}{2}x^3 + 4x^2 + 4x}}, \boxed{\phantom{4x^3 + 4x^2 + 2x + 2}} \right) \\ &= \text{ggT}\left( \boxed{\phantom{x^6 + x^5 + \frac{1}{2}x^4 + \frac{1}{2}x^3 + 4x^2 + 4x}}, \boxed{\phantom{4x^3 + 4x^2 + 2x + 2}} \right) \\ &= \text{ggT}\left( \boxed{\phantom{x^6 + x^5 + \frac{1}{2}x^4 + \frac{1}{2}x^3 + 4x^2 + 4x}}, \boxed{\phantom{4x^3 + 4x^2 + 2x + 2}} \right) \\ &= \boxed{\phantom{x^6 + x^5 + \frac{1}{2}x^4 + \frac{1}{2}x^3 + 4x^2 + 4x}} \end{aligned}$$

Alle Polynome in dieser Aufgabe stammen aus der Menge  $\mathbb{Z}_2[x]$ .

### Aufgabe 2.22



a) Vervollständigen Sie die leergelassenen Felder:

**Webcode  
2704**

$$\begin{array}{ll} x^2 \cdot x = \boxed{\phantom{x^2 \cdot x}} & x^2 \cdot (x+1) = \boxed{\phantom{x^2 \cdot (x+1)}} \\ (x^2 + 1) \cdot x = \boxed{\phantom{(x^2 + 1) \cdot x}} & (x^2 + 1) \cdot (x+1) = \boxed{\phantom{(x^2 + 1) \cdot (x+1)}} \\ (x^2 + x) \cdot x = \boxed{\phantom{(x^2 + x) \cdot x}} & (x^2 + x) \cdot (x+1) = \boxed{\phantom{(x^2 + x) \cdot (x+1)}} \\ (x^2 + x + 1) \cdot x = \boxed{\phantom{(x^2 + x + 1) \cdot x}} & (x^2 + x + 1) \cdot (x+1) = \boxed{\phantom{(x^2 + x + 1) \cdot (x+1)}} \end{array}$$

b) In  $\mathbb{Z}_2[x]$  gibt es 2 irreduzible Polynome vom Grad 3. Finden Sie heraus, welche dies sind.

$$p_1(x) =$$

$$p_2(x) =$$

c) Vervollständigen Sie die nachstehenden Multiplikationstabellen:

■ Multiplikation in  $\mathbb{Z}_2[x]_{p_1(x)}$

	0	1	$x$	$x+1$	$x^2$	$x^2+x$	$x^2+x+1$
0	0	1	$x$	$x+1$	$x^2$	$x^2+x$	$x^2+x+1$
1	1	0	$x+1$	$x^2+x$	$x^2+x+1$	$x$	$x^2$
$x$							
$x+1$							
$x^2$							
$x^2+1$							
$x^2+x$							
$x^2+x+1$							

■ Multiplikation in  $\mathbb{Z}_2[x]_{p_2(x)}$

	0	1	$x$	$x+1$	$x^2$	$x^2+x$	$x^2+x+1$
0	0	1	$x$	$x+1$	$x^2$	$x^2+x$	$x^2+x+1$
1	1	0	$x+1$	$x^2+x$	$x^2+x+1$	$x$	$x^2$
$x$							
$x+1$							
$x^2$							
$x^2+1$							
$x^2+x$							
$x^2+x+1$							

d) Zeigen Sie, dass die von  $p_1(x)$  und  $p_2(x)$  erzeugten Körper isomorph sind.

In dieser Aufgabe geht es um Polynome aus  $\mathbb{Z}_2[x]_{x^4+x^3+1}$ . Die Multiplikation in diesem Körper folgt dem üblichen Schema: Die zwei Operanden werden zunächst multipliziert und anschließend durch das irreduzible Polynom  $x^4 + x^3 + 1$  geteilt. Der Divisionsrest ist das gesuchte Ergebnis.

**Aufgabe 2.23****Webcode****2711**

Das Produkt von  $x^3 + 1$  und  $x^3 + x + 1$  lässt sich beispielsweise so ausrechnen:

- Multiplikation von  $x^3 + 1$  und  $x^3 + x + 1$

$$\begin{array}{r} 1001 \cdot 1011 \\ 1001 \\ 0000 \\ 1001 \\ \hline 1001 \end{array}$$

- Division durch  $x^4 + x^3 + 1$

$$\begin{array}{r} 1010011 : 11001 \\ 11001 \\ \hline 11011 \\ 11001 \\ \hline 0101 \end{array}$$

Als Ergebnis erhalten wir den Koeffizientenvektor **0101**, der ausgeschrieben das Polynom  $x^2 + 1$  ergibt.

- In Abschnitt 2.4.3 haben Sie gelernt, wie die Multiplikation ohne anschließende Polynomdivision durchgeführt werden kann. Multiplizieren Sie die Beispieldenomme auf diese Weise.
- Ferner wurde in Abschnitt 2.4.3 besprochen, wie sich die Multiplikation über Tabellenzugriffe beschleunigen lässt. Was wir hierfür benötigen, ist ein Generatorpolynom, das alle Elemente von  $\mathbb{Z}_2[x]_{x^4+x^3+1}$  erzeugt. Finden Sie ein solches Polynom.
- Füllen Sie für das gefundene Polynom die Logarithmen- und die Potenzentabelle aus:

- Logarithmentabelle

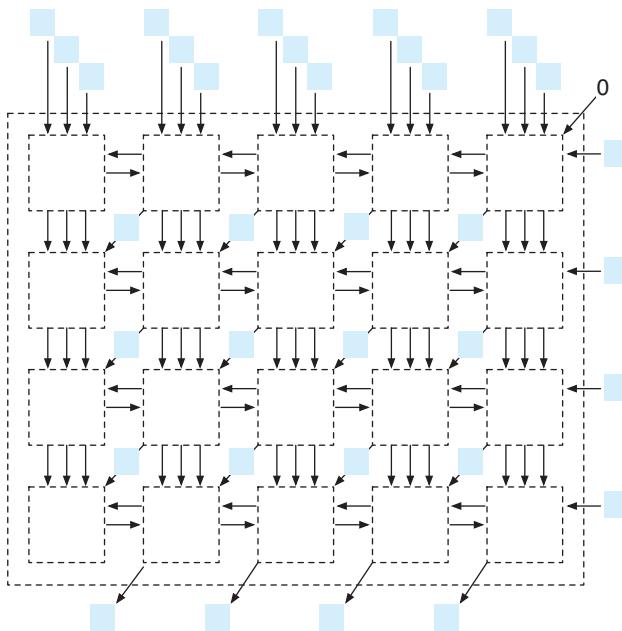
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

- Potenzentabelle

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Jede Hexadezimalziffer steht für die 4 Koeffizienten eines Polynoms aus  $\mathbb{Z}_2[x]_{x^4+x^3+1}$ .

- Berechnen Sie das Produkt der oben eingeführten Beispieldenomme mithilfe der erstellten Tabellen.
- Zeigen Sie, dass die Hardwareschaltung aus Abbildung 2.46 das gleiche Ergebnis liefert. Ergänzen Sie hierzu die schematische Zeichnung des Galois-körper-Multiplizierers:

**Aufgabe 2.24**

**Webcode  
2781**

In der Definition des Vektorraums haben wir gefordert, dass die 1 ein neutrales Element ist. Das bedeutet, dass die Multiplikation von 1 mit einem beliebigen Vektor  $x$  immer wieder den Vektor selbst ergibt:

$$1 \cdot x = x \text{ für alle } x \in V \quad (2.32)$$

Zeigen Sie, dass (2.32) nicht aus den anderen Vektorraumaxiomen gefolgert werden kann. Führen Sie den Beweis, indem Sie eine Struktur angeben, die (2.32) nicht erfüllt, aber allen anderen Eigenschaften eines Vektorraums genügt.

**Aufgabe 2.25**

**Webcode  
2811**

Es sei  $\mathbb{K}$  ein Körper und  $V$  die Menge aller Funktionen der Form

$$f : \mathbb{K} \rightarrow \mathbb{K}. \quad (2.33)$$

Die Addition zweier Funktionen und die Multiplikation einer Funktion mit einem Skalar seien folgendermaßen definiert:

$$(f+g)(x) := f(x) + g(x) \\ (c \cdot f)(x) := c \cdot f(x)$$

- a) Zeigen Sie, dass  $V$  mit den vereinbarten Verknüpfungen einen  $\mathbb{K}$ -Vektorraum bildet.

- b) Haben wir auch dann noch einen  $\mathbb{K}$ -Vektorraum vor uns, wenn wir eine beliebige nicht-leere Menge  $M$  wählen und in (2.33) die Funktionen  $f : \mathbb{K} \rightarrow \mathbb{K}$  durch Funktionen der folgenden Form ersetzen?

$$f : M \rightarrow \mathbb{K} \quad (2.34)$$

- c) Haben wir immer noch einen  $\mathbb{K}$ -Vektorraum vor uns, wenn wir in (2.34) den Körper  $\mathbb{K}$  durch einen  $\mathbb{K}$ -Vektorraum ersetzen? Die Funktionen, die auf diese Weise entstehen, bilden die Elemente aus der Menge  $M$  also nicht mehr auf Elemente eines Körpers, sondern auf Vektoren eines Vektorraums ab.

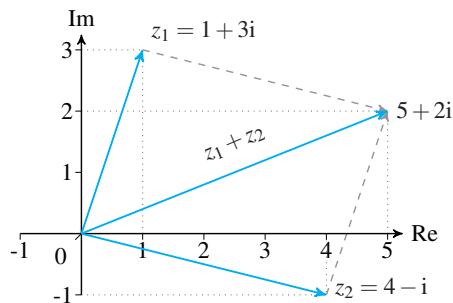
Auf der Menge  $\mathbb{C}$  der komplexen Zahlen sind die Addition und die Multiplikation folgendermaßen definiert:

$$(a+bi) + (c+di) := (a+c) + (b+d)i$$

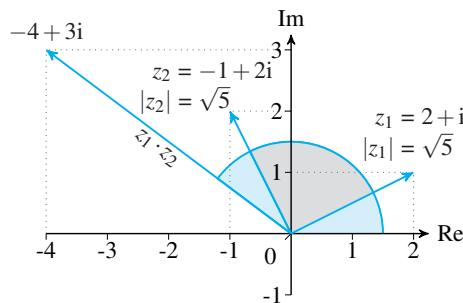
$$(a+bi) \cdot (c+di) := (ac-bd) + (ad+bc)i$$

Beide Verknüpfungen lassen sich geometrisch veranschaulichen, wenn wir uns komplexe Zahlen als Vektoren vorstellen. Die Summe zweier komplexer Zahlen können wir ausrechnen, indem wir die Vektoren der beiden Summanden addieren. Die Multiplikation lässt sich anschaulich deuten, wenn die komplexen Zahlen in *polaren Koordinaten*  $(r, \alpha)$  dargestellt werden. In dieser Darstellung ist  $r$  die Vektorlänge und  $\alpha$  der Winkel zur  $x$ -Achse. Das Produkt zweier komplexer Zahlen  $(r_1, \alpha_1)$  und  $(r_2, \alpha_2)$  ist dann der Vektor  $(r_1 \cdot r_2, \alpha_1 + \alpha_2)$ .

#### ■ Addition in $\mathbb{C}$



#### ■ Multiplikation in $\mathbb{C}$



Die Multiplikation einer reellen und einer komplexen Zahl ist ein Spezialfall der Multiplikation zweier komplexer Zahlen. Eine reelle Zahl ist eine komplexe Zahl mit Imaginäranteil 0.

- a) Zeigen Sie, dass die Menge der komplexen Zahlen einen  $\mathbb{R}$ -Vektorraum bildet.  
 b) Ist der Vektorraum  $\mathbb{C}$  isomorph zu  $\mathbb{R}^2$ ?

### Aufgabe 2.26

Webcode  
2871

**Aufgabe 2.27****Webcode  
2887**

Mit  $\mathbb{R}[x]_{\leq n}$  bezeichnen wir die Menge aller Polynome aus  $\mathbb{R}[x]$  mit einem Grad  $\leq n$ . Offenbar ist  $\mathbb{R}[x]_{\leq n}$  für jede natürliche Zahl  $n$  ein Untervektorraum von  $\mathbb{R}[x]$ .

- Welche Dimension besitzt der Untervektorraum  $\mathbb{R}[x]_{\leq n}$ ?
- Welchen bekannten Vektorraum erhalten wir für den Fall  $n = 0$ ?
- Welche der nachstehenden Mengen bilden eine Basis von  $\mathbb{R}[x]_{\leq 2}$ ?

- |  |  |
|--|--|
| <input type="checkbox"/> $B_1 = \{x + 1, x^2 + x\}$          | <input type="checkbox"/> $B_4 = \{x^2 + 1, x, x^2 + x + 1\}$   |
| <input type="checkbox"/> $B_2 = \{1, x, x^2\}$               | <input type="checkbox"/> $B_5 = \{x^2 + 1, x, x^2 + 2x + 1\}$  |
| <input type="checkbox"/> $B_3 = \{x + 1, x^2 + 1, x^2 + x\}$ | <input type="checkbox"/> $B_6 = \{x^2 + 1, x, 2x^2 + 2x + 2\}$ |

**Aufgabe 2.28****Webcode  
2910**

Mit  $V$  sei ein beliebiger  $\mathbb{K}$ -Vektorraum gegeben. In Abschnitt 2.5.2 haben wir die Verknüpfung  $, \cdot '$  :  $V \times V \rightarrow \mathbb{K}$  mit

$$\mathbf{x} \cdot \mathbf{y} := \sum_{i=0}^{n-1} x_i \cdot y_i \quad (2.35)$$

als *inneres Produkt* bezeichnet. In manchen Büchern werden an diesen Begriff strengere Forderungen geknüpft, die (2.35) nicht in jedem Fall erfüllt.

Nach dieser strengerer Definition ist eine Abbildung  $, \cdot '$  :  $V \times V \rightarrow \mathbb{K}$  genau dann ein inneres Produkt, wenn die folgenden Eigenschaften gelten:

- $, \cdot '$  ist eine Bilinearform
  - $(\mathbf{x} + \mathbf{y}) \cdot \mathbf{z} = (\mathbf{x} \cdot \mathbf{z}) + (\mathbf{y} \cdot \mathbf{z})$  für alle  $\mathbf{x}, \mathbf{y}, \mathbf{z} \in V$
  - $\mathbf{z} \cdot (\mathbf{x} + \mathbf{y}) = (\mathbf{z} \cdot \mathbf{x}) + (\mathbf{z} \cdot \mathbf{y})$  für alle  $\mathbf{x}, \mathbf{y}, \mathbf{z} \in V$
  - $\lambda \cdot (\mathbf{x} \cdot \mathbf{y}) = ((\lambda \mathbf{x}) \cdot \mathbf{y}) = (\mathbf{x} \cdot (\lambda \mathbf{y}))$  für alle  $\mathbf{x}, \mathbf{y} \in V, \lambda \in \mathbb{K}$
- $, \cdot '$  ist symmetrisch
  - $\mathbf{x} \cdot \mathbf{y} = \mathbf{y} \cdot \mathbf{x}$  für alle  $\mathbf{x}, \mathbf{y} \in V$
- $, \cdot '$  ist positiv definit
  - $\mathbf{0} \cdot \mathbf{0} = 0$  und  $\mathbf{x} \cdot \mathbf{x} > 0$  für alle  $\mathbf{x} \in V$

Ein inneres Produkt wird auch als *Skalarprodukt* bezeichnet.

- Zeigen Sie, dass (2.35) nach dieser Definition tatsächlich ein inneres Produkt von  $\mathbb{R}^n$  ist.

- b) Zeigen Sie, dass (2.35) in manchen Körpern nur noch eine symmetrische Bilinearform, aber kein inneres Produkt mehr ist.

Die beiden nachstehenden Generatormatrizen erzeugen zwei Untervektorräume von  $\mathbb{Z}_2^3$ :

$$G_1 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \quad G_2 = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$$

**Aufgabe 2.29**



**Webcode**  
**2916**

- a) Welche Dimension haben die erzeugten Vektorräume?  
b) Bestimmen Sie für beide den jeweils orthogonalen Vektorraum.

$U$  sei der Untervektorraum des  $\mathbb{Z}_2^7$ , der durch die nachstehende Basis gegeben ist:

$$\left\{ \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \right\}$$

**Aufgabe 2.30**



**Webcode**  
**2949**

- a) Beschreiben Sie  $U$  mithilfe einer Generatormatrix.  
b) Konstruieren Sie eine Basis für den Orthogonalraum  $U^\perp$ . Welche Dimension hat  $U^\perp$ ?  
c) Bringen Sie die erzeugte Matrix in die systematische Form.

Mit  $\mathbb{Z}_2^\infty$  bezeichnen wir die Menge aller Folgen aus Elementen der Menge  $\mathbb{Z}_2$ :

$$\mathbb{Z}_2^\infty := \{(x_0, x_1, x_2, x_3, \dots) \mid x_i \in \{0, 1\}\}$$

**Aufgabe 2.31**



**Webcode**  
**2979**

Die Elemente von  $\mathbb{Z}_2^\infty$  können wir uns als unendliche lange Bitvektoren vorstellen.

- a) Zeigen Sie, dass  $\mathbb{Z}_2^\infty$  zu einem Vektorraum über  $\mathbb{Z}_2$  wird, wenn die Addition zweier Folgen und die Multiplikation einer Folge mit einem Skalar komponentenweise erklärt wird.  
b) Ist die Menge  $\{(1, 0, 0, 0, \dots), (0, 1, 0, 0, \dots), (0, 0, 1, 0, \dots), \dots\}$  eine Basis von  $\mathbb{Z}_2^\infty$ ?

**Aufgabe 2.32**

Eine unendliche Menge  $M$  heißt



**Webcode  
2984**

- *abzählbar*, wenn eine bijektive Abbildung zwischen  $M$  und  $\mathbb{N}$  existiert,
- *überabzählbar*, wenn eine solche Abbildung nicht existiert.

Im Jahr 1874 publizierte der deutsche Mathematiker Georg Cantor eine Arbeit mit dem Titel „Über eine Eigenschaft des Inbegriffs aller reellen algebraischen Zahlen“ [15].

Der Inhalt dieser Arbeit war gewaltig. Cantor bewies dort zum ersten Mal die Überabzählbarkeit der reellen Zahlen (des sogenannten *Kontinuums*). Ab jetzt war klar, dass die Anzahl der reellen Zahlen jene der natürlichen Zahlen so sehr übersteigt, dass es unmöglich ist, eine Eins-zu-eins-Zuordnung zwischen den Elementen beider Mengen herzustellen.

Für die Mathematik war dieses Ergebnis von unschätzbarem Wert, denn Cantor hatte gezeigt, dass die Menge der natürlichen Zahlen und die Menge der reellen Zahlen stellvertretend für verschiedene Unendlichkeiten stehen. Mit seiner Methodik war es fortan möglich, den Begriff der Unendlichkeit systematisch zu untersuchen.



Georg Cantor (1845 – 1918)

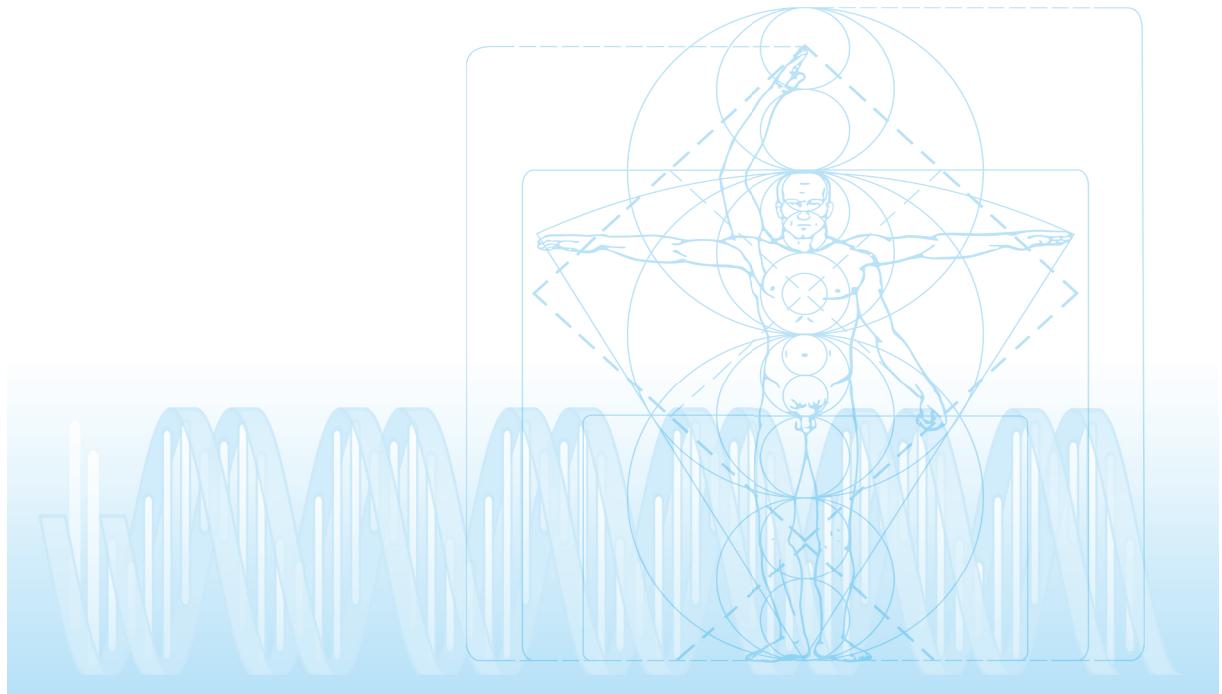
Zeigen oder widerlegen Sie die folgenden Behauptungen:

- a) Die Menge, die alle endlichen Folgen von natürlichen Zahlen umfasst, ist abzählbar.
- b) Das Intervall  $[0; 1]$  enthält überabzählbar viele Elemente.
- c) Jede Basis von  $\mathbb{Z}_2^\infty$  enthält überabzählbar viele Elemente.

# 3 Codierungen, Codes und Information

In diesem Kapitel werden Sie ...

- den Unterschied zwischen Codierungen und Codes verstehen,
- längenvariable Codes und Blockcodes unterscheiden,
- mehrere Beispiele aus der Praxis kennenlernen,
- den Begriff des linearen Codes verinnerlichen,
- dem Shannon'schen Übertragungsmodell wiederbegegnen,
- die Kapazität des Morse-Kanals formal berechnen,
- die verschiedenen Facetten des Informationsbegriffs analysieren.

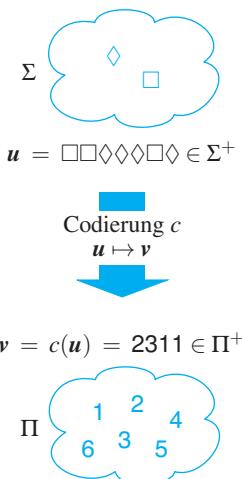


## 3.1 Motivation

In diesem Kapitel führen wir elementare Begriffe aus dem Bereich der Informations- und Codierungstheorie ein, auf die wir in den späteren Kapiteln immer wieder zurückgreifen werden. Zunächst klären wir in Abschnitt 3.2, was unter einer *Codierung* und einem *Code* genau zu verstehen ist. Danach verschaffen wir uns einen Überblick über die wichtigsten Codeklassen und füllen die eingeführten Begriffe in den Abschnitten 3.3 und 3.4 mit Leben. In Abschnitt 3.5 kommen wir auf das Shannon'sche Übertragungsmodell zurück und definieren wichtige Termini wie die *Übertragungsrate* und die *Kanalkapazität*. Den Schlusspunkt setzt Abschnitt 3.6 mit einer allgemeinen Diskussion über den Shannon'schen Informationsbegriff.

## 3.2 Definition und Eigenschaften

*Das Quellenalphabet enthält die Symbole, aus denen die Nachrichten aufgebaut sind.*



*Das Codealphabet enthält alle Symbole, aus denen die codierten Nachrichten bestehen.*

**Abb. 3.1:** Eine Codierung  $c$  ist eine Abbildung, die jede Nachricht  $u \in \Sigma^+$  auf ein Codewort  $v \in \Pi^+$  abbildet.

Wir beginnen mit der vielleicht wichtigsten Definition dieses Buchs:



### Definition 3.1 (Codierung)

$\Sigma$  und  $\Pi$  seien zwei endliche Mengen von Grundsymbolen. Unter einer *Codierung* verstehen wir eine injektive Abbildung der Form

$$c : \Sigma^+ \rightarrow \Pi^+$$

$\Sigma$  ist das *Quellenalphabet* und  $\Pi$  das *Codealphabet* von  $c$ .

Die Mengen  $\Sigma^+$  und  $\Pi^+$  enthalten alle endlichen Sequenzen, die mit den Symbolen aus  $\Sigma$  bzw.  $\Pi$  aufgebaut werden können und mindestens ein Zeichen lang sind (Abbildung 3.1):

$$\begin{aligned} \Sigma^+ &:= \{\sigma_1\sigma_2\dots\sigma_n \mid n \geq 1, \sigma_i \in \Sigma\} \\ \Pi^+ &:= \{\tau_1\tau_2\dots\tau_n \mid n \geq 1, \tau_i \in \Pi\} \end{aligned}$$

In der digitalen Datenübertragung spielt das Codealphabet  $\Pi = \{0, 1\}$  eine hervorgehobene Rolle. Wir sprechen in diesem Fall von *binären Codierungen*, da sie endliche Sequenzen von Nullen und Einsen produzieren. Fast alle Codierungen, die wir in diesem Buch betrachten, sind binär oder zu einer binären Codierung äquivalent.

Typische Quellenalphabete sind die Mengen  $\{0, 1\}$  und  $\{0, \dots, 255\}$ . Das erste wird von Codierungen verwendet, die endliche Binärsequenzen als Eingabe entgegennehmen, und das zweite von Codierungen, die einen digitalen Datenstrom Byte für Byte übersetzen.

In unserer Definition haben wir explizit gefordert, dass  $c$  eine injektive Abbildung ist, und somit die folgende Bedingung erfüllt:

$$c(\mathbf{u}_1) = c(\mathbf{u}_2) \Rightarrow \mathbf{u}_1 = \mathbf{u}_2$$

Die Injektivität sorgt dafür, dass die Originalnachricht durch die codierte Sequenz eindeutig bestimmt ist, und sich daher eins zu eins rekonstruieren lässt. Folgerichtig haben wir es in diesem Buch ausschließlich mit *verlustfreien* Codierungen zu tun.

Nicht alle Codierungen, die in der Praxis verwendet werden, sind verlustfrei. Beispiele sind die gebräuchlichen Kompressionsformate aus dem Audio- und Videobereich, die auf die Speicherung bestimmter Informationen bewusst verzichten, und genau hierdurch sehr hohe Kompressionsraten erzielen. Solche *verlustbehafteten* Codierungen sind ein eigenes Forschungsfeld und werden hier nicht behandelt.

## Lauflängencodierung

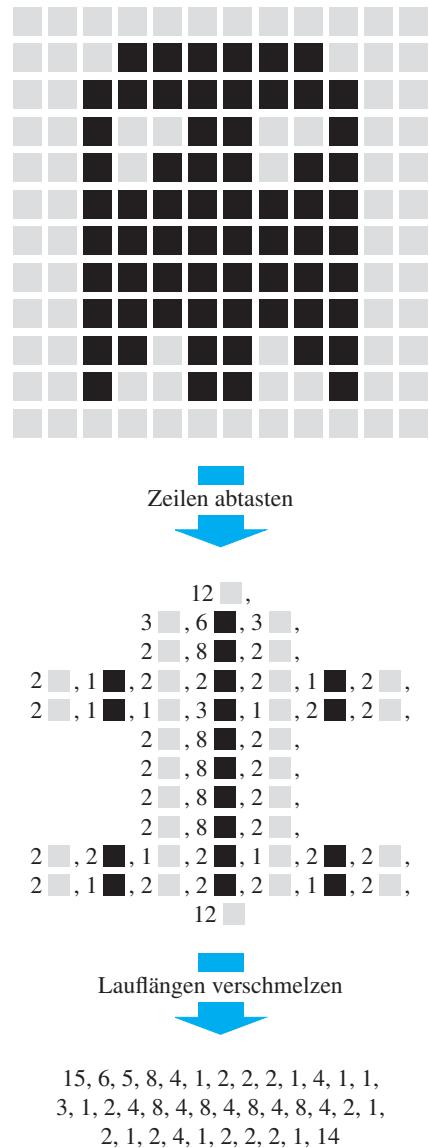
Ein Beispiel, das die wesentlichen Merkmale des abstrakten Codierungsbegriffs deutlich zum Vorschein bringt, ist die *Lauflängencodierung* (*Run-Length Encoding*, kurz RLE). Bei dieser Art der Codierung wird die Idee verfolgt, mehrfache Wiederholungen des gleichen Quellsymbols numerisch zu erfassen und durch einen Zähler auszudrücken.

In dem folgenden Beispiel nehmen wir an, das Quellenalphabet  $\Sigma$  sei die Menge  $\{A, \dots, Z\}$  und das zu codierende Wort sei KAFFEE. Eine Lauflängencodierung ist dann auf zwei prinzipiell unterschiedliche Weisen möglich:

- Codierung ohne Zählerzeichen

KAFFEE  $\mapsto$  1K1A2F2E

Bei dieser Art der Lauflängencodierung wird jedem Zeichen ein Zähler vorangestellt, der die Anzahl der Wiederholungen spezifiziert. Dieses Vorgehen erleichtert vor allem die Decodierung, da die Position eines Zeichens darüber Auskunft gibt, ob es sich um ein Zeichen des Quellenalphabets oder eine numerische Lauflänge handelt. Auf der negativen Seite führt diese Form der Codierung zu einer



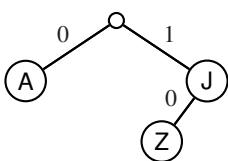
**Abb. 3.2:** Lauflängencodierung am Beispiel einer binären Nachricht

■ Codierung  $c_1$

Tabellarische Darstellung:

$\sigma$	$c_1(\sigma)$
A	0
J	1
Z	10

Baumdarstellung:



■ Codierung  $c_2$

Tabellarische Darstellung:

$\sigma$	$c_2(\sigma)$
A	0
J	10
Z	11

Baumdarstellung:

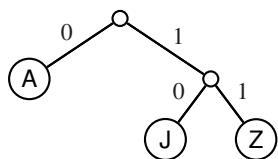


Abb. 3.3: Zwei Darstellungsmöglichkeiten für zeichenweise arbeitende Codierungen

erheblichen Verlängerung der Originalnachricht, wenn sich dort nur wenige Symbole wiederholen.

■ Codierung mit Zählerzeichen

KAFFEE  $\mapsto$  KA#2F#2E

In diesem Fall wird jedem Zähler ein spezielles Markierungssymbol, z. B. #, vorangestellt, um eine Verwechslung mit den Symbolen des Quellenalphabets zu vermeiden. Dieses Vorgehen ist der erstgenannten Variante überlegen, falls sich nur wenige Symbole in Form von Ketten wiederholen; es kann in der Praxis aber nur dann verwendet werden, wenn ein freies Symbol zur Verfügung steht, das die Rolle des Markierungszeichens übernehmen kann.

Besonders effizient ist die Lauflängencodierung dann, wenn ein binäres Quellenalphabet verwendet wird, wie beispielsweise bei der Codierung von Schwarz-Weiß-Bildern (Abbildung 3.2). Da sich die beiden Quellsymbole in einer Nachricht dann stets abwechseln, brauchen wir lediglich die Lauflängen abzuspeichern. Diese reichen aus, um die Originalnachricht verlustfrei zu rekonstruieren.

## Zeichenweise Codierung

Viele Codierungen sind so beschaffen, dass jedem Element des Quellenalphabets eine bestimmte Sequenz von Symbolen des Codealphabets zugeordnet wird und die Codierung einer Nachricht zeichenweise erfolgt. Es gilt dann

$$c(\mathbf{u}) = c(u_0 u_1 \dots u_{n-1}) = c(u_0) c(u_1) \dots c(u_{n-1}) \quad (3.1)$$

Wir haben in den allermeisten Fällen solche Codierungen im Sinn, wenn wir die nachstehenden Begriffe verwenden:



### Definition 3.2 (Code, Codewort)

Sei  $c$  eine Codierung. Der von  $c$  erzeugte *Code* ist die Menge

$$C := c(\Sigma) := \{c(\sigma) \mid \sigma \in \Sigma\}$$

Die Zeichenkette  $c(\sigma)$  ist das *Codewort* des Zeichens  $\sigma$ .

Für Codierungen, die zeichenweise arbeiten, sind zwei verschiedene Darstellungen gebräuchlich (Abbildung 3.3):

### ■ Tabellarische Darstellung

Die Symbole des Quellenalphabets werden zusammen mit ihren Codewörtern tabellarisch aufgelistet. Diese Art der Darstellung ist für uns nicht neu; wir haben sie bereits ausgiebig in Kapitel 1 eingesetzt.

### ■ Baumdarstellung

Die Codierung wird durch einen Baum dargestellt, in dem jeder Pfad von der Wurzel bis zu einem Knoten ein mögliches Codewort beschreibt. Ist das Codealphabet  $\Pi$ , wie in den betrachteten Beispielen, die Menge  $\{0, 1\}$ , so entsteht ein Binärbaum. In diesem Fall verwenden wir die Konvention, dass eine Kante nach links dem Symbol 0 und eine Kante nach rechts dem Symbol 1 entspricht.

Codes, die von zeichenweise arbeitenden Codierungen erzeugt werden, lassen sich in *längenvariable Codes* und *Blockcodes* unterscheiden. Beide Codeklassen wollen wir nun genauer ansehen.

### ■ Codierung $c_1$

A $\mapsto$ 0	J $\mapsto$ 1	Z $\mapsto$ 10
---------------	---------------	----------------

JAZZ  $\mapsto$  101010

### ■ Codierung $c_2$

A $\mapsto$ 0	J $\mapsto$ 10	Z $\mapsto$ 11
---------------	----------------	----------------

JAZZ  $\mapsto$  1001111

**Abb. 3.4:** Codierung der Nachricht JAZZ mit den beiden längenvariablen Codes aus Abbildung 3.3

## 3.3 Längenvariable Codes

Ein Code oder eine Codierung fällt in diese Klasse, wenn mindestens zwei Codewörter unterschiedlich lang sind. Längenvariable Codes spielen vor allem in der Datenkompression eine Rolle. Dort wird die Idee verfolgt, häufig vorkommende Symbole mit kürzeren und selten vorkommende Sequenzen mit längeren Sequenzen zu belegen.

Die in Abbildung 3.3 gezeigten Codierungen  $c_1$  und  $c_2$  sind beide längenvariabel. Benutzen wir sie, um beispielsweise die Nachricht JAZZ zu codieren, so erhalten wir das in Abbildung 3.4 dargestellte Ergebnis.

### 3.3.1 Präfixfreie Codes

Auch wenn unsere Beispielcodierungen  $c_1$  und  $c_2$  auf den ersten Blick sehr ähnlich wirken, sind sie in einem wichtigen Punkt anders. Der Unterschied tritt zu Tage, sobald wir versuchen, eine Symbolsequenz zu decodieren. Mit dem zweiten Code ist dies immer in einer eindeutigen Weise möglich, mit dem ersten Code dagegen nicht. Abbildung 3.5 zeigt, dass mehrere Nachrichten auf die gleiche Sequenz abgebildet werden.

Ein genauer Blick auf die Codetabelle macht klar, wie der zweite Code dieses Problem vermeidet. Dort sind die Codewörter so gewählt, das

### ■ Codierung $c_1$

A $\mapsto$ 0	J $\mapsto$ 1	Z $\mapsto$ 10
---------------	---------------	----------------

ZZZ  $\mapsto$  101010

JAZZ  $\mapsto$  101010

ZJAZ  $\mapsto$  101010

ZZJA  $\mapsto$  101010

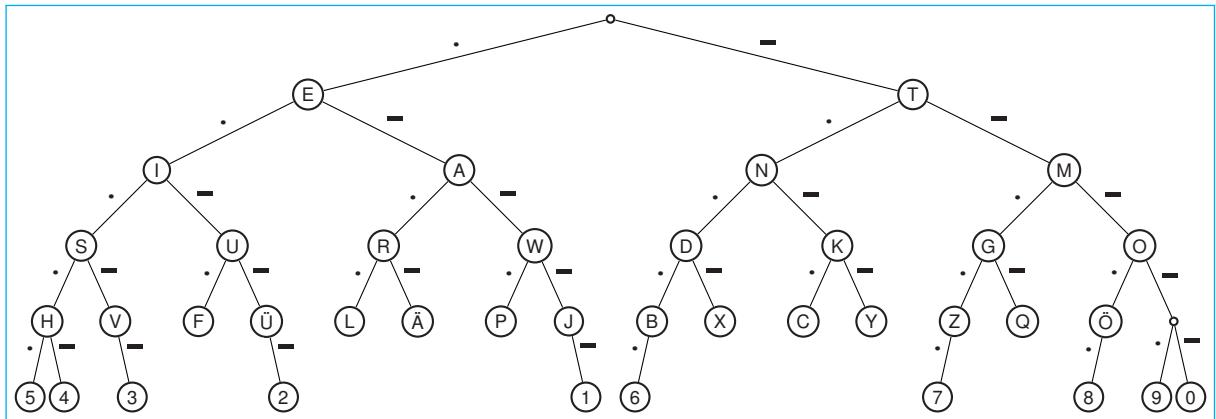
JAJAZ  $\mapsto$  101010

JAZJA  $\mapsto$  101010

ZJAJA  $\mapsto$  101010

JAJAJA  $\mapsto$  101010

**Abb. 3.5:** Wird auf die Übertragung von Trennzeichen verzichtet, entstehen Mehrdeutigkeiten. Die Originalnachricht lässt sich nicht mehr in jedem Fall eindeutig rekonstruieren.



**Abb. 3.6:** Baumdarstellung des Morse-Codes (ohne Sonderzeichen). Der Code ist nicht präfixfrei, da neben den Blättern auch innere Knoten mit Quellsymbolen markiert sind.

keines das Anfangsstück eines anderen ist. Wir sagen in diesem Fall, der Code sei *präfixfrei* oder, was das Gleiche ist: er erfülle die *Fano-Bedingung*. Diese Codes sind so wichtig, dass wir ihnen eine eigene Definition spendieren:



### Definition 3.3 (Präfixfreier Code, Fano-Bedingung)

Ist kein Codewort der Anfang eines anderen Codeworts, so

- ist der Code *präfixfrei* und
- erfüllt die *Fano-Bedingung*.

Ob ein Code präfixfrei ist oder nicht, lässt sich sofort an dessen Baumdarstellung erkennen: Er ist es genau dann, wenn kein innerer Knoten mit einem Symbol des Quellenalphabets markiert ist; dann und nur dann ist kein Codewort der Anfang eines anderen. Mit diesem Wissen reicht ein flüchtiger Blick auf Abbildung 3.6 aus, um den Morse-Code als nicht präfixfrei zu identifizieren.

In der Morse-Telegrafie wird die fehlende Präfixfreiheit kompensiert, indem nach jedem Buchstaben und jedem Wort eine Pause eingefügt wird. Alfred Vail und Samuel Morse taten dies bewusst, da selbst ein geschulter Telegrafist erhebliche Schwierigkeiten hätte, Buchstaben und Wörter ohne Pausen voneinander zu trennen. Dies gilt für alle Kommunikationsszenarien, in denen der Mensch die Rolle des Empfängers

übernimmt. Anders ist die Situation, wenn die Decodierung automatisch erfolgt. Hier ist die Präfixfreiheit von Vorteil, und so verwundert es nicht, dass in diesem Bereich fast ausschließlich Codes verwendet werden, die keine Pausenzeichen zur Trennung der Wörter benötigen.

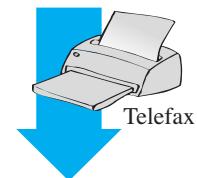
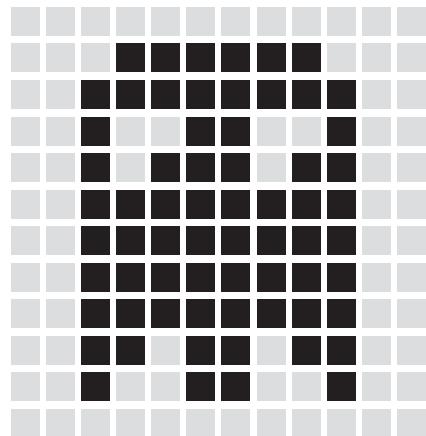
### Beispiel: Faxcodierung

In der Realität werden Sie häufig auf Encoder treffen, die eine Nachricht stufenweise übersetzen und in jeder Stufe ein anderes Codierungsverfahren verwenden. Viele praxisrelevante Codierungsalgorithmen sind daher Mischformen der Algorithmen, die wir in diesem Buch voneinander getrennt besprechen. Ein lehrreiches Beispiel in dieser Hinsicht ist die Codierung, die bei der Übertragung von Telefaxnachrichten zum Einsatz kommt. Eine eingelegte Textvorlage wird in zwei Schritten bearbeitet:

- Im ersten Schritt wird die Vorlage Zeile für Zeile von einer Fotodiode abgetastet und für jede gelesene Pixelkette die Länge berechnet. Die Telefaxcodierung beruht daher im Kern auf einer Lauflängencodierung, wie wir sie in Abbildung 3.2 kennengelernt haben.
- Um die Lauflängen mit möglichst wenigen Bits zu übertragen, werden sie durch spezielle Bitmuster repräsentiert, die in Tabelle 3.1 zusammengefasst sind. Wie bei allen längenvariablen Codierungen wird auch hier das Ziel verfolgt, häufiger vorkommende Symbole (hier sind die Symbole Lauflängen) mit kurzen Bitsequenzen und seltener vorkommende Symbole mit langen Bitsequenzen zu codieren. Da weiße und schwarze Pixelketten unterschiedlich häufig auftreten, benutzen Faxgeräte für jede Farbe eine separate Tabelle.

Mithilfe von Tabelle 3.1 können wir bestimmen, mit welcher Bitsequenz ein Faxgerät die Grafik aus Abbildung 3.2 übertragen würde. Schlagen wir abwechselnd in den Spalten für weiße und schwarze Pixel nach, so erhalten wir die Bitsequenz aus Abbildung 3.7. Die Codewörter, die aus der Tabelle für schwarze Lauflängen stammen, sind in einer anderen Farbe abgedruckt. Die Hervorhebung ist lediglich der Übersichtlichkeit geschuldet. Da die weißen und die schwarzen Lauflängen jeweils präfixfrei codiert sind, lassen sich die Codewörter immer eindeutig aus dem Bitstrom extrahieren.

Ist Ihnen aufgefallen, dass in Tabelle 3.1 nicht alle Lauflängen ein Codewort besitzen? Dies ist eine Besonderheit der Faxcodierung, die in der Praxis aber keinerlei Probleme bereitet. Um z. B. eine Kette von



```
110101001011000001011011010  
01111011101010110100001111  
00001111101100010110110001  
011011000101101100010110111  
100011111000111111011011010011  
1110111010110100
```

Abb. 3.7: Faxcodierung

<i>n</i>	Weiß	Schwarz	<i>n</i>	Weiß	Schwarz	<i>n</i>	Weiß	Schwarz
0	00110101	0000110111	32	00011011	000001101010	64	11011	0000001111
1	000111	010	33	00010010	000001101011	128	10010	000011001000
2	0111	11	34	00010011	000011010010	192	010111	000011001001
3	1000	10	35	00010100	000011010011	256	0110111	000001011011
4	1011	011	36	00010101	000011010100	320	00110110	000000110011
5	1100	0011	37	00010110	000011010101	384	00110111	000000110100
6	1110	0010	38	00010111	000011010110	448	01100100	000000110101
7	1111	00011	39	00101000	000011010111	512	01100101	0000001101100
8	10011	000101	40	00101001	000001101100	576	01101000	0000001101101
9	10100	000100	41	00101010	000001101101	640	01100111	0000001001010
10	00111	0000100	42	00101011	000011011010	704	011001100	0000001001011
11	01000	0000101	43	00101100	000011011011	768	011001101	0000001001100
12	001000	0000111	44	00101101	000001010100	832	011010010	0000001001101
13	000011	00000100	45	000000100	000001010101	896	011010011	0000001110010
14	110100	00000111	46	000000101	000001010110	960	011010100	0000001110011
15	110101	000011000	47	00001010	000001010111	1024	011010101	0000001110100
16	101010	0000010111	48	00001011	000001100100	1088	011010110	0000001110101
17	101011	0000011000	49	01010010	000001100101	1152	011010111	0000001110110
18	0100111	0000001000	50	01010011	000001010010	1216	011011000	0000001110111
19	0001100	00001100111	51	01010100	000001010011	1280	011011001	0000001010010
20	0001000	00001101000	52	01010101	000000100100	1344	011011010	0000001010011
21	0010111	00001101100	53	00100100	000000110111	1408	011011011	0000001010100
22	0000011	00000110111	54	00100101	000000111000	1472	010011000	0000001010101
23	0000100	00000101000	55	01011000	000000100111	1536	010011001	0000001011010
24	0101000	00000010111	56	01011001	000000101000	1600	010011010	0000001011011
25	0101011	00000011000	57	01011010	000001011000	1664	011000	0000001100100
26	0010011	000011001010	58	01011011	000001011001	1728	010011011	0000001100101
27	0100100	000011001011	59	01001010	000000101011	1792	00000001000	00000001000
28	0011000	000011001100	60	01001011	000000101100	1856	00000001100	00000001100
29	00000010	000011001101	61	00110010	000001011010	1920	00000001101	00000001101
30	00000011	000001101000	62	00110011	000001100110	1984	000000010010	000000010010
31	00011010	000001101001	63	00110100	000001100111	EOL	0000000000001	0000000000001

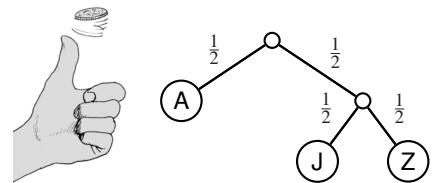
Tab. 3.1: Lauflängencodierung am Beispiel des Faxprotokolls

68 weißen Pixeln zu übertragen, wird diese ganz einfach in eine Kette zerlegt, die aus 64 weißen, 0 schwarzen, und 4 weißen Pixeln besteht:



Ein ähnlicher Trick wird angewendet, wenn das erste Pixel schwarz ist. Um die Konvention einzuhalten, dass die Übertragung mit weißen Pixelketten beginnt, wird zu Beginn das Bitmuster für die Lauflänge 0 aus

der Tabelle für weiße Pixel übertragen.



**Abb. 3.8:** Der Codebaum eines präfixfreien Codes lässt sich als die Beschreibung eines mehrstufigen Wahrscheinlichkeitsexperiments interpretieren. Die Kanten sind in diesem Fall mit Eintrittswahrscheinlichkeiten markiert.

### 3.3.2 Kraft'sche Ungleichung

Weiter oben haben wir aufgedeckt, dass sich die die Eigenschaft der Präfixfreiheit an der Baumdarstellung eines Codes sofort erkennen lässt; sie ist genau dann vorhanden, wenn kein innerer Knoten mit einem Symbol des Quellenalphabets markiert ist. Dies führt uns auf direktem Weg zu einer ganz anderen Eigenschaft, die alle präfixfreien Codes erfüllen müssen. Wir kommen ihr auf die Schliche, wenn wir die Codebäume als die Visualisierung eines mehrstufigen Zufallsexperiments deuten (Abbildung 3.8).

Im Falle eines Binärbaums verläuft dieses Experiment so: Wir starten an der Wurzel und werfen eine Münze. Anschließend wählen wir einen der beiden Kindknoten als Nachfolger aus, je nachdem, ob die Münze Kopf oder Zahl zeigt. Den Münzwurf wiederholen wir so oft, bis ein Blatt erreicht wird. Das dort annotierte Symbol ist dann das Ergebnis des Zufallsexperiments.

An der Baumdarstellung in Abbildung 3.8 können wir für die Beispielcodierung  $c_2$  ablesen, mit welchen Wahrscheinlichkeiten das Experiment die verschiedenen Quellsymbole hervorbringt. Es ist

$$p(A) = \frac{1}{2}, p(J) = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}, p(Z) = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$$

Bezeichnen wir die Länge der Codewörter  $c(A)$ ,  $c(J)$  und  $c(Z)$  mit  $l_1$ ,  $l_2$  und  $l_3$ , so können wir diese Beziehungen auch so ausdrücken:

$$p(A) = \frac{1}{2^{l_1}}, p(J) = \frac{1}{2^{l_2}}, p(Z) = \frac{1}{2^{l_3}}$$

Die Wahrscheinlichkeit, durch das kontinuierliche Werfen der Münze eines der Symbole A, J oder Z zu erhalten, ist 1, sodass wir die folgende Gleichung aufstellen können:

$$\sum_{i=1}^3 \frac{1}{2^{l_i}} = 1 \quad (3.2)$$

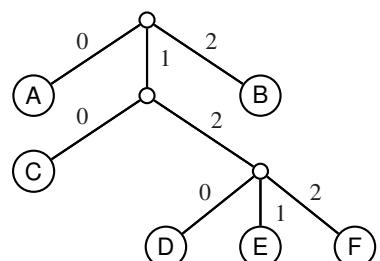
Dieses Ergebnis lässt sich problemlos verallgemeinern, wenn wir die beiden in Abbildung 3.9 angedeuteten Besonderheiten beachten:

#### Codierung $c_3$

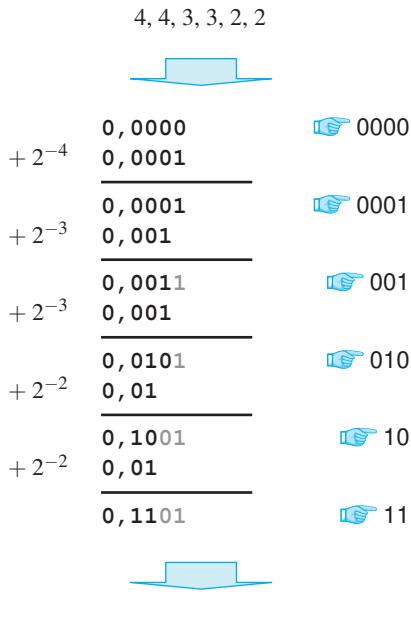
Tabellarische Darstellung:

$\sigma$	$c_3(\sigma)$
A	0
B	2
C	10
D	120
E	121
F	122

Baumdarstellung:



**Abb. 3.9:** Codebaum eines präfixfreien Codes mit einem dreielementigen Codealphabet



$$C = \{0000, 0001, 001, 010, 10, 11\}$$

**Abb. 3.10:** Konstruktion eines präfixfreien Codes, dessen Codewörter die vorgegebenen Längen aufweisen.

■ Im Allgemeinen ist der Baum eines präfixfreien Codes kein Binärbaum, sondern ein Baum, dessen innere Knoten bis zu  $|\Pi|$  Nachfolger haben. Das bedeutet, dass jedes Codewort mit dem Gewicht  $\frac{1}{|\Pi|^{l_i}}$  in die Summe (3.2) eingeht.

■ Der Codebaum eines präfixfreien Codes muss nicht saturiert sein; es kann also innere Knoten mit weniger als  $|\Pi|$  Kindern geben. In diesem Fall führt nicht jeder Ausgang unseres Zufallsexperiments zu einem Symbol des Quellenalphabets, sodass wir das Gleichheitszeichen in (3.2) durch  $\leq$  ersetzen müssen.

Insgesamt folgt aus unserer Überlegung, dass ausnahmslos jeder präfixfreie Code die folgende Bedingung erfüllt:

$$\sum_i \frac{1}{|\Pi|^{l_i}} \leq 1 \quad (3.3)$$

Was wir hier vor uns haben, ist die *Kraft'sche Ungleichung*, mit der Leon Gordon Kraft im Jahr 1949 eine prinzipielle Limitierung präfixfreier Codes zum Ausdruck brachte [52]:

### Satz 3.1 (Kraft'sche Ungleichung, Leon Kraft, 1949)

Für jede Folge  $l_1, \dots, l_n$  von natürlichen Zahlen gilt:

$$\text{Es existiert ein präfixfreier Code mit dem Codealphabet } \Pi \text{ und den Codewortlängen } l_1, \dots, l_n \Leftrightarrow \sum_i \frac{1}{|\Pi|^{l_i}} \leq 1$$

Die Aussage dieses Satzes ist stärker als das bis jetzt Bewiesene. Die oben angestellte Überlegung hat die Richtung von links nach rechts hervorgebracht, die Aussage des Satzes gilt aber auch in die andere Richtung. Wird die Kraft'sche Ungleichung erfüllt, so existiert ein präfixfreier Code, dessen Codewörter die gewünschten Längen aufweisen.

Wir werden nun herausarbeiten, wie sich ein solcher Code konstruieren lässt, und gehen dabei zunächst von einem binären Codealphabet aus. Es sei also  $|\Pi| = 2$ , und  $l_1, l_2, \dots, l_n$  sei eine Folge von  $n$  natürlichen Zahlen mit

$$\sum_i \frac{1}{2^{l_i}} \leq 1$$

Wir nehmen an, die Zahlen sind absteigend sortiert: Es ist  $l_i \geq l_{i+1}$ .

Um den gewünschten Code zu konstruieren, bilden wir jetzt nacheinander die Summen  $s_1$  bis  $s_n$  mit

$$s_1 := 0 \quad (3.4)$$

$$s_{i+1} := s_i + 2^{-l_{i+1}} \quad (3.5)$$

und schreiben die ermittelten Zahlenwerte in Binärdarstellung auf. Extrahieren wir von der  $i$ -ten Zahl die ersten  $l_i$  Nachkommaziffern, so entsteht ein Code, dessen Codewörter die gewünschten Längen aufweisen. Abbildung 3.10 verdeutlicht die Konstruktion am Beispiel der Folge

$$4, 4, 3, 3, 2, 2$$

Ein Blick auf die Codewörter zeigt, dass der erzeugte Code tatsächlich präfixfrei ist. Wir werden nun beweisen, dass die Konstruktion immer zu einem präfixfreien Code führt, solange die Codewortlängen die Ungleichung (3.3) erfüllen.

Zunächst folgt aus (3.5) die Beziehung

$$s_{i+j} = s_{i+j-1} + 2^{-l_{i+j}} \geq s_i + 2^{-l_{i+j}}, \quad (1 \leq j \leq n-i)$$

die wir folgendermaßen umschreiben können:

$$s_{i+j} - s_i \geq 2^{-l_{i+j}} \quad (3.6)$$

Wir nehmen jetzt an, der erzeugte Code sei nicht präfixfrei. Dann existieren zwei Summen  $s_i$  und  $s_{i+j}$ , sodass  $s_{i+j}$  in den ersten  $l_{i+j}$  Nachkommastellen mit  $s_i$  übereinstimmt. Die Voraussetzung

$$\sum_i \frac{1}{2^{l_i}} \leq 1$$

sorgt dafür, dass die Vorkommaanteile von  $s_i$  und  $s_{i+j}$  beide gleich 0 sind. Das bedeutet, dass  $s_i$  und  $s_{i+j}$  so nahe beieinander liegen, dass die Differenz weniger als  $2^{-l_{i+j}}$  beträgt:

$$s_{i+j} - s_i < 2^{-l_{i+j}}$$

Dies steht im Widerspruch zu (3.6), sodass wir die Annahme, der Code sei nicht präfixfrei, verwerfen müssen.

Die Konstruktion ist so gestaltet, dass sie einen Binärkode hervorbringt, lässt sich aber ohne Weiteres auf beliebige Codealphabete übertragen. Ist ein Codealphabet  $\Pi$  mit  $b$  Elementen gegeben, müssen wir in (3.5) lediglich den Ausdruck  $2^{-l_{i+1}}$  durch  $b^{-l_{i+1}}$  ersetzen und die ermittelten Zahlenwerte in ihrer  $b$ -adischen Darstellung, d. h. als Zahlen zur Basis  $b$ , aufschreiben (Abbildung 3.11). Damit ist auch die Rückrichtung von Satz 3.1 bewiesen.  $\square$

3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 1

	000
+ 3 <sup>-3</sup>	0,000 0,001
+ 3 <sup>-3</sup>	0,001 0,001
+ 3 <sup>-3</sup>	0,002 0,001
+ 3 <sup>-3</sup>	0,010 0,001
+ 3 <sup>-3</sup>	0,011 0,001
+ 3 <sup>-3</sup>	0,012 0,001
+ 3 <sup>-2</sup>	0,020 0,01
+ 3 <sup>-2</sup>	0,100 0,01
+ 3 <sup>-2</sup>	0,110 0,01
+ 3 <sup>-1</sup>	0,120 0,1
	0,220

$$C = \{000, 001, 002, 010, 011, 012, 020, 10, 11, 12, 2\}$$

**Abb. 3.11:** Die durchgeführte Rechnung bringt einen präfixfreien Code mit einem dreielementigen Codealphabet hervor.

## 3.4 Blockcodes

Wir sprechen immer dann von einem Blockcode bzw. einer Blockcodierung, wenn alle Codewörter die gleiche Länge aufweisen. Blockcodes kommen in vielen Ausprägungen vor, von denen wir uns einige wichtige ansehen wollen.

Unser erstes Beispiel ist eine Codierung über dem Quellenalphabet

$$\Sigma := \{ A, R, N, D, C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, V \} \quad (3.7)$$

und dem Codealphabet

$$\Pi := \{ A, G, C, T \}$$

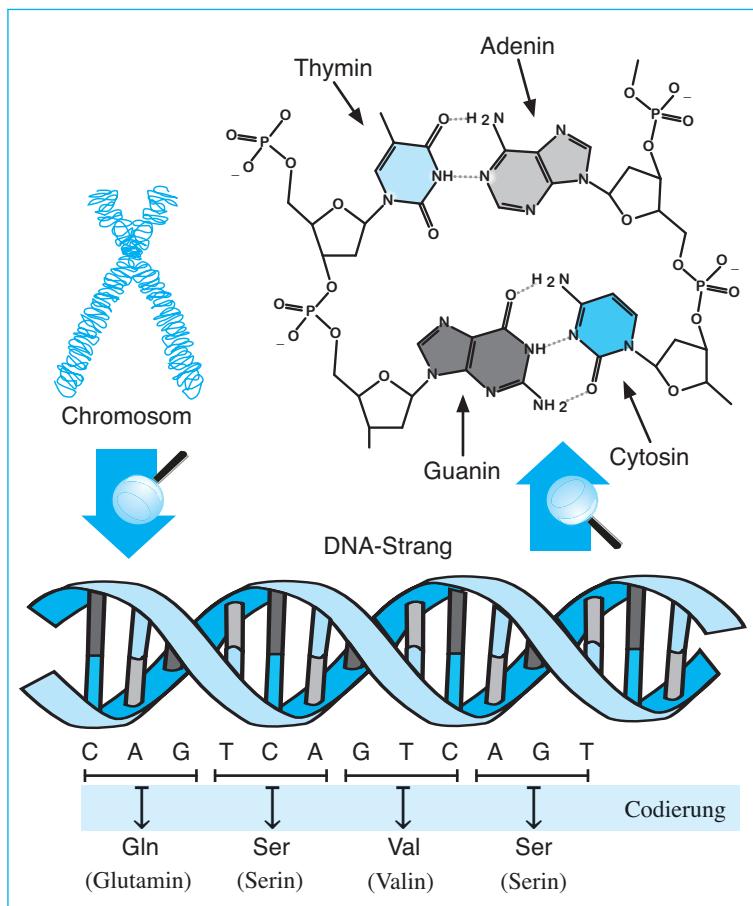
Jedes Quellsymbol wird auf eine Sequenz von drei Codesymbolen – ein sogenanntes *Triplet* – abgebildet. Wie die Codewörter im Einzelnen aussehen, ist in Tabelle 3.2 vermerkt.

Erfunden wurde dieser Code von der Natur. Es handelt sich um den *genetischen Code*, ohne den kein tierisches oder pflanzliches Leben in der uns bekannten Form möglich wäre (Abbildung 3.12). Die in (3.7) genannten Quellsymbole sind die Abkürzungen der insgesamt 20 *Aminosäuren*.

$u$	$c(u)$	Alternatives $c(u)$	$u$	$c(u)$	Alternatives $c(u)$
A	GCT	GCC, GCA, GCG	K	AAA	AAG
R	CGT	CGC, CGA, CGG, AGA, AGG	M	ATG	
N	AAT	AAC	F	TTT	TTC
D	GAT	GAC	P	CCT	CCC, CCA, CCG
C	TGT	TGC	S	TCT	TCC, TCA, TCG, AGT, AGC
Q	CAA	CAG	T	ACT	ACC, ACA, ACG
E	GAA	GAG	W	TGG	
G	GGT	GGC, GGA, GGG	Y	TAT	TAC
H	CAT	CAC	V	GTT	GTC, GTA, GTG
I	ATT	ATC, ATA			
L	TTA	TTG, CTT, CTC, CTA, CTG			

**Tab. 3.2:** Der genetische Code. Die Quellsymbole sind die Abkürzungen der insgesamt 20 *Aminosäuren*, die für den Bau komplexer Proteine benötigt werden. Die Codesymbole sind die Abkürzungen der vier Basen *Guanin* (G), *Cytosin* (C), *Adenin* (A) und *Thymin* (T). Die dritte Spalte zeigt, dass die Natur keine eindeutige Zuordnung gewählt hat. Die meisten Aminosäuren können durch verschiedene Basentripletts codiert werden, was den Organismus vor Mutationen schützt.





**Abb. 3.12:** Der Mensch besteht aus ca. 100 Billionen Zellen, von denen nahezu alle das vollständige Erbgut mit sich tragen [33, 89]. Die Information ist auf den DNA-Strängen von insgesamt 46 Chromosomen codiert, die eng aufgewickelt im Kern einer Zelle liegen. Jedes Chromosom ist ein fadenförmiges Riesenmolekül aus Desoxyribonukleinsäure (deoxyribonucleic acid, kurz DNA). Die DNA hat die Form einer Doppelhelix, die durch innenliegende Sprossen verbunden ist. Jede Sprosse wird durch zwei Nukleotide gebildet, die aus jeweils einem Zucker, einem Phosphat und einer der vier möglichen Basen Guanin (G), Cytosin (C), Adenin (A) oder Thymin (T) bestehen. Würden wir die DNA-Stränge aller 46 Chromosomen entwirren und hintereinander legen, so erhielten wir einen Strang, der eine Länge von rund 2 m erreichen und ca. 3,27 Milliarden Basenpaare enthalten würde. Die Nukleotide treten immer in komplementären Paaren auf. Adenin und Thymin bilden ein solches Paar, genauso wie Cytosin und Guanin. Trennen wir die Helix in der Mitte auf, so entstehen zwei zueinander komplementäre Stränge, die eine Abfolge von Gs, Cs, As und Ts codieren. Diese Abfolge ist die Erbinformation; jeder Mensch trägt sie in sich – in fast jeder Zelle.

nosäuren, aus denen in den Ribosomen der Körperzellen komplexe Proteine – die Grundbausteine aller Lebewesen – gebildet werden. Die Codesymbole sind die Abkürzungen der vier Basen Guanin (G), Cytosin (C), Adenin (A) und Thymin (T), von denen jeweils drei eine Aminosäure codieren.

Wie clever die Natur vorgegangen ist, zeigt die dritte Spalte von Tabelle 3.2. Sie macht klar, dass die Zuordnung nicht eindeutig ist und für die Codierung von Aminosäuren in den meisten Fällen mehrere Möglichkeiten zur Verfügung stehen. Besonders deutlich wird dies am Beispiel der Aminosäuren Arginin (R), Leucin (L) und Serin (S). Für alle drei existieren jeweils fünf verschiedene Codierungsmöglichkeiten. Die vorhandene Mehrdeutigkeit ist keine willkürliche Laune der Natur, sondern ein leistungsfähiges Mittel zur Fehlerkorrektur. Wird beispielswei-

A (Ala)  Alanin	R (Arg)  Arginin	N (Asn)  Asparagin	D (Asp)  Asparaginsäure	C (Cys)  Cystein
Q (Gln)  Glutamin	E (Glu)  Glutaminsäure	G (Gly)  Glycin	H (His)  Histidin	I (Ile)  Isoleucin
L (Leu)  Leucin	K (Lys)  Lysin	M (Met)  Methionin	F (Phe)  Phenylalanin	P (Pro)  Prolin
S (Ser)  Serin	T (Thr)  Threonin	W (Trp)  Tryptophan	Y (Tyr)  Tyrosin	V (Val)  Valin

**Abb. 3.13:** Aminosäuren sind die Grundbausteine der Proteine. Als abkürzende Notationen haben sich eine Einbuchstaben- und eine Dreibuchstabenschreibweise etabliert, die hier beide aufgeführt sind. Das Quellenalphabet (3.7) verwendet die kompakte Einbuchstabenbeschreibung.

se die dritte Base in der Sequenz GCT durch eine Mutation beschädigt, so wirkt sich dies nicht auf den Organismus aus; es wird auch weiterhin die Aminosäure Alanin (A) synthetisiert.

In den nächsten Abschnitten werden wir weitere Blockcodes betrachten. Wir beginnen mit der Vorstellung bekannter Zeichen- und Zahlencodes und beschäftigen uns im Anschluss daran ausführlich mit der wichtigen Klasse der linearen Blockcodes.

Brailleschrift (Auszug)						Äquivalenter Binärkode					
· ·	· ·	· ·	· ·	· ·	· ·	a	100000	m	110010	y	110111
· ·	· ·	· ·	· ·	· ·	· ·	b	101000	n	110110	z	100111
· ·	· ·	· ·	· ·	· ·	· ·	c	110000	o	100110	1	100000
· ·	· ·	· ·	· ·	· ·	· ·	d	110100	p	111010	2	101000
· ·	· ·	· ·	· ·	· ·	· ·	e	100100	q	111110	3	110000
· ·	· ·	· ·	· ·	· ·	· ·	f	111000	r	101110	4	110100
· ·	· ·	· ·	· ·	· ·	· ·	g	111100	s	011010	5	100100
· ·	· ·	· ·	· ·	· ·	· ·	h	101100	t	011110	6	111000
· ·	· ·	· ·	· ·	· ·	· ·	i	011000	u	100011	7	111100
· ·	· ·	· ·	· ·	· ·	· ·	j	011100	v	101011	8	101100
· ·	· ·	· ·	· ·	· ·	· ·	k	100010	w	011101	9	011000
· ·	· ·	· ·	· ·	· ·	· ·	l	101010	x	110011	0	011100
5	6	7	8	9	0						

Abb. 3.14: Die Brailleschrift ist ein klassischer Blockcode. Alle Codewörter haben die gleiche Länge.

### 3.4.1 Zeichencodes

#### Brailleschrift

Auf der linken Seite in Abbildung 3.14 ist ein bekannter Zeichencode zu sehen, der nach dem Franzosen Louis Braille benannt ist. Braille hatte sie im frühen 19. Jahrhundert entwickelt, um blinden Menschen das Lesen zu ermöglichen.

In der *Brailleschrift* wird jeder Buchstabe durch eine Punktmatrix repräsentiert, die drei Zeilen hoch und zwei Spalten breit ist. Gelesen wird ein Text durch das Überstreichen mit dem Finger. Damit ein Schriftsymbol taktil erfasst werden kann, werden die dick dargestellten Punkte mit einem Brailledrucker so von hinten in Papier gepresst, dass sie als kleine Erhebungen fühlbar werden.

Die Brailleschrift ist zu dem binären Blockcode äquivalent, der in Abbildung 3.14 in der rechten Spalte zu sehen ist. Dieser ist ganz einfach dadurch entstanden, dass die zweidimensionalen Muster Punkt für Punkt in eine 6-stellige Binärzahl übertragen wurden.



Louis Braille (Abbildung 3.15) wurde am 4. Januar 1809 in Coupvray, einem kleinen Dorf südöstlich von Paris, geboren. Aufgewachsen ist er, zusammen mit drei älteren Geschwistern, auf dem Land. Sein Vater war Ledermacher und betrieb auf dem Hof der Familie eine Sattlerei. Im Alter von drei Jahren verletzte sich Louis beim Spielen in der väterlichen Werkstatt mit einem Spitzbohrer eines seiner Augen. Die Wunde infizierte sich und griff wenige Wochen später auf das gesunde Auge über. Louis nahm die Welt jetzt nur noch durch einen grauen Schleier wahr, der sich von Tag zu Tag verdunkelte. Die Ärzte konnten dem Jungen nicht helfen und so war sein Augenlicht wenige Monate später unwiederbringlich verloren. Seine Eltern versuchten, ihren erblindeten Sohn so gut wie möglich zu fördern, und schickten ihn im Alter von 10 Jahren an das *Institut National des Jeunes Aveugles*, kurz INJA, nach Paris. Das INJA wurde im Jahr 1784 von Valentin Haüy gegründet und ist die älteste Blindenschule der Welt. Braille lernte dort eine Punktschrift kennen, die von Charles Barbier de La Serre, einem Hauptmann der französische Armee, für die militärische Kommunikation bei Nacht entwickelt wur-

de. Hierfür sah Barbier spezielle Punktmuster vor, die für die Codierung der Zeilen- und Spaltennummern einer Silbenmatrix verwendet wurden. Braille erkannte das Potenzial dieser Schrift und arbeitete in den Folgejahren intensiv an einer Vereinfachung. Im Alter von 16 Jahren war sein Projekt vollendet. Er hatte eine tastbare Schrift geschaffen, die jedes Symbol mit 6 Punkten codierte und viel einfacher zu erlernen war als die Codes, die ihm als Vorbild dienten. Trotzdem setzte sich die Schrift, die heute seinen Namen trägt und vielen Generationen von sehbehinderten Menschen das Lesen mit den Händen ermöglicht hat, damals nur langsam durch. Erst im Jahr 1850 wurde sie von der Pädagogischen Akademie in Frankreich offiziell anerkannt und es dauerte mehr als 20 weitere Jahre, bis sie auch im Ausland akzeptiert wurde. Braille hatte den internationalen Durchbruch seiner Schrift nicht mehr erlebt. Er starb an der damals so töckischen Tuberkulose und wurde – im jungen Alter von 43 Jahren – in seinem Geburtsort beigesetzt. An seinem hundertsten Todestag wurde sein Leichnam exhumiert und in eine Grabkammer im Pariser Panthéon gebracht. Dort hat Louis Braille seine letzte, ehrende Ruhestätte gefunden.

## ASCII



Louis Braille  
(1809 – 1852)

Eine der bekanntesten und im Computerbereich wichtigsten Zeichencodes ist der *American Standard Code for Information Interchange*, kurz ASCII. Obwohl seine historischen Wurzeln bis in die Anfänge der 60er-Jahre zurückreichen, wird der Code auch heute noch von fast allen Betriebssystemen und Programmiersprachen unterstützt.

In seiner Reinform besteht er aus 128 Zeichen, die mit jeweils 7 Bit codiert werden (Tabelle 3.3). Neben den Schriftzeichen existiert eine Reihe von Steuerbefehlen, die unter anderem der Formatierung der Ausgabe auf einem textbasierten Terminal dienen. Mit der Einführung grafischer Benutzeroberflächen haben diese Zeichen ihre Bedeutung verloren und spielen daher kaum noch eine Rolle.

Die ASCII-Tabelle, wie wir sie heute verwenden, hat sich über mehrere Jahre entwickelt. In ihrer ursprünglichen Form von 1963 bestand sie aus nur 99 Einträgen, die sich auf diverse Steuerzeichen, Ziffern und Großbuchstaben verteilten. Insbesondere sah die ursprüngliche Version keinerlei Möglichkeit zur Codierung von Kleinbuchstaben vor. Später wurde die Tabelle ergänzt und der ASCII-Code im Jahre 1974 unter der

Abb. 3.15: Bronzefigur von Louis Braille

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/	
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL
	NUL	Null value	BS	Backspace	DLE	Data Link Escape	ESC	Escape								
	SOH	Start of Heading	HT	Horizontal Tabulation	DC	Device Control	FS	File Separator								
	STX	Start of Text	LF	Line Feed	NAK	Negative Acknowledge	GS	Group Separator								
	ETX	End of Text	VT	Vertical Tabulation	SYN	Synchronous idle	RS	Record Separator								
	EOT	End of Transmission	FF	Form Feed	ETB	End of Transmission Block	US	Unit Separator								
	ENQ	Enquiry	CR	Carriage Return	CAN	Cancel	SP	Space								
	ACK	Acknowledge	SO	Shift-Out	EM	End of Medium	DEL	Delete								
	BEL	Bell	SI	Shift-In	SUB	Substitute character										

**Tab. 3.3:** Der *American Standard Code for Information Interchange* (ASCII) ist der Dinosaurier unter den Zeichencodes.

Normenbezeichnung ISO 646 als internationaler Standard festgeschrieben.

Im Hauptspeicher werden ASCII-Texte byteweise abgelegt und somit jedes Zeichen mit 8 Bit repräsentiert. Da die Codewörter der einzelnen Zeichen nur jeweils 7 Bit umfassen, entstehen mehrere Möglichkeiten für den Umgang mit dem zusätzlichen Bit. Auf älteren Rechnersystemen wurde das freie Bit manchmal als *Paritätsbit* verwendet. Hierzu wird das ASCII-Bitmuster eines Zeichens so um ein zusätzliches Bit erweitert, dass die Anzahl der Einsen gerade wird.

Moderne Computersysteme füllen das Bitmuster mit einer 0 auf. Auf diese Weise lässt sich der ASCII-Zeichensatz auf einen 8-Bit-Code erweitern, ohne die Kompatibilität mit alten Texten zu verlieren. In der Tat arbeiten heute fast alle Computersysteme mit erweiterten Zeichensätzen, die in den unteren 128 Zeichen mit dem ASCII-Code übereinstimmen. Eine hervorgehobene Rolle spielen in diesem Zusammenhang die Zeichensätze der ISO 8859 (Abbildung 3.16).

## ISO 8859

Für den westeuropäischen Raum ist vor allem die Zeichensatztabelle ISO 8859-1 (Latin-1) von Bedeutung, die neben verschiedenen franzö-

ISO 8859 – x	
Westeuropäisch (Latin-1)	1
Osteuropäisch (Latin-2)	2
Südeuropäisch (Latin-3)	3
Baltisch (Latin-4)	4
Kyrillisch	5
Arabisch	6
Griechisch	7
Hebräisch	8
Türkisch (Latin-5)	9
Nordisch (Latin-6)	10
Thai	11
Noch nicht verabschiedet	12
Baltisch (Latin-7)	13
Keltisch (Latin-8)	14
Westeuropäisch (Latin-9)	15
Südosteuropäisch (Latin-10)	16

**Abb. 3.16:** Die Teilnormen der ISO 8859



ਏ ਓ ਔ ਕ ਖ ਗ  
 ਠ ਡ ਡ ਣ ਤ ਥ ਵ ਧ  
 ਰ ਰ ਲ ਲ ਲ ਵ ਸ਼ ਷  
 ਮੈ ਅੁ ਨ ਦ ਨ ਹ  
 ਅੱ ਹ ਨ ਹ  
 ਕੁ ਲੁ ਅੁ ਲੁ | || ੦ ੧  
 ਝ ਤ ਊ ਕੁ ਲੁ ਏ ਏ ਏ  
 ਘ ਭ ਚ ਛ ਜ ਝ ਅ ਟ  
 ਨ ਨ ਪ ਫ ਬ ਭ ਮ ਯ  
 ਸ ਹ || . ੫ ਤ ਫ  
 ਾ ਂ ਂ ਂ ਂ , ਿ  
 ਕ ਖ ਗ ਝ ਡ ਧ ਫ ਯ  
 ੨ ੩ ੪ ੫ ੬ ੭ ੮ ੯  
 ਜ ਬ ਗ ਜ ? ਭ ਬ

**Abb. 3.17:** Ein kleiner Ausschnitt aus dem Symbolvorrat des Unicodes. Im Bereich zwischen 0x0900 und 0x097F befinden sich die Schriftzeichen des *Devanagari-Alphabets*. Sie sind die Grundlage mehrerer indischer Sprachen wie Hindi, Sanskrit, Marathi, Kashmiri, Sindhi und Nepali.

sischen und skandinavischen Sonderzeichen auch die deutschen Umlaute und den scharfen ß-Laut enthält. Deutsche Texte lassen sich allerdings auch mit den meisten anderen Zeichensatztabellen verfassen, da die entsprechenden Sonderzeichen nicht nur im Latin-1-Zeichensatz, sondern auch in allen anderen Latin-Zeichensätzen außer Latin-7 vorhanden sind.

Obwohl die Zeichentabellen der ISO 8859 gegenüber dem reinen ASCII-Code einen Fortschritt darstellen, bleiben aufgrund der geringen Bitbreite von 8 Bit pro Zeichen einige der ursprünglichen Probleme bestehen. Beispielsweise existiert sowohl für die deutschen als auch für die russischen Schriftzeichen eine entsprechende Zeichentabelle, es gibt jedoch keine, die beide Schriftzeichen in sich vereint. Das Verfassen eines Textes, der neben deutschen Passagen auch russische enthält, bereitet hierdurch erhebliche Probleme. Ferner existieren Sprachen, insbesondere aus dem asiatischen Raum, die das 255-Zeichen-Repertoire der ISO-Tabellen bei weitem sprengen. Aus diesem Grund wurden Mitte der 80er-Jahre Codierungen entwickelt, die ein Zeichen mit mehr als 8 Bit repräsentieren und den Darstellungsraum auf diese Weise drastisch vergrößern. Die wichtigste Codierung dieser Art ist der *Unicode* [2, 51].

## Unicode

Der Unicode ist eine universelle Symboltabelle, die jedem bekannten Zeichen ein eindeutig bestimmtes Codewort zuordnet. Die Codewörter sind auf jeder Hardware, unter jedem Betriebssystem und in jeder Programmiersprache stets dieselben, sodass Interoperabilitätsprobleme aufgrund falsch ausgewählter oder inkompatibler Codetabellen nicht mehr möglich sind.

Der Coderaum des Unicodes umfasst insgesamt 17 Bereiche, sogenannte *planes*, die jeweils 65.536 verschiedene Zeichen aufnehmen können. Damit stehen insgesamt mehr als eine Million Codewörter zur Verfügung, von denen bisher jedoch nur knapp 10 % einem Zeichen zugewiesen sind. Zukünftige Erweiterungen des Unicodes müssen sich um Platzprobleme daher keine Sorgen machen.

Die für uns wichtigsten Zeichen und Symbole befinden sich in der *plane 0*, der *Basic Multilingual Plane* (BMP), von der ein kleiner Auszug in Abbildung 3.17 dargestellt ist. Der Unicode versucht eine möglichst große Rückwärtskompatibilität mit den existierenden Zeichencodes zu gewährleisten. So sind die ersten 256 Zeichen der BMP identisch mit der Latin-1-Zeichentabelle der ISO 8859-1.

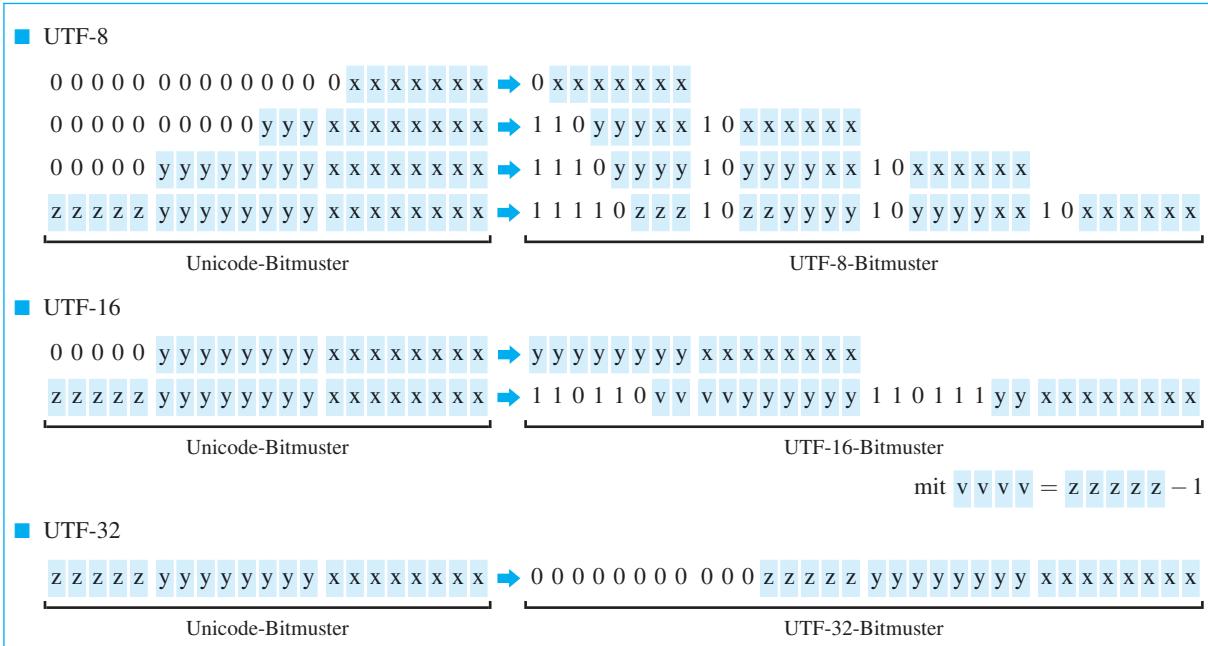


Abb. 3.18: Konvertierung von Unicode-Zeichen in das UTF-Format

Jedes der 65.536 Zeichen aus der BMP lässt sich mit 16 Bit eindeutig codieren. Angewendet wird dieses Prinzip zum Beispiel in der Programmiersprache Java, die alle Zeichenketten intern im Unicode ablegt und für jedes Zeichen 2 Byte im Hauptspeicher reserviert. Auf Datenträgern werden im Unicode verfasste Textdateien gerne im *Universal Transformation Format*, kurz UTF, gespeichert. Genauer handelt es sich hierbei um eine Formatfamilie, die mit UTF-8, UTF-16 und UTF-32 drei verschiedene Unterformate definiert. Abbildung 3.18 zeigt, wie die Bitsequenz eines Unicode-Zeichens in den verschiedenen Unterformaten dargestellt wird.

Das UTF-8-Format ist besonders zur Darstellung von Texten geeignet, die viele ASCII-Zeichen beinhalten. Jedes ASCII-Zeichen wird exakt mit dem gleichen Bitmuster codiert, sodass das UTF-8-Format vollständig rückwärtskompatibel ist. Die Unicode-Zeichen oberhalb der 128-Zeichen-Grenze werden mit 16, 24 oder 32 Bit dargestellt. Zu erkennen ist ein verlängertes Codewort an den führenden Einsen im Bitmuster. Die Anzahl der Einsen ist bedeutend: Sie entspricht der Anzahl der Bytes, die zur Codierung des Zeichens benötigt werden. Jedes der Folgebytes wird mit dem Bitmuster 10 eingeleitet, sodass der Beginn und

das Ende einer Zeichensequenz auch dann sicher erkannt werden kann, wenn erst in der Mitte einer laufenden Übertragung mit dem Mitlesen des Textes begonnen wird.

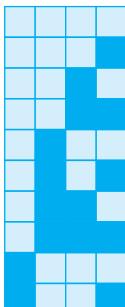
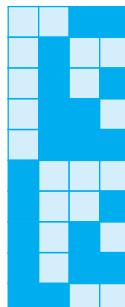
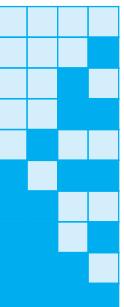
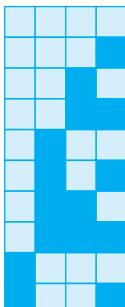
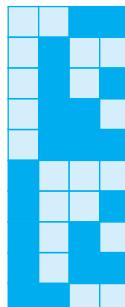
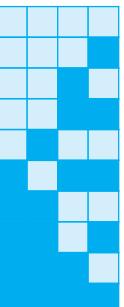
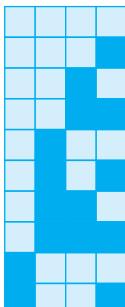
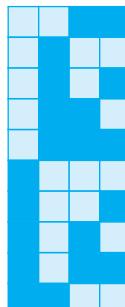
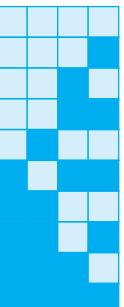
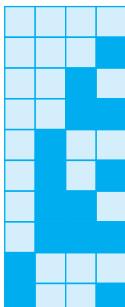
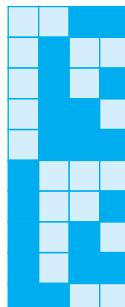
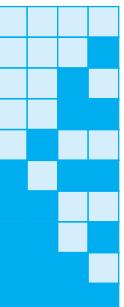
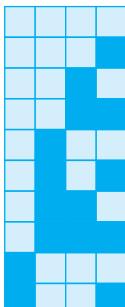
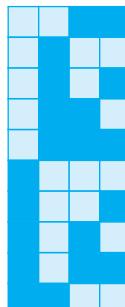
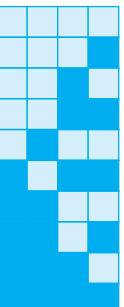
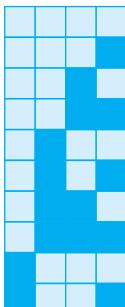
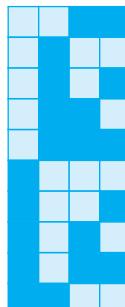
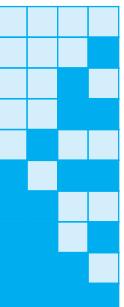
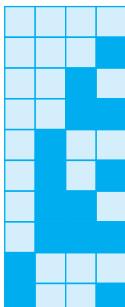
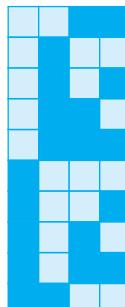
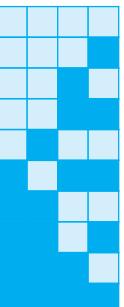
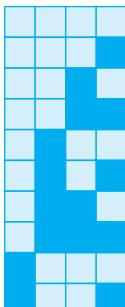
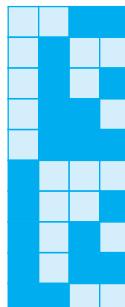
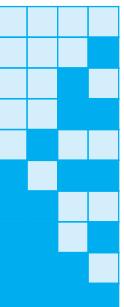
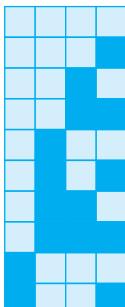
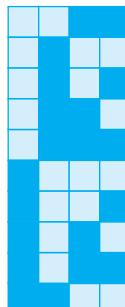
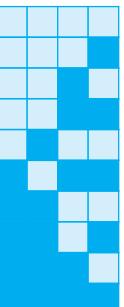
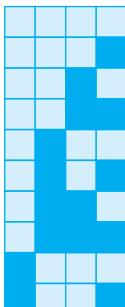
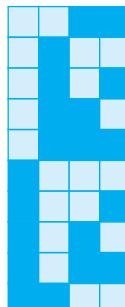
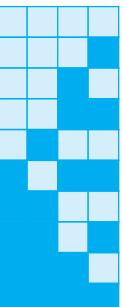
Enthält ein Text nur wenige ASCII-Zeichen, dafür aber viele Zeichen der Basic Multilingual Plane, so bietet sich die Codierung im UTF-16-Format an. Hier werden die ersten 65.536 Zeichen, d. h. alle Unicode-Zeichen der BMP, mithilfe von zwei Byte codiert. Für die Codierung höher positionierter Zeichen werden 32 Bit verwendet, von denen die ersten 16 als *High-Surrogate* und die nachfolgenden 16 als *Low-Surrogate* bezeichnet werden. Beide beginnen zur eindeutigen Unterscheidung mit einer individuellen Bitsequenz. Im UTF-16-Format ist zu beachten, dass die ersten 5 Unicode-Bits durch die Subtraktion von eins auf 4 Bits komprimiert werden. Die Subtraktion kann bedenkenlos vorgenommen werden, da der einzige kritische Fall  $zzzz = 0000$  bereits durch die 16-Bit-Codierung abgedeckt ist.

UTF-32 ist das letzte und einfachste Format der UTF-Familie. Jedes Unicode-Zeichen wird ausnahmslos mit 4 Byte repräsentiert. In der Konsequenz benötigt ein UTF-32-codierter ASCII-Text genau viermal so viel Speicherplatz wie der gleiche Text im UTF-8-Format. Selbst Texte, die sämtliche Zeichen der BMP verwenden, belegen im UTF-32-Format immer noch doppelt so viel Speicherplatz wie der gleiche Text in UTF-16. Auf der positiven Seite besticht der Code durch seine einfache Handhabung. So lässt sich die Position des  $n$ -ten Zeichens innerhalb der Bytesequenz direkt berechnen, und an der Größe einer UTF-32-Datei lässt sich sofort erkennen, wie viele Zeichen der gespeicherte Text umfasst.

### 3.4.2 Zahlencodes

In diesem Abschnitt werden wir mehrere bekannte Blockcodes betrachten, die uns zur Codierung von Zahlenwerten zur Verfügung stehen. Besonders wichtig sind in diesem Zusammenhang die *Tetraden-Codes*, die für die ziffernweise Codierung von Dezimalzahlen geschaffen wurden. Jede Ziffer wird durch eine vier Bit breite *Tetrade* dargestellt. Da wir im Falle von Dezimalzahlen nur 10 der 16 möglichen Bitmuster benötigen, um die Ziffern 0 bis 9 unterscheiden zu können, existieren in jedem Tatreden-Code 6 *Pseudo-Tetraden*, die keine Bedeutung besitzen.

In Tabelle 3.4 sind mehrere Codes zusammengefasst, die in der Praxis eine Rolle spielen.

BCD-Code		Stibitz-Code		Aiken-Code		Gray-Code		
0		0000	0		0011	0		0000
1		0001	1		0100	1		0001
2		0010	2		0101	2		0010
3		0011	3		0110	3		0011
4		0100	4		0111	4		0100
5		0101	5		1000	5		1011
6		0110	6		1001	6		1100
7		0111	7		1010	7		1101
8		1000	8		1011	8		1110
9		1001	9		1100	9		1111

**Tab. 3.4:** Tetraden-Codes in der Übersicht

### ■ BCD-Code

Der BCD-Code ist der am häufigsten eingesetzte Tetraden-Code. Er weist den einzelnen Bits innerhalb einer Tetrade die Wertigkeiten 8, 4, 2 und 1 zu und wird aus diesem Grund auch als *8421-Code* bezeichnet. Im BCD-Code können wir jede Tetrade wie eine gewöhnliche Binärzahl interpretieren, sodass sich jede Dezimalzahl direkt in das BCD-Format übersetzen lässt. Das Codewort einer Ziffer entsteht ganz einfach dadurch, dass deren Binärdarstellung durch das Voranstellen von Nullen auf vier Bit ergänzt wird.

### ■ Stibitz-Code

Die Bitmuster des *Stibitz-Codes* sind im Vergleich zu den Bitmustern der BCD-Darstellung um drei verschoben. Aus diesem Grund wird der Stibitz-Code auch als *Excess-3-Code* bezeichnet. Im Gegensatz zum BCD-Code kann das *Neunerkomplement*, das für eine Dezimalziffer  $z$  dem Wert  $9 - z$  entspricht, ganz einfach durch das Invertieren aller Bits berechnet werden. Das Neunerkomplement wird zum Ausführen der Subtraktion benötigt, sodass sich arithmetische Operationen im Stibitz-Code einfacher ausführen lassen als im BCD-Code.

### ■ Aiken-Code

Der Aiken-Code teilt mit dem Stibitz-Code die Eigenschaft, dass sich das Neunerkomplement durch das Invertieren aller Bits erzeugen lässt. Anders als im Stibitz-Code lässt sich jeder Bitposition mit 2, 4, 2 und 1 jetzt wieder eine feste Stelligkeit zuordnen. Aus diesem Grund wird der Aiken-Code auch gerne als *2421-Code* bezeichnet.

0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
Erweiterung ...	
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000

**Tab. 3.5:** Der erweiterte Gray-Code entsteht aus dem Gray-Code durch die Hinzunahme von 6 weiteren Bitmustern.

### ■ Gray-Code

Der *Gray-Code* fällt in die Klasse der *progressiven* oder *einschrittigen Codes*, da sich die Bitmuster zweier benachbarter Ziffern in genau einem Bit unterscheiden [37]. Tabelle 3.4 zeigt, dass diese Eigenschaft beim Wechsel von 9 auf 0 verloren geht. Abhilfe schafft der *erweiterte Gray-Code*, der alle Zahlen zwischen 0 und 15 in einer einzigen Tetrade codiert (Tabelle 3.5). Bei einem Überlauf geht das Bitmuster 1000 in dem erweiterten Code durch den Wechsel eines einzigen Bits in das Bitmuster 0000 über.

Da das Rechnen mit progressiven Codes schwierig ist, bietet es sich an, den Gray-Code vor der Durchführung arithmetischer Operationen in den Binärcode zu übersetzen. Computergestützt kann dies mit einem Programm oder mithilfe sehr einfacher Hardwareschaltungen geschehen (Abbildung 3.19).

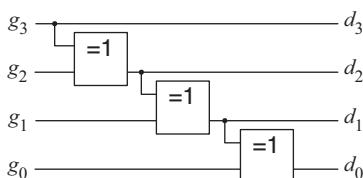
## 3.4.3 Lineare Codes

In diesem Abschnitt betrachten wir Blockcodes, deren Codealphabet  $\Pi$  ein endlicher Körper  $\mathbb{F}$  ist. Jedes Codewort können wir dann als ein Element des Vektorraums  $\mathbb{F}^n$  auffassen, wobei  $n$  für die Länge der Codewörter steht. Als Beispiele sind in Abbildung 3.20 vier verschiedene Codierungen zu sehen, deren Codewörter eine Teilmenge des Vektorraums  $\mathbb{Z}_2^3$  sind.

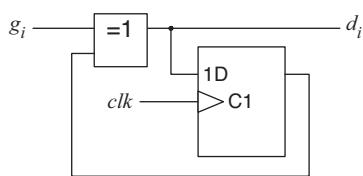
Die Beispiele lehren uns, dass wir zuweilen pedantisch zwischen Codierungen und Codes unterscheiden müssen.  $c_1$ ,  $c_2$  und  $c_3$  sind verschiedene Codierungen, da sie die Quellsymbole unterschiedlich abbilden. Nichtsdestotrotz erzeugen sie alle den gleichen Code, da die Mengen ihrer Codewörter identisch sind.

Die drei ersten Codierungen weisen eine weitere Besonderheit auf. Der Code, den sie erzeugen, ist nicht nur eine Teilmenge von  $\mathbb{Z}_2^3$ , sondern gleichzeitig ein Untervektorraum. Codes mit dieser Eigenschaft tragen einen eigenen Namen: Sie heißen *linear*.

### ■ Parallele Konvertierung



### ■ Serielle Konvertierung



**Abb. 3.19:** Hardwareschaltungen für die Konvertierung des Gray-Codes in das Binärsystem



### Definition 3.4 (Linearer Code)

Sei  $\mathbb{F}^n$  ein Vektorraum über einem endlichen Körper  $\mathbb{F}$ . Die Menge

$C \subseteq \mathbb{F}^n$  heißt *linearer*  $[n, k]$ -Code,

wenn sie ein  $k$ -dimensionaler Untervektorraum von  $\mathbb{F}^n$  ist.

Ab und an sprechen wir auch von *linearen Codierungen* und meinen damit Codierungen, die einen linearen Code erzeugen. Folgerichtig sind  $c_1$ ,  $c_2$  und  $c_3$  lineare Codierungen, nicht aber  $c_4$ .

## Paritätscode

$c_1$ ,  $c_2$  und  $c_3$  erzeugen den geraden *Paritätscode (parity code)*. Sein Name geht auf die Eigenschaft seiner Codewörter zurück, eine gerade Anzahl an Einsen aufzuweisen. Tatsächlich handelt es sich bei dem Paritätscode um einen alten Bekannten. Zeichnen wir die Codewörter grafisch auf, so entsteht der Untervektorraum aus Abbildung 3.21, dem wir auf Seite 146 schon einmal begegnet sind.

Durch die Vorarbeit, die wir in Abschnitt 2.5 geleistet haben, bereitet uns der mathematische Umgang mit linearen Codes jetzt keine Probleme mehr. Insbesondere wissen wir schon, dass sich Untervektorräume mithilfe von Generatormatrizen erzeugen lassen. Eine solche Matrix entsteht, indem wir eine Basis  $\{\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_k\}$  bestimmen und die darin enthaltenen Vektoren zeilenweise aufschreiben. Ist  $n$  die Länge der Codewörter, so erhalten wir eine Matrix  $G$  mit  $k$  Zeilen und  $n$  Spalten, die für die Erzeugung der Codewörter genutzt werden kann. Die Codewörter entstehen, indem die Vektoren aus  $\mathbb{F}^k$  nacheinander von links mit der Matrix multipliziert werden:

$$C = \{(u_0 \ \dots \ u_{k-1}) \cdot G \mid (u_0 \ \dots \ u_{k-1}) \in \mathbb{F}^k\}$$

Der Vektorraum aus Abbildung 3.21 wird durch die Basisvektoren

$$\left\{ \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \right\}$$

aufgespannt, sodass die folgende Matrix unseren Zweck erfüllt:

$$G_1 := \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \quad (3.8)$$

Beachten Sie, dass eine Generatormatrix  $G_1$  nicht nur einen Code, sondern immer auch eine Codierung beschreibt, die auf natürliche Weise über die Zuordnung

$$(u_0 \ \dots \ u_{k-1}) \mapsto (u_0 \ \dots \ u_{k-1}) \cdot G_1 \quad (3.9)$$

gegeben ist. Der obere Teil von Abbildung 3.22 macht deutlich, dass die gewählte Matrix die Codierung  $c_1$  hervorbringt.

### Codierung $c_1$

$u$	$c_1(u)$
00	000
01	011
10	101
11	110

### Codierung $c_2$

$u$	$c_2(u)$
00	000
01	101
10	011
11	110

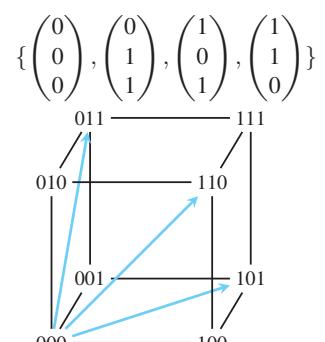
### Codierung $c_3$

$u$	$c_3(u)$
00	110
01	101
10	011
11	000

### Codierung $c_4$

$u$	$c_4(u)$
00	000
01	001
10	010
11	100

**Abb. 3.20:** Alle Codewörter sind Elemente des Vektorraums  $\mathbb{Z}_2^3$ .



**Abb. 3.21:** Der Paritätscode, dargestellt als Untervektorraum von  $\mathbb{Z}_2^3$

■ Generatormatrix  $G_1$

$$(0 \ 0) \cdot \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} = (0 \ 0 \ 0) \quad (0 \ 1) \cdot \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} = (0 \ 1 \ 1)$$

$$(1 \ 0) \cdot \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} = (1 \ 0 \ 1) \quad (1 \ 1) \cdot \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} = (1 \ 1 \ 0)$$

$u$	$c_1(u)$
00	000
01	011
10	101
11	110

■ Generatormatrix  $G_2$

$$(0 \ 0) \cdot \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} = (0 \ 0 \ 0) \quad (0 \ 1) \cdot \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} = (1 \ 0 \ 1)$$

$$(1 \ 0) \cdot \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} = (0 \ 1 \ 1) \quad (1 \ 1) \cdot \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} = (1 \ 1 \ 0)$$

$u$	$c_2(u)$
00	000
01	101
10	011
11	110

Abb. 3.22: Die Generatormatrizen  $G_1$  (3.8) und  $G_2$  (3.10) erzeugen die Codierungen  $c_1$  und  $c_2$  aus Abbildung 3.20.

Der untere Teil von Abbildung 3.22 zeigt, dass wir für  $c_2$  ebenfalls eine passende Matrix finden können. Sie entsteht aus der Matrix (3.8) durch die Vertauschung der ersten beiden Spalten:

$$G_2 := \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \quad (3.10)$$

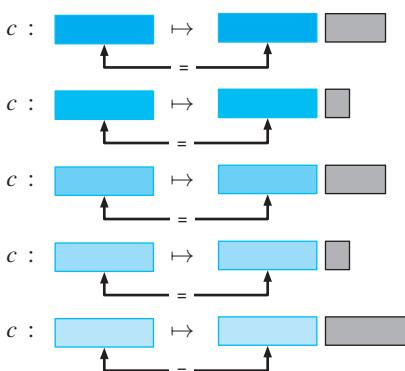


Abb. 3.23: Systematische Codierungen sind so beschaffen, dass die Originalnachricht lediglich verlängert wird, aber ansonsten unverändert bleibt.

Im Falle von  $c_3$  müssen wir kapitulieren. Obwohl es sich auch hier um eine lineare Codierung handelt, können wir die gewählte Zuordnung nicht durch die Multiplikation mit einer Generatormatrix berechnen. Warum dies so ist, lässt sich leicht erklären. Multiplizieren wir den Nullvektor  $\mathbf{0}$  mit einer Matrix, so erhalten wir als Ergebnis immer den Nullvektor zurück, unabhängig von der Wahl der Matrixkoeffizienten. Das bedeutet, dass wir überhaupt nur dann eine Chance haben, eine passende Matrix zu finden, wenn die Nachricht  $0 \dots 0$  auf das Codewort  $0 \dots 0$  abgebildet wird.

Von den drei besprochenen Codierungen ist  $c_1$  die wichtigste. Obwohl  $c_2$  und  $c_3$  ebenfalls den Paritätscode hervorbringen, ist  $c_1$  diejenige, die mit diesem Code auf natürliche Weise verbunden ist. Sie ist so gestaltet, dass die Codewörter durch das Anhängen eines Prüfbits, des sogenannten *Paritätsbits*, entstehen.

Eine Codierung wie  $c_1$ , die eine Nachricht lediglich verlängert, besitzt einen besonderen Namen: Sie heißt *systematisch* (Abbildung 3.23).

3-Bit-Paritätscode		4-Bit-Paritätscode		5-Bit-Paritätscode	
$u$	$c(u)$	$u$	$c(u)$	$u$	$c(u)$
00	000	000	0000	0000	00000
01	011	001	0011	0001	00011
10	101	010	0101	0010	00101
11	110	011	0110	0011	00110
$\{ \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \}$		$\{ \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} \}$		$\{ \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} \}$	

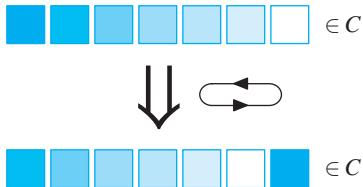
Abb. 3.24: Der gerade Paritätscode ist ein linearer Blockcode. In allen Codewörtern ist die Anzahl der Einsen gerade.



### Definition 3.5 (Systematische Codierung)

Ein Codierung heißt *systematisch*, wenn die codierte Nachricht die Originalnachricht als Anfangsstück enthält.

Abbildung 3.24 zeigt, dass sich längere Paritätscodes auf die gleiche Weise erzeugen lassen. Die Generatormatrix des Paritätscodes der Länge  $n+1$  entsteht, indem die  $n \times n$ -Einheitsmatrix um eine mit Einsen gefüllte Spalte ergänzt wird. Multiplizieren wir einen Vektor  $u$  von links, so sorgt die Einheitsmatrix dafür, dass der Anfang des Codeworts mit  $u$  übereinstimmt. Die zusätzliche Spalte führt dazu, dass das letzte Bit die



**Abb. 3.25:** Zyklische Codes besitzen die Eigenschaft, dass die zyklische Verschiebung (Rotation) eines Codeworts wieder zu einem Codewort führt.

Summe der anderen Bits ist: Da die Berechnung modulo 2 erfolgt, erhalten wir als Ergebnis genau dann eine 1, wenn die Anzahl der Einsen in  $u$  ungerade ist.

Der Paritätscode besitzt eine weitere elementare Eigenschaft. Rotieren wir ein Codewort, indem wir auf einer Seite ein Teilstück entfernen und dieses an der anderen Seite wieder anhängen, so entsteht erneut ein Codewort. Solche Codes heißen *zyklisch* (Abbildung 3.25).



### Definition 3.6 (Zyklischer Code)

Ein Code heißt *zyklisch*, wenn jede zyklische Verschiebung eines Codeworts ebenfalls ein Codewort ist.

## Hamming-Code

Als nächstes Beispiel betrachten wir den Untervektorraum von  $\mathbb{Z}_2^7$ , der durch die Basisvektoren aus Abbildung 3.26 aufgespannt wird.

Auch hier erhalten wir die Generatormatrix, indem wir die Basisvektoren zeilenweise aufschreiben. Als Ergebnis entsteht die Matrix, die uns bereits auf Seite 148 begegnet ist:

$$G := \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \quad (3.11)$$

$$\left\{ \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \right\}$$

**Abb. 3.26:** Die Elemente des aufgespannten Untervektorraums sind die Codewörter des Hamming-Codes.

Diese Matrix erzeugt den **[7,4]-Hamming-Code**, dessen Codewörter in der linken Spalte in Abbildung 3.27 aufgelistet sind. Hamming-Codes spielen eine große Rolle in der Kanalcodierung und werden ausführlich in Abschnitt 6.4.2 behandelt. Dort wird auch klar werden, dass der angegebene Code unter den vielen Blockcodes, die sich mit ähnlichen Generatormatrizen erzeugen lassen, eine Sonderstellung einnimmt.

Aus Abschnitt 2.5.2 wissen wir, dass zu jedem Untervektorraum  $U \subseteq V$  ein Untervektorraum  $U^\perp \subseteq V$  existiert, dessen Vektoren orthogonal auf den Vektoren von  $U$  stehen. Die Elemente dieses Orthogonalraums bilden ebenfalls einen linearen Code, der in der Codierungstheorie einen eigenständigen Namen trägt:

Hamming-Code		Simplex-Code		Erweiterter Hamming-Code	
$u$	$c(u)$	$u$	$c(u)$	$u$	$c(u)$
0000	0000000	000	0000000	0000	00000000
0001	1101001	001	1101001	0001	11010010
0010	0101010	010	1011010	0010	01010101
0011	1000011	011	0110011	0011	10000111
0100	1001100	100	0111100	0100	10011001
0101	0100101	101	1010101	0101	01001011
0110	1100110	110	1100110	0110	11001100
0111	0001111	111	0001111	0111	00011110
1000	1110000			1000	11100001
1001	0011001			1001	00110011
1010	1011010			1010	10110100
1011	0110011			1011	01100110
1100	0111100			1100	01111000
1101	1010101			1101	10101010
1110	0010110			1110	00101101
1111	1111111			1111	11111111

Abb. 3.27: Der Hamming-Code, der Simplex-Code und der erweiterte Hamming-Code im Vergleich



### Definition 3.7 (Dualer Code)

Definiert der Untervektorraum  $U$  den linearen Code  $C$ , dann definiert der Orthogonalraum  $U^\perp$  den zu  $C$  dualen Code.

Wie sich der Orthogonalraum  $U^\perp$  erzeugen lässt, wissen wir bereits. Seine Generatormatrix entsteht, indem zunächst die Generatormatrix von  $U$  in die systematische Form gebracht wird und anschließend die Spalten rechts der Einheitsmatrix als Zeilenvektoren aufgeschrieben werden. Wird diese Teilmatrix jetzt noch von rechts um die Einheitsmatrix ergänzt, so entsteht daraus die Generatormatrix für  $U^\perp$ .

Generatormatrix des erweiterten Hamming-Codes

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Systematische Form

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Übergang in den Orthogonalraum

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Systematische Form

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Generatormatrix des Orthogonalraums in systematischer Form

**Abb. 3.28:** Der erweiterte Hamming-Code ist mit seinem dualen Code identisch; er ist *selbstdual*.

Für die Matrix (3.11) haben wir diese Transformation bereits auf Seite 151 durchgeführt und am Ende das folgende Ergebnis erhalten:

$$G^\perp = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \quad (3.12)$$

Diese Matrix erzeugt den sogenannten *Simplex-Code*, der in Abbildung 3.27 in der mittleren Spalte zu sehen ist. In Abschnitt 6.4.7 werden wir auf diesen Code zurückkommen und dort aufdecken, warum er diesen Namen verdient.

### Erweiterter Hamming-Code

Ein genauso interessanter Code wird durch diese Matrix erzeugt:

$$G' := \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Von der weiter oben eingeführten *Hamming-Matrix* (3.11) unterscheidet sie sich lediglich durch eine zusätzliche Spalte. Das bedeutet, dass  $G'$  einen Code erzeugt, der genauso viele Elemente wie der [7,4]-Hamming-Code enthält, aber um 1 Bit verlängerte Codewörter aufweist. Wie die Codewörter dieses *erweiterten Hamming-Codes* im Einzelnen aussehen, zeigt die rechte Spalte in Abbildung 3.27. Ein gezielter Blick macht deutlich, dass das zusätzlich erzeugte Bit ein Paritätsbit ist. Es ist genau dann gleich 1, wenn die Anzahl der Einsen in  $u$  ungerade ist.

Als Nächstes wollen wir untersuchen, wie der duale Code des erweiterten Hamming-Codes aufgebaut ist. Zu diesem Zweck werfen wir einen Blick auf Abbildung 3.28. Dort wird die Generatormatrix des erweiterten Hamming-Codes in die systematische Form gebracht und daraus die Generatormatrix des Orthogonalraums abgeleitet. Wird diese, wie in Abbildung 3.28 geschehen, selbst wieder in die systematische Form gebracht, so tritt eine erstaunliche Eigenschaft zu Tage. Die Generatormatrix in systematischer Form ist mit der Generatormatrix des Orthogonalraums in systematischer Form identisch, d. h., beide Untervektorräume sind gleich. Lineare Codes mit dieser Eigenschaft haben einen besonderen Namen: Sie heißen *selbstdual*.



### Definition 3.8 (Selbstdualer Code)

Ein linearer Code  $C$  mit  $C = C^\perp$  heißt *selbstdual*.

Als nächstes Beispiel betrachten wir den Code, der durch die folgende Generatormatrix erzeugt wird:

$$G'': \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (3.13)$$

Abbildung 3.29 zeigt, dass diese Matrix einen Code generiert, der dem erweiterten Hamming-Code sehr ähnlich ist. Beide Codetabellen unterscheiden sich nur dadurch, dass die Codewörter in anderen Zeilen auftauchen und das Paritätsbit einmal an der ersten und einmal an der letzten Stelle auftritt. Warum dies so ist, lässt sich leicht erklären: Die Matrix  $G''$  ist so aufgebaut, dass sie sich aus  $G'$  durch elementare Zeilen- und Spaltenoperationen konstruieren lässt (Abbildung 3.29 Mitte). Die Vertauschung bzw. die Addition zweier Zeilen sorgt dafür, dass die Reihenfolge der Codewörter innerhalb der Tabelle variiert und die Vertauschung von Spalten verändert die Reihenfolge der Bits innerhalb der Codewörter.

Codes, die sich auf diese Weise ineinander überführen lassen, heißen *äquivalent*.



### Definition 3.9 (Code-Äquivalenz)

Zwei lineare Codes, die durch die Matrizen  $G_1$  bzw.  $G_2$  erzeugt werden, heißen *äquivalent*, wenn  $G_1$  durch

- die Vertauschung von Zeilen und Spalten und
- die Addition von Zeilen

nach  $G_2$  überführt werden kann.

Natürlich existieren auch zu dem Hamming-Code in Abbildung 3.27 eine Reihe äquivalenter Codes. Zwei prominente wollen wir an dieser Stelle herleiten. Für die Konstruktion der ersten Generatormatrix müssen wir lediglich die Zeilen und Spalten geeignet umsortieren. Im ersten

Erweiterter Hamming-Code		Äquivalenter Code	
$u$	$c(u)$	$u$	$c(u)$
0000	00000000	0000	00000000
0001	11010010	0001	11110000
0010	01010101	0010	01100110
0011	10000111	0011	10010110
0100	10011001	0100	11001100
0101	01001011	0101	00111100
0110	11001100	0110	10101010
0111	00011110	0111	01011010
1000	11100001	1000	01101001
1001	00110011	1001	10011001
1010	10110100	1010	00001111
1011	01100110	1011	11111111
1100	01111000	1100	10100101
1101	10101010	1101	01010101
1110	00101101	1110	11000011
1111	11111111	1111	00110011

$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$ 
  
  
 $\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$ 
  
  
 $\begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$ 
  
  
 $\begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$

**Abb. 3.29:** Der erweiterte Hamming-Code und der rechts abgedruckte Code sind äquivalent. Die Generatormatrizen lassen sich durch elementare Zeilen- und Spaltenumformungen ineinander überführen.

Schritt bringen wir die Zeilen in die neue Reihenfolge

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix} \quad (3.14)$$

und bringen anschließend jene Spalten nach vorne, die genau eine 1 enthalten. Als Ergebnis erhalten wir die Matrix, die in Abbildung 3.30 in der mittleren Spalte zu sehen ist. Ein Blick auf die Codewörter offenbart eine interessante Eigenschaft: Rotieren wir die Bits innerhalb eines Codeworts um eine beliebige Anzahl an Positionen nach links oder rechts,

Hamming-Code		Zyklischer Hamming-Code (1)		Zyklischer Hamming-Code (2)	
$u$	$c(u)$	$u$	$c(u)$	$u$	$c(u)$
0000	0000000	0000	0000000	0000	0000000
0001	1101001	0001	0001101	0001	0001011
0010	0101010	0010	0010111	0010	0010110
0011	1000011	0011	0011010	0011	0011101
0100	1001100	0100	0100011	0100	0100111
0101	0100101	0101	0101110	0101	0101100
0110	1100110	0110	0110100	0110	0110001
0111	0001111	0111	0111001	0111	0111010
1000	1110000	1000	1000110	1000	1000101
1001	0011001	1001	1001011	1001	1001110
1010	1011010	1010	1010001	1010	1010011
1011	0110011	1011	1011100	1011	1011000
1100	0111100	1100	1100101	1100	1100010
1101	1010101	1101	1101000	1101	1101001
1110	0010110	1110	1110010	1110	1110100
1111	1111111	1111	1111111	1111	1111111

Abb. 3.30: Der mittlere und der rechte Code sind systematisch und zyklisch. Sie sind zu dem [7,4]-Hamming-Code äquivalent.

so entsteht erneut ein Codewort. Das bedeutet, dass der Hamming-Code zu einem zyklischen Code äquivalent ist, der, in naheliegender Weise, als *zyklischer Hamming-Code* bezeichnet wird.

Die rechte Spalte in Abbildung 3.30 zeigt, dass der zyklische Hamming-Code nicht eindeutig definiert ist. Indem wir die Codewörter eines zyklischen Codes spiegeln, erhalten wir erneut einen zyklischen Code. Entsprechend einfach lässt sich auch die dazugehörige Generatormatrix konstruieren. Sie entsteht, indem die Spalten zunächst in umgekehrter Reihenfolge aufgeschrieben werden und die Matrix anschließend durch die Anwendung elementarer Zeilenumformungen in die normalisierte Form gebracht wird.

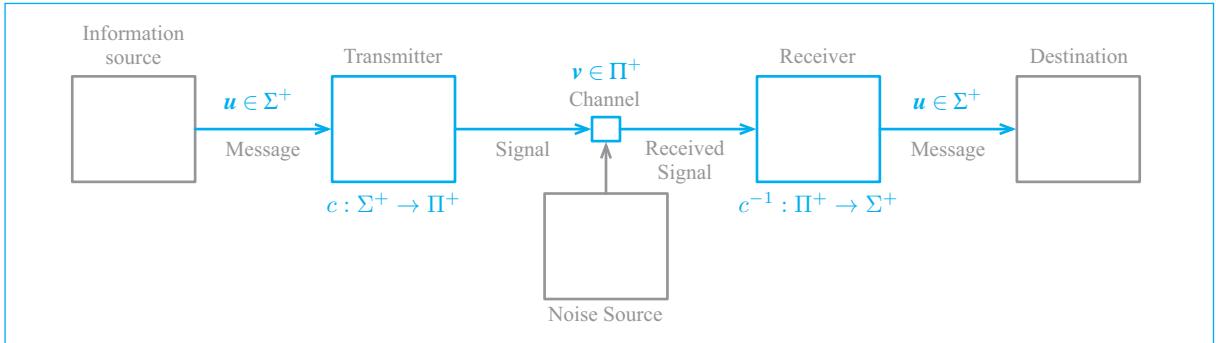


Abb. 3.31: Unser Begriffsgerüst im Lichte des Shannon'schen Kommunikationsmodells [81]

### 3.5 Der Übertragungskanal

*„Teletype and telegraphy are two simple examples of a discrete channel for transmitting information. Generally, a discrete channel will mean a system whereby a sequence of choices from a finite set of elementary symbols S<sub>1</sub>...S<sub>n</sub> can be transmitted from one point to another.“*

Claude Shannon [82]

Wir kommen in diesem Abschnitt auf das Shannon'sche Übertragungsmodell zurück. Die kommentierte Fassung in Abbildung 3.31 zeigt, dass sich das Modell problemlos mit den bisher eingeführten Begriffen in Einklang bringen lässt. Zunächst wird dem Sender (*transmitter*) eine Nachricht  $u \in \Sigma^+$ , d. h. eine Sequenz von Zeichen aus dem Quellenalphabet  $\Sigma$ , übergeben. Anschließend wird die Nachricht in ein Signal, d. h. eine Reihe von Zeichen aus dem Codealphabet, übersetzt und in den Übertragungskanal (*communication channel*) eingespeist.

Machen wir zu einem bestimmten Zeitpunkt eine Momentaufnahme, so hat der Sender bis dahin eine endliche Anzahl von Quellsymbolen entgegengenommen und daraus eine endliche Anzahl von Codesymbolen generiert. Mit anderen Worten: Die Eingabe und die Ausgabe sind zwei Wörter  $u \in \Sigma^+$  und  $v \in \Pi^+$  mit der Eigenschaft

$$c(u) = v$$

$c$  ist die vorgenommene Codierung. Der *Empfänger (receiver)* nimmt das Signal entgegen und wandelt es in eine Nachricht zurück.

### 3.5.1 Kanalkapazität

In diesem Abschnitt beschäftigen wir uns mit der Frage, wie sich die Kapazität des Übertragungskanals quantitativ erfassen lässt, und folgen dabei weitgehend der Argumentationslinie aus Shannons historischer Arbeit [81]. Um die Betrachtungen einfach zu halten, gehen wir von einem Sender aus, der mit einem binären Eingabestrom gespeist wird. Wir nehmen an, dass nach  $T$  Zeiteinheiten  $n$  Bits eingespeist wurden. Mit  $n$  Bits lassen sich  $2^n$  verschiedene Bitsequenzen konstruieren, sodass es  $2^n$  verschiedene Möglichkeiten gibt, den Sender zu versorgen.

Ist der Sender in der Lage, in einem Zeitraum von  $T$  Zeiteinheiten  $2^n$  verschiedene Signale auf dem Übertragungskanal zu erzeugen, so kann er in diesem Zeitraum die  $n$  eingespeisten Bits übertragen. Mit der Vereinbarung

$$N(T) := \text{Anzahl der erlaubten Signale der Länge } T$$

können wir den Sachverhalt auch so ausdrücken: Ist

$$N(T) = 2^n,$$

so kann der Übertragungskanal in  $T$  Zeiteinheiten  $n$  Bits transportieren. Genauso gut können wir sagen, dass auf dem Kanal innerhalb von  $T$  Zeiteinheiten durchschnittlich

$$\frac{\log_2 N(T)}{T} \quad (3.15)$$

Bits pro Zeiteinheit übertragen werden.

Wenn wir (3.15) für immer größere Werte von  $T$  betrachten, wird sich der Quotient für alle Übertragungskanäle, die in der Praxis eine Rolle spielen, stabilisieren. Dies versetzt uns in die Lage, die Kanalkapazität folgendermaßen zu definieren:

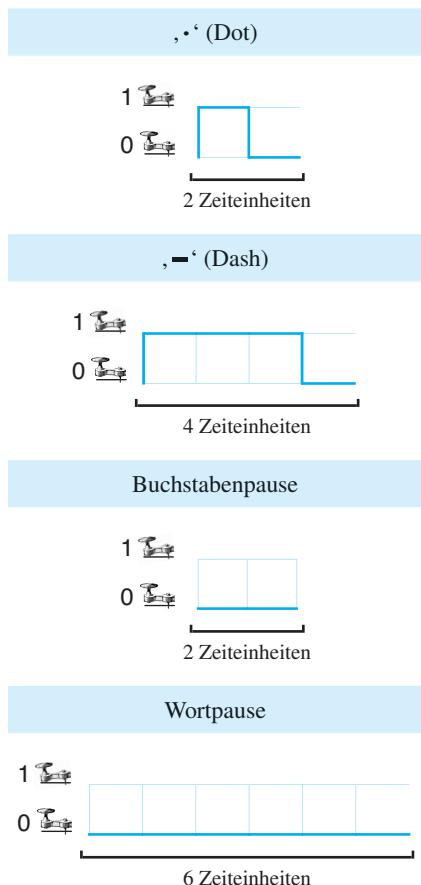


#### Definition 3.10 (Kanalkapazität nach Shannon, 1948)

Ein diskreter Übertragungskanal besitzt die *Kanalkapazität*

$$C := \lim_{T \rightarrow \infty} \frac{\log_2 N(T)}{T}$$

Die Kanalkapazität ist die maximale Übertragungsrate in Bits pro Zeiteinheit, die aufgrund der gegebenen physikalischen Randbedingungen über lange Zeiträume aufrechterhalten werden kann.



**Abb. 3.32:** Physikalische Repräsentation der Morse-Codesymbole auf dem Übertragungskanal

### 3.5.2 Kapazität des Morse-Kanals

Mit Shannons Begriffsdefinition sind wir in der Lage, auch komplexe Szenarien zu erfassen, wie die Übertragungen von Nachrichten im Morse-Code. Dabei müssen wir zwei wichtige Punkte beachten.

- Die Übertragung der Codesymbole dauert unterschiedlich lange (Abbildung 3.32). In der klassischen Morse-Telegrafie wird das Symbol ,·‘ übertragen, indem die Leitung für eine Zeiteinheit geschlossen und anschließend für eine Zeiteinheit geöffnet wird. Für das Symbol ,-' wird die Leitung für drei Zeiteinheiten geschlossen und danach für eine Zeiteinheit geöffnet. Besondere Regeln gelten für die Pausen zwischen Buchstaben und Wörtern. Sie werden dargestellt, indem die Leitung z. B. für 2 bzw. 6 Zeiteinheiten geöffnet bleibt.
- Nicht alle Sequenzen, die mit den Symbolen ,·‘ und ,-' gebildet werden können, sind wohlgeformte Signale. Beispielsweise ist es weder erlaubt, die Leitung für mehr als drei Zeiteinheiten geschlossen zu halten, noch dürfen mehrere Pausenzeichen direkt hintereinander gesendet werden. Würden wir auf die zuletzt genannte Restriktion verzichten, so wäre eine zweimal hintereinander gesendete Buchstabenpause nicht von einer Wortpause zu unterscheiden.

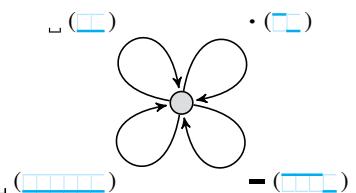
Halten wir uns streng an Definition 3.10, so können wir die Kapazität des Morse-Übertragungskanals annähern, indem wir  $N(T)$ , die Anzahl der zulässigen Signale, für immer größer werdende  $T$  abzählen und den Logarithmus dieses Werts durch  $T$  dividieren. Da die Anzahl der Signale, die wir dabei in Betracht ziehen müssten, exponentiell mit  $T$  wächst, würden wir jedoch schnell an eine praktische Grenze stoßen.

Aus diesem Grund wollen wir versuchen, die Kanalkapazität auf einem systematischeren Weg zu berechnen. Ignorieren wir für den Moment die Restriktion, dass die Pausenzeichen nicht direkt hintereinander gesendet werden dürfen, so können wir die Situation mit einem Zustandsübergangsdiagramm darstellen, wie es in Abbildung 3.33 zu sehen ist. Das Diagramm besteht aus einem einzigen Zustand und vier Kanten für die beiden Zeichen ,·‘ und ,-' sowie die beiden Pausen. Die Kantenmarkierungen zeigen die Signalpegel, die auf dem Übertragungskanal erzeugt werden; im Falle von ,·‘ ist die Leitung beispielsweise für eine Zeiteinheit geschlossen und dann für eine Zeiteinheit offen () und im Falle von ,-' für drei Zeiteinheiten geschlossen, bevor sie wieder geöffnet wird ().

Die Übertragung von  $\cdot\text{--}\cdot$  erzeugt z. B. den folgenden Signalverlauf:

$$\cdot\text{--}\cdot \mapsto \square\ \square\ \square\ \square\ \square\ \square$$

Betrachten wir den Signalverlauf auf dem Übertragungskanal, nachdem eine größere Anzahl der Symbole,  $\cdot$ ,  $\text{--}$  und  $\cdot$  übertragen wurde, so finden wir dort eine Sequenz vor, für die einer von drei Fällen zutrifft:



**Abb. 3.33:** Modellierung des Morse-Übertragungskanals mithilfe eines Zustandsübergangsdiagramms

- Zuletzt wurde das Symbol  $\cdot$  übertragen.

In diesem Fall endet die Sequenz mit  $\square$  und es gibt  $N(T - 2)$  solche Sequenzen der Länge  $T$ .

- Zuletzt wurde das Symbol  $\text{--}$  übertragen.

In diesem Fall endet die Sequenz mit  $\square\square$  und es gibt  $N(T - 4)$  solche Sequenzen der Länge  $T$ .

- Zuletzt wurde eine Buchstaben- oder eine Wortpause übertragen.

In diesem Fall endet die Sequenz mit  $\square$  und es gibt  $N(T - 2)$  solche Sequenzen der Länge  $T$ .

Damit lässt sich  $N(T)$  über die folgende *Rekurrenzgleichung* erfassen:

$$N(T) = N(T - 2) + N(T - 4) + N(T - 2)$$

Diese Gleichung können wir zu

$$N(T) = N(T - T_1) + N(T - T_2) + \dots + N(T - T_n) \quad (3.16)$$

verallgemeinern und für deren Lösung auf ein bekanntes Ergebnis aus der Rekursionstheorie zurückgreifen [81]. Dieses besagt, dass sich die Lösung von (3.16) asymptotisch dem Wert

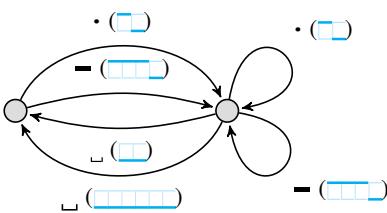
$$A \cdot X_0^T \quad (3.17)$$

nähert, wobei  $A$  eine Konstante ist und  $X_0$  die größte reellwertige Lösung der *charakteristischen Gleichung*

$$X^{-T_1} + X^{-T_2} + \dots + X^{-T_n} = 1.$$

Setzen wir (3.17) in die Formel ein, die in Definition 3.10 für die Berechnung der Kanalkapazität angegeben ist, so ergibt sich

$$\begin{aligned} C &= \lim_{T \rightarrow \infty} \frac{\log_2 N(T)}{T} = \lim_{T \rightarrow \infty} \frac{\log_2 A \cdot X_0^T}{T} = \lim_{T \rightarrow \infty} \frac{\log_2 A + T \log_2 X_0}{T} \\ &= \lim_{T \rightarrow \infty} \frac{\log_2 A}{T} + \lim_{T \rightarrow \infty} \frac{T \log_2 X_0}{T} = \log_2 X_0 \end{aligned}$$



**Abb. 3.34:** Das gezeigte Zustandsübergangsdiagramm berücksichtigt, dass Buchstaben- und Wortpausen nicht direkt hintereinander gesendet werden dürfen.

Damit ist der Weg frei für die Berechnung der Kapazität des Morse-Kanals. Die größte reellwertige Lösung von

$$X^{-2} + X^{-4} + X^{-2} = 1$$

ist die Zahl  $X_0 \approx 1,5538$ , sodass wir das folgende Ergebnis erhalten:

$$C = \log_2 X_0 \approx \log_2 1,5538 \approx 0,6358 \frac{\text{Bits}}{\text{Zeiteinheit}}$$

Ganz fertig sind wir damit noch nicht, denn bisher haben wir bei der Berechnung missachtet, dass Pausensymbole nicht direkt hintereinander gesendet werden dürfen. Um dieser Forderung ebenfalls Rechnung zu tragen, gehen wir genauso vor wie eben und modellieren die möglichen Signalverläufe mithilfe eines Zustandsdiagramms. Das Ergebnis ist in Abbildung 3.34 zu sehen.

Im nächsten Schritt stoßen wir auf ein Problem. Da das neue Übergangsdiagramm zwei Zustände besitzt, können wir es nicht mehr ohne weiteres in eine Rekurrenzgleichung übersetzen. Im Falle des Morse-Übertragungskanals haben wir jedoch Glück, da sich das Übergangsdiagramm sehr einfach in ein äquivalentes Diagramm übersetzen lässt, das mit nur einem Zustand auskommt. Abbildung 3.35 zeigt, wie dies gelingt.

Aus dem reduzierten Diagramm können wir ablesen, dass zur Berechnung der Kanalkapazität die größte reellwertige Lösung der Gleichung

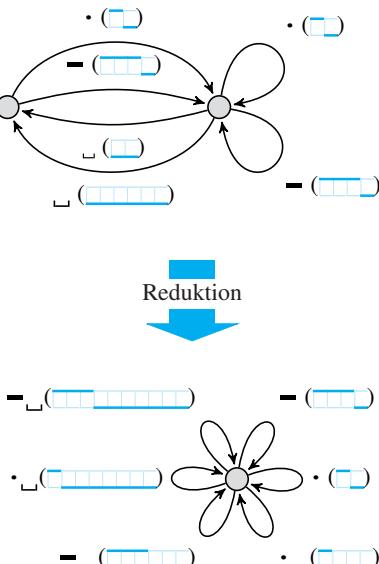
$$X^{-2} + X^{-4} + X^{-4} + X^{-6} + X^{-8} + X^{-10} \equiv 1$$

benötigt wird. Mit  $X_0 \approx 1,49305$  erhalten wir einen Wert, der geringfügig kleiner ist als der weiter oben berechnete, und dies führt dazu, dass sich die Kanalkapazität geringfügig verringert. Es ist

$$C = \log_2 X_0 \approx \log_2 1,49305 \approx 0,57826 \frac{\text{Bits}}{\text{Zeiteinheit}} \quad (3.18)$$

Damit haben wir ein wichtiges Zwischenergebnis erreicht: Der Morse-Übertragungskanal überträgt in einer Zeiteinheit bis zu 0,57826 Bits.

Um die Übertragungsrate in Bits pro Sekunde auszurechnen, ist es von hier nur ein kleiner Schritt; wir müssen dazu lediglich herausbekommen, wie lange eine Zeiteinheit bei der Morse-Übertragung normalerweise dauert. Die Geschwindigkeit eines Telegrafisten wird gewöhnlich in der Einheit WpM (*Wörter pro Minute*) gemessen und dabei PARIS als Referenzwort verwendet. Im Morse-Code wird PARIS durch die folgende Sequenz von Punkten, Strichen und Pausen dargestellt:



**Abb. 3.35:** Reduktion auf ein Zustandsübergangsdiagramm mit einem einzigen Zustand

Gesprochen wird diese Sequenz so:

„DiDaDaDit DiDah DiDaDit DiDit DiDiDit“

Auf dem Übertragungskanal wird sie folgendermaßen dargestellt:



Folglich besteht das Referenzwort aus 50 Zeiteinheiten; es ist 50-mal so lange wie der hörbare Anteil eines einzelnes „Di“ oder „Dit“. Man sagt deshalb auch, das Wort PARIS sei 50 Dits lang.

Tabelle 3.6 zeigt für verschiedene WpM-Werte, wie viele Dits pro Minute gesendet werden und wie lange ein einzelnes Dit dann jeweils ist. Die Anzahl und die Zeitspanne verhalten sich umgekehrt proportional zueinander. Eine Verdopplung der gesendeten Dits führt zu einer Halbierung der Zeit, die für ein einzelnes Dit zur Verfügung steht.

Telegrafisten unterscheiden sich stark in ihrer Geschwindigkeit. Während Amateure mit weniger als 5 WpM übertragen, liegen geübte Telegrafisten im Bereich von 15 WpM. Ab ca. 25 WpM beginnt die Morse-Schnelltelegrafie und Profi-Telegrafisten kommen auf eine Geschwindigkeit von ca. 40 WpM. Der Weltrekord liegt bei 75,2 WpM. Er wurde 1939 von dem Amerikaner Theodore R. McElroy aufgestellt und hat bis heute Bestand.

Bei einer Geschwindigkeit von 75,2 WpM erstreckt sich die Dauer eines Dits über 15,96 Millisekunden. Legen wir diese Zahl als Zeiteinheit für den Morse-Kanal zugrunde, so können wir die Kanalkapazität problemlos in Bits pro Sekunde ausdrücken: Aus (3.18) folgt

$$C = 0,57826 \frac{\text{Bits}}{\frac{15,96}{1000} \text{sec}} = 36,232 \frac{\text{Bits}}{\text{sec}} \quad (3.19)$$

Jetzt haben wir das Ergebnis schwarz auf weiß vor uns: Der Morse-Übertragungskanal hat eine Kapazität von ca. 36 Bits pro Sekunde.

WpM	Dits Minute	Länge eines Dits	
1	50	1200	msec
5	250	240	msec
10	500	120	msec
15	750	80	msec
20	1000	60	msec
25	1250	48	msec
30	1500	40	msec
35	1750	34,29	msec
40	2000	30	msec
45	2250	26,67	msec
50	2500	24	msec
55	2750	21,82	msec
60	3000	20	msec
65	3250	18,46	msec
70	3500	17,14	msec
75	3750	16	msec
75,2	3760	15,96	msec

1) Amateurbereich

2) Schnelltelegrafie

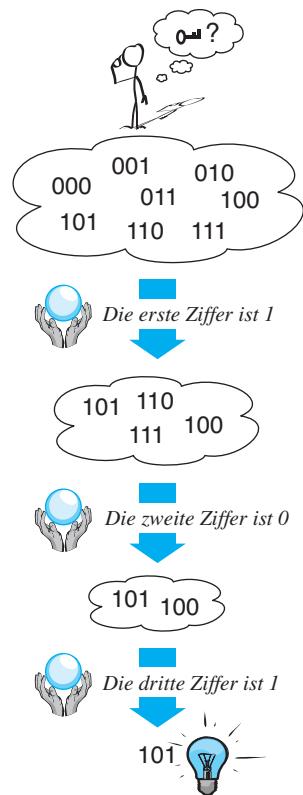
3) Profi-Telegrafie

4) Weltrekord

**Tab. 3.6:** Typische Morse-Geschwindigkeiten im Vergleich

## 3.6 Der Informationsbegriff

Im letzten Abschnitt haben wir unsere Aufmerksamkeit auf die Kanalkapazität gerichtet und dabei mehrfach von übertragenen Bits, aber noch nicht von übertragener Information gesprochen. Momentan haben wir nur eine intuitive Vorstellung davon, was Information tatsächlich ist,



**Abb. 3.36:** Information reduziert Ungewissheit. Jedes Bit an Information lässt die Anzahl der in Frage kommenden Sequenzen um die Hälfte sinken.

und eine formale Definition steht bis jetzt aus. In diesem Abschnitt werden wir das Versäumte nacharbeiten und dem Begriff der Information ein konturenreiches Gesicht verleihen.

Die angestellten Überlegungen folgen einem Leitgedanken, der selten so klar ausgedrückt wurde wie in dem folgenden Zitat des amerikanischen Psychologen Fred Attneave:

„Eine Aussage oder Beobachtung ist dann informativ, wenn sie uns etwas mitteilt, das uns nicht vorher schon bekannt war. Nur über solche Dinge können wir jedenfalls Information gewinnen, über die bei uns ein gewisses Maß an Nichtwissen oder Ungewissheit besteht. In der Tat lässt sich Information als dasjenige erklären, das Ungewissheit beseitigt oder reduziert.“

Fred Attneave [5]

Der letzte Satz bringt den Leitgedanken auf den Punkt: *Information ist das, was Ungewissheit beseitigt oder reduziert.*

Wir wollen versuchen, Attneaves Worte mit Leben zu füllen, und konfrontieren uns mit der fiktiven Aufgabe, in ein Computersystem einzubrechen zu müssen. Wir nehmen an, das System sei mit einem Binärschlüssel gesichert, von dem wir lediglich wissen, dass er 3 Bit lang ist. In unserem fiktiven Szenario dürfen wir zusätzlich auf die Hilfe eines Orakels vertrauen, das sich bereit erklärt, uns alle drei Binärziffern auf Nachfrage nacheinander zu verraten (Abbildung 3.36).

Zu Beginn der Befragung wissen wir noch nichts über den Binärschlüssel; unsere Unwissenheit ist maximal, oder, was dasselbe ist: Wir sind im Besitz von 0 Bits an Information. Befragen wir das Orakel nach der ersten Binärziffer, so wird es uns entweder mit „0“ oder „1“ antworten. Gehen wir davon aus, dass der Binärschlüssel zufällig gewählt wurde, so sind beide Antworten gleichwahrscheinlich. Die Antwort des Orakels entspricht dann 1 Bit an Information.

Die Antwort des Orakels reduziert unsere Unwissenheit, denn wir wissen nun, dass von den ursprünglich 8 möglichen Sequenzen nur noch 4 in Frage kommen. Befragen wir das Orakel erneut, so erhalten wir ein weiteres Bit an Information, und unsere Unwissenheit verringert sich abermals. Sobald wir das zweite Bit kennen, wird die Anzahl der in Frage kommenden Sequenzen auf nur noch 2 reduziert. Die dritte Frage an das Orakel bringt uns ein weiteres Bit an Information und beendet das Spiel: Wir kennen nun die Sequenz, nach der wir gesucht haben.

Betrachten wir das Spiel abstrakt, so tritt ein elementarer Zusammenhang zwischen dem Informationsgewinn und dem Unsicherheitsverlust zum Vorschein, den wir plakativ so formulieren können:

- 8 Möglichkeiten auf 4 Möglichkeiten einzugrenzen  
entspricht einer Information von 1 Bit.
  - 8 Möglichkeiten auf 2 Möglichkeiten einzugrenzen  
entspricht einer Information von 2 Bit.
  - 8 Möglichkeiten auf 1 Möglichkeit einzugrenzen  
entspricht einer Information von 3 Bit.
- Diesen Zusammenhang können wir mühelos verallgemeinern. Sind  $n$  und  $m$  zwei natürliche Zahlen mit  $n \geq m$ , so gilt:
- $2^n$  Möglichkeiten auf  $2^m$  Möglichkeiten einzugrenzen  
entspricht einer Information von  $n - m$  Bit.

Bei der mathematischen Erfassung des Informationsbegriffs gibt es keinen Grund, unsere Betrachtung auf Zweierpotenzen einzuschränken. Folgerichtig können wir das Gesagte noch weiter verallgemeinern:

- $n$  Möglichkeiten auf  $m$  Möglichkeiten einzugrenzen  
entspricht einer Information von  $\log_2 n - \log_2 m$  Bit.

Ersetzen wir  $\log_2 n - \log_2 m$  durch den äquivalenten Ausdruck  $\log_2 \frac{n}{m}$ , so erreichen wir ein wichtiges Zwischenergebnis. Es ist uns gelungen, den Informationsbegriff quantitativ zu erfassen:

  $n$  Möglichkeiten auf  $m$  Möglichkeiten einzugrenzen  
entspricht einer Information von

$$\log_2 \frac{n}{m} \text{ Bit}$$

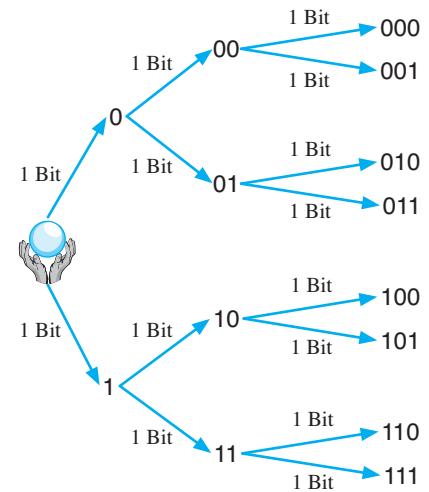


Abb. 3.37: Die möglichen Ausgänge der Orakelbefragung

Abbildung 3.37 visualisiert die oben durchgeführte Orakelbefragung auf grafische Weise. Das Spiel ist in Form eines Baums dargestellt, wobei jeder Pfad von der Wurzel bis zu einem Blatt einen möglichen Verlauf beschreibt. Jede Kante ist mit der Information markiert, die wir beim Eintreten des entsprechenden Ereignisses gewinnen.

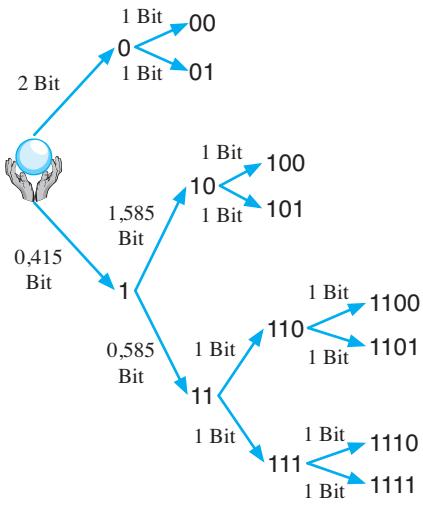


Abb. 3.38: Variation der Orakelbefragung

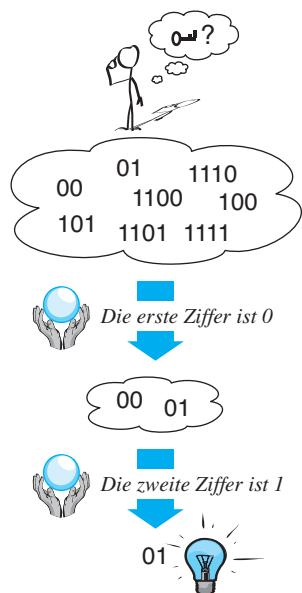


Abb. 3.39: Die erste Antwort reduziert die Anzahl der Möglichkeiten von 8 auf 2. Dies entspricht 2 Bit an Information.

Als Nächstes wollen wir untersuchen, wie belastbar unser erworbenes Begriffsgerüst ist, und wandeln unsere ursprünglich zu lösende Aufgabe geringfügig ab. Anstatt von einem Binärschlüssel der Länge 3 auszugehen, nehmen wir an, der gesuchte Schlüssel stamme aus der Menge

$$\{00, 01, 100, 101, 1100, 1101, 1110, 1111\}.$$

Genau wie im ersten Fall können wir die möglichen Ausgänge der Orakelbefragung in Form eines Baums visualisieren (Abbildung 3.38).

Ein Blick auf die Kantenmarkierungen bringt ein interessantes Phänomen zum Vorschein: Die Information, die wir durch die Befragung des Orakels gewinnen, variiert jetzt in Abhängigkeit der gelieferten Antwort. Warum dies so ist, lässt sich an der Baumstruktur erkennen. Beantwortet das Orakel unsere Frage nach dem ersten Bit mit „0“, so haben wir die 8 ursprünglichen Möglichkeiten schlagartig auf 2 eingegrenzt (Abbildung 3.39). Dies entspricht einem Informationsgehalt von

$$\log_2 \frac{8}{2} \text{ Bit} = \log_2 4 \text{ Bit} = 2 \text{ Bit}.$$

Antwortet das Orakel stattdessen mit „1“, so verbleiben 6 Möglichkeiten. In diesem Fall haben wir also lediglich

$$\log_2 \frac{8}{6} \text{ Bit} = \log_2 \frac{4}{3} \text{ Bit} \approx 0,415 \text{ Bit}$$

an Information gewonnen.

Weiter oben haben wir erklärt, dass „8 Möglichkeiten auf 1 Möglichkeit einzugrenzen“ einer Information von 3 Bit entspricht. Wenn der Informationsbegriff, wie wir ihn definiert haben, einen Sinn ergeben soll, so müsste die Summe der Einzelinformationen, die unsere Orakelbefragungen liefern, immer gleich 3 sein. In unserem ersten Beispiel war dies tatsächlich der Fall, und die folgende Rechnung zeigt, dass dieser Zusammenhang auch in unserem zweiten Beispiel gilt:

Sequenz 00 :	$I = (2 + 1) \text{ Bit} = 3 \text{ Bit}$
Sequenz 01 :	$I = (2 + 1) \text{ Bit} = 3 \text{ Bit}$
Sequenz 100 :	$I = (0,415 + 1,585 + 1) \text{ Bit} = 3 \text{ Bit}$
Sequenz 101 :	$I = (0,415 + 1,585 + 1) \text{ Bit} = 3 \text{ Bit}$
Sequenz 1100 :	$I = (0,415 + 0,585 + 1 + 1) \text{ Bit} = 3 \text{ Bit}$
Sequenz 1101 :	$I = (0,415 + 0,585 + 1 + 1) \text{ Bit} = 3 \text{ Bit}$
Sequenz 1110 :	$I = (0,415 + 0,585 + 1 + 1) \text{ Bit} = 3 \text{ Bit}$
Sequenz 1111 :	$I = (0,415 + 0,585 + 1 + 1) \text{ Bit} = 3 \text{ Bit}$

Wie von Geisterhand summieren sich die gebrochenen Informationsgehalte zu dem immer gleichen Ergebnis. Die Art und Weise, wie die Information quantitativ erfasst wird, scheint also tatsächlich zu einer plausiblen Maßzahl zu führen, und diesen Eindruck wollen wir an einem weiteren Beispiel bestätigen. Wir betrachten eine Variante des bekannten *Hüttchenspiels (shell game)*, bei dem eine Perle so schnell zwischen vier Hüttchen verschoben wird, dass wir am Ende nicht wissen, wo sie sich befindet. Unsere Aufgabe ist es, das Hüttchen mit der Perle zu finden.



Genau wie in den ersten beiden Beispielen starten wir in vollkommener Ungewissheit: Wir haben keinerlei Anhaltspunkte darüber, unter welchem Hüttchen sich die Perle verbirgt. Unter dieser Voraussetzung haben wir keine andere Wahl, als uns im ersten Schritt zufällig für eine der vier Möglichkeiten zu entscheiden. Raten wir richtig, so ist das Spiel beendet. Raten wir falsch, so müssen wir unter den drei verbleibenden Hüttchen ein weiteres aussuchen (Abbildung 3.40 oben). Tragen wir die möglichen Spielverläufe grafisch auf, so entsteht eine Baumstruktur, wie sie in Abbildung 3.40 (unten) dargestellt ist. Die Kantenmarkierungen zeigen auch hier, wie viel Information, gemessen in Bits, wir beim Eintreten eines bestimmten Ereignisses gewinnen.

Wie diese Werte zustande kommen, lässt sich leicht nachvollziehen. Raten wir gleich beim ersten Versuch richtig, so haben wir vier Möglichkeiten auf eine reduziert. Dies entspricht einem Informationsgewinn von

$$\log_2 \frac{4}{1} \text{ Bit} = 2 \text{ Bit}$$

Raten wir falsch, so bleiben drei mögliche Hüttchen übrig, d. h., wir haben die ursprünglichen vier Möglichkeiten auf drei reduziert, was einem Informationsgewinn von

$$\log_2 \frac{4}{3} \text{ Bit} \approx 0,415 \text{ Bit}$$

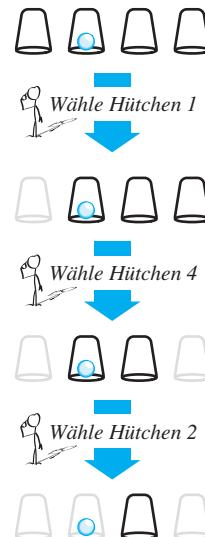
entspricht. Raten wir beim zweiten Mal richtig, so haben wir die drei verbleibenden Möglichkeiten auf eine reduziert und die Information

$$\log_2 \frac{3}{1} \text{ Bit} \approx 1,585 \text{ Bit}$$

gewonnen. Raten wir erneut falsch, so verbleiben zwei Möglichkeiten, was einem Informationsgewinn von

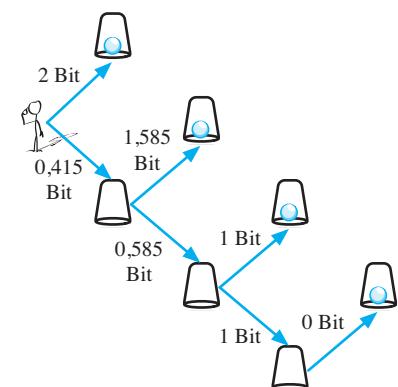
$$\log_2 \frac{3}{2} \text{ Bit} \approx 0,585 \text{ Bit}$$

#### ■ Möglicher Verlauf des Hüttchenspiels



#### ■ Es gibt vier mögliche Spielverläufe

- 1) Erfolg im 1. Rateversuch
- 2) Erfolg im 2. Rateversuch
- 3) Erfolg im 3. Rateversuch
- 4) Erfolg im 4. Rateversuch



**Abb. 3.40:** Eine Variante des Hüttchenspiels mit vier Hüttchen. Wo verbirgt sich die Perle?

bedeutet. Sie sehen, dass die Argumentation, die wir für die Bestimmung des Informationsgewinns verwenden, immer die gleiche ist und der oben skizzierten Leitidee folgt:  $n$  Möglichkeiten auf  $m$  Möglichkeiten einzuschränken entspricht einer Information von  $\log_2 \frac{n}{m}$  Bit.

Dass der vierte Rateversuch zu keinem Informationsgewinn führt, ist übrigens kein Fehler. Haben wir nämlich drei leere Hütchen aufgedeckt, so wissen wir, dass sich die Perle unter dem vierten Hütchen befinden muss; dieses Hütchen aufzudecken liefert uns deshalb keine zusätzliche Information. Natürlich gilt dies nur, solange alles mit rechten Dingen zugeht, und genau an diesem Punkt unterscheidet sich das diskutierte Hütchenspiel von den in der Praxis exerzierten.

Wir wollen ausloten, ob der Informationsbegriff, wie wir ihn definiert haben, auch im Falle des Hütchenspiels einen Sinn ergibt. Sollte dies der Fall sein, so muss auch hier die kumulierte Information, die wir am Ende des Spiels erhalten haben, unabhängig davon sein, wie das Spiel verlaufen ist. Denn egal wie oft wir raten: Am Ende haben wir 4 Möglichkeiten auf eine reduziert und damit eine Information von 2 Bit gewonnen. Die folgende Rechnung zeigt, dass unser Begriffsgerüst auch dieser Feuerprobe standhält. Für die vier möglichen Fälle gilt:

$$\text{Erfolg im 1. Rateversuch : } I = 2 \text{ Bit}$$

$$\text{Erfolg im 2. Rateversuch : } I = (0,415 + 1,585) \text{ Bit} = 2 \text{ Bit}$$

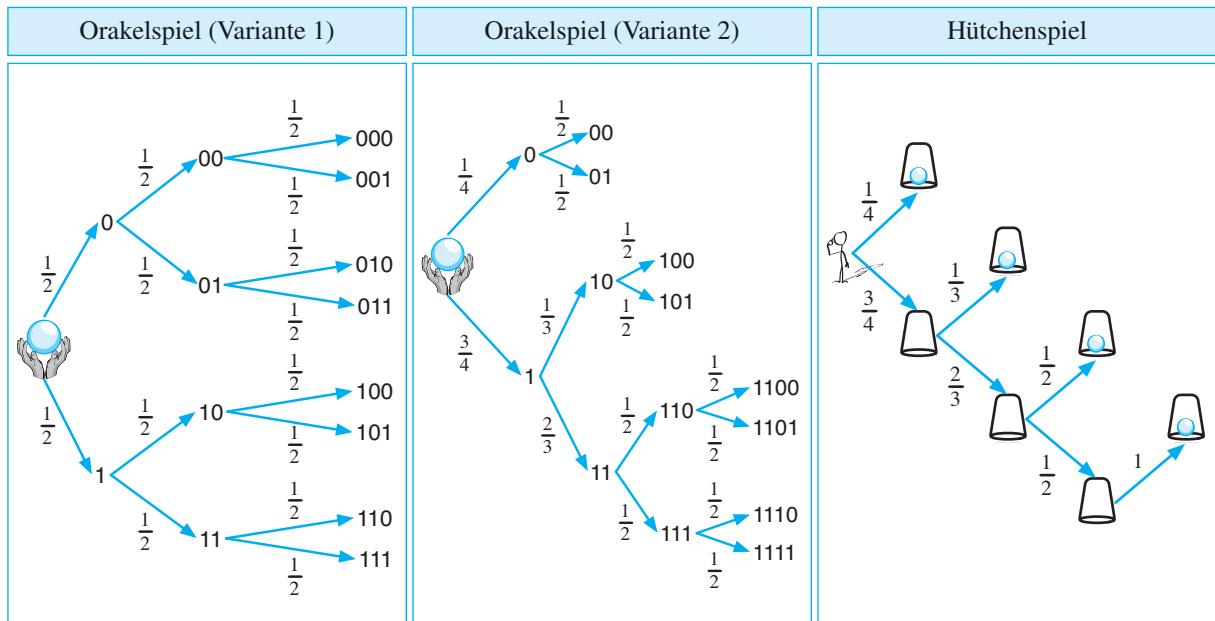
$$\text{Erfolg im 3. Rateversuch : } I = (0,415 + 0,585 + 1) \text{ Bit} = 2 \text{ Bit}$$

$$\text{Erfolg im 4. Rateversuch : } I = (0,415 + 0,585 + 1) \text{ Bit} = 2 \text{ Bit}$$

Erneut summieren sich die gebrochenen Informationsgehalte, wie von magischer Hand gerechnet, zu dem immer gleichen Ergebnis.

Tatsächlich lässt sich mit einem einfachen Argument begründen, warum die Summe immer gleich sein muss. Durchlaufen wir in unserem Entscheidungsbaum, auf dem Weg von der Wurzel zu einem Blatt,  $k$  Kanten, so können wir die erhaltenen Antworten ganz allgemein so interpretieren:

- Die 1. Antwort reduziert  $n$  Möglichkeiten auf  $m_1$  Möglichkeiten.
- Die 2. Antwort reduziert  $m_1$  Möglichkeiten auf  $m_2$  Möglichkeiten.
- Die 3. Antwort reduziert  $m_2$  Möglichkeiten auf  $m_3$  Möglichkeiten.
- ...
- Die  $k$ -te Antwort reduziert  $m_{k-1}$  Möglichkeiten auf 1 Möglichkeit.



**Abb. 3.41:** Die möglichen Spielverläufe haben verschiedene Wahrscheinlichkeiten. Ein vergleichender Blick mit den Originärbäumen macht deutlich, dass der Informationsgewinn in einem direkten Zusammenhang zu der Eintrittswahrscheinlichkeit eines Ereignisses steht.

Damit können wir den kumulierten Informationsgehalt folgendermaßen ausrechnen:

$$\begin{aligned}
 I &= \log_2 \frac{n}{m_1} + \log_2 \frac{m_1}{m_2} + \log_2 \frac{m_2}{m_3} + \dots + \log_2 \frac{m_{k-1}}{1} \\
 &= \log_2 n - \log_2 m_1 \\
 &\quad + \log_2 m_1 - \log_2 m_2 \\
 &\quad + \log_2 m_2 - \dots - \log_2 m_{k-1} \\
 &\quad + \log_2 m_{k-1} - \log_2 1 \\
 &= \log_2 n - \log_2 1 \\
 &= \log_2 n
 \end{aligned}$$

Die Rechnung macht deutlich, warum die Summe immer gleich sein muss. Die Logarithmen löschen sich paarweise aus, sodass am Ende immer der Ausdruck  $\log_2 n$  übrig bleibt.

Als Nächstes werden wir eine zentrale Beziehung zwischen dem Informationsgehalt eines Ereignisses und dessen Eintrittswahrscheinlichkeit aufdecken und auf diese Weise zu einer formalen Definition des Informationsbegriffs gelangen. Um den Zusammenhang zu verdeutli-

chen, sind in Abbildung 3.41 erneut die drei Baumstrukturen zu sehen, mit denen wir die möglichen Verläufe der Orakelbefragungen und des Hütchenspiels visualisiert haben. Im Gegensatz zu den ursprünglichen Bäumen sind die Kanten aber nicht mehr mit der Information markiert, die mit dem Eintreten eines Ereignisses verbunden ist, sondern mit der Wahrscheinlichkeit, mit der wir dieses Ereignis beobachten können.

Ein gezielter Blick auf die Kantenmarkierungen macht klar, dass wir den Informationsgewinn direkt aus der Auftrittswahrscheinlichkeit berechnen können und umgekehrt. Ist  $p$  die Wahrscheinlichkeit, dass ein Ereignis eintritt, so liefert uns dieses Ereignis exakt

$$\log_2 \frac{1}{p} = -\log_2 p \quad (3.20)$$

Bit an Information. Dieser Zusammenhang ist so elementar, dass wir ihn als formale Grundlage für die Definition des Informationsbegriffs heranziehen:



### Definition 3.11 (Informationsgehalt)

Der Informationsgehalt eines Ereignisses, das mit der Wahrscheinlichkeit  $p$  ( $p > 0$ ) eintritt, ist definiert als

$$I(p) := \log_2 \frac{1}{p} = -\log_2 p$$

Der Informationsgehalt  $I(p)$ , wie wir ihn in Definition 3.11 definiert haben, besitzt vier interessante Eigenschaften:

- $I(p)$  ist nichtnegativ

$I(p) \geq 0$

Die Wahrscheinlichkeit  $p$  ist eine Zahl aus dem halb offenen Intervall  $(0; 1]$ . Somit ist  $\frac{1}{p} \geq 1$  und  $\log_2 \frac{1}{p}$  größer oder gleich 0. Diese Eigenschaft deckt sich mit unserem intuitiven Verständnis, dass Information Unsicherheit reduziert, und der Informationsbegriff somit eine Größe beschreibt, die niemals negativ sein kann. Der Fall  $I(p) = 0$  tritt übrigens genau dann ein, wenn  $p$  gleich 1 ist. In diesem Fall liefert die Beobachtung des Ereignisses tatsächlich keine Information, da dessen Eintreten bereits feststand.

- $I(p)$  ist stetig

$p_i \rightarrow p \Rightarrow I(p_i) \rightarrow I(p)$

Der Informationsgehalt  $I(p)$  ist eine stetige Funktion. Das bedeutet, dass eine kleine Änderung der Eintrittswahrscheinlichkeit nur zu einer kleinen Änderung in der gewonnenen Information führen kann.

Dies deckt sich mit dem intuitiven Verständnis eines Informationsbegriffs, der mit einer sprunghaften Funktion nicht vereinbar wäre.

■  $I(p)$  ist monoton

$$\text{☞ } p_1 < p_2 \Rightarrow I(p_1) > I(p_2)$$

Die Monotonieeigenschaft besagt hier, dass der Informationsgehalt eines Ereignisses mit sinkender Eintrittswahrscheinlichkeit größer wird. Dies ist ein Phänomen, das wir auch in der natürlichen Sprache beobachten können. Hier tragen die am häufigsten verwendeten Wörter – meist sind dies Füllwörter – kaum zu der übermittelten Information bei, während selten genannte Begriffe oft den gesamten Inhalt eines Satzes beeinflussen.

■  $I(p)$  ist additiv

$$\text{☞ } I(p_1 p_2) = I(p_1) + I(p_2)$$

Für zwei stochastisch unabhängige Ereignisse ist die Wahrscheinlichkeit  $p$ , dass beide hintereinander eintreten, das Produkt der Einzelwahrscheinlichkeiten  $p_1$  und  $p_2$ . Das bedeutet, dass wir durch die Beobachtung des Gesamtereignisses die folgende Information gewinnen:

$$\begin{aligned} I(p) &= I(p_1 p_2) \\ &= \log_2 \frac{1}{p_1 p_2} \\ &= \log_2 \frac{1}{p_1} + \log_2 \frac{1}{p_2} \\ &= I(p_1) + I(p_2) \end{aligned}$$

Der Informationsgehalt des Gesamtereignisses ist damit genauso hoch wie die Summe der Informationsgehalte der Einzelereignisse. Auch dies deckt sich mit unserer intuitiven Erwartung an einen sinnvoll gewählten Informationsbegriff.



Die Art und Weise, wie wir in diesem Buch den Informationsbegriff eingeführt haben, ist nicht die einzige mögliche.

In vielen Lehrbüchern wird ein alternativer Weg beschritten, der die Funktion  $I(p)$  über vier notwendig zu erfüllende Eigenschaften charakterisiert. Diese Eigenschaften sind die *Nichtnegativität*, die *Stetigkeit*, die *Monotonie* und die *Additivität*, also genau jene vier, die wir im Anschluss an Definition 3.11 besprochen haben. Tatsächlich schränken diese vier Eigenschaften die in Frage kommenden Funktionen so weit ein, dass die Logarithmusfunktion

$$I(p) = \log \frac{1}{p}$$

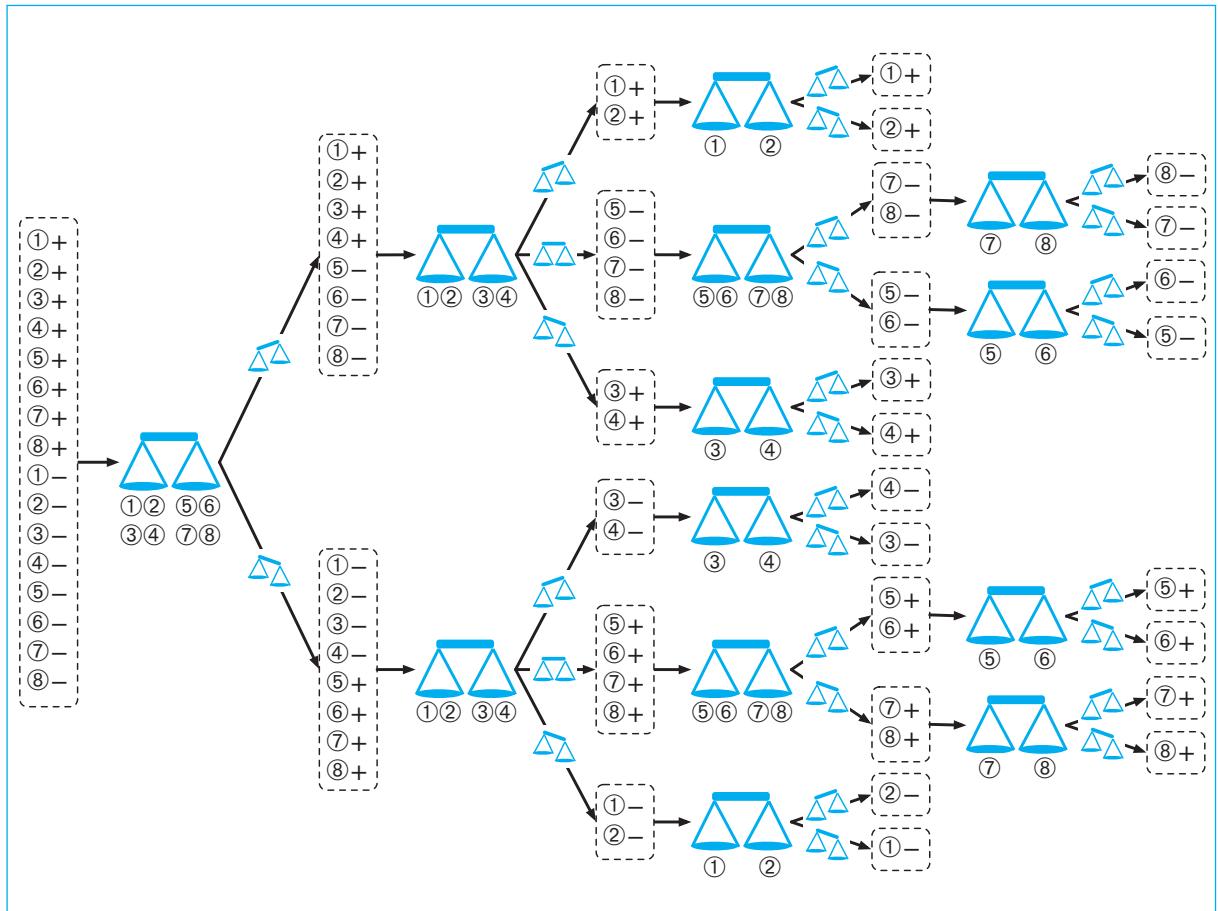
als einziger Kandidat übrig bleibt. Ganz eindeutig ist die Funktion dennoch nicht bestimmt: Für die Basis dürfen wir nämlich einen beliebigen Wert einsetzen, ohne die vier postulierten Eigenschaften zu verletzen.

Dass wir uns in Definition 3.11 für den Logarithmus zur Basis 2, den *Logarithmus dualis*, entschieden haben, ist aber bei weitem kein Zufall. Die getroffene Wahl sorgt dafür, dass wir die Information mit einer Einheit messen können, die uns aus der Informatik wohlvertraut ist: dem *Bit*.

In den folgenden Betrachtungen werden wir herausarbeiten, dass sich der theoretisch geprägte Informationsbegriff auch ganz praktisch für die Konstruktion von Lösungsstrategien einsetzen lässt. Welche Strategien wir dabei im Sinn haben, werden wir an einem konkreten Beispiel herausarbeiten: dem *Wiegeproblem*. Die Ausgangssituation ist schnell erklärt. Vor uns liegen 8 Kugeln,

① ② ③ ④ ⑤ ⑥ ⑦ ⑧

von denen genau eine ein anderes Gewicht besitzt. Leider wissen wir weder, wo sich diese Kugel befindet, noch, ob sie schwerer oder leichter ist. Unsere Aufgabe ist es, den Abweichler mithilfe einer Pendelwaage



**Abb. 3.42:** Mögliche Lösung des Kugelproblems mit 8 Kugeln

( $\Delta\Delta$ ) zu identifizieren und dabei mit möglichst wenigen Wiegeschritten auszukommen.

Ad hoc begegnen die meisten Menschen diesem Problem mit einer Strategie, wie sie in Abbildung 3.42 dargestellt ist. Im ersten Schritt werden die 8 Kugeln in zwei Vierergruppen aufgeteilt und mit der Pendelwaage gewogen. Im zweiten Schritt werden die verbleibenden 4 Kugeln in zwei Zweiergruppen aufgeteilt und erneut gewogen. Fahren wir auf diese Weise fort, so haben wir die gesuchte Kugel nach maximal vier Wiegeschritten identifiziert.

Wir wollen die verfolgte Strategie etwas genauer untersuchen und betrachten den ersten Wiegevorgang. Schlägt die Waage nach links aus, so können wir den Schluss ziehen, dass

- die linke Vierergruppe eine Kugel enthält, die schwerer ist, oder
- die rechte Vierergruppe eine Kugel enthält, die leichter ist.

Schlägt die Waage nach rechts aus, so gilt etwas ganz Ähnliches. Wir können dann den Schluss ziehen, dass

- die linke Vierergruppe eine Kugel enthält, die leichter ist, oder
- die rechte Vierergruppe eine Kugel enthält, die schwerer ist.

Die Information, die wir in diesem Schritt gewinnen, können wir exakt beziffern. Zu Beginn herrscht maximale Unsicherheit, da wir weder wissen, wo sich die gesuchte Kugel befindet, noch, ob sie schwerer oder leichter ist als die anderen. Damit gibt es insgesamt 16 Möglichkeiten:

- Die erste Kugel ist schwerer als die anderen. (☞ ①+)
- Die zweite Kugel ist schwerer als die anderen. (☞ ②+)
- ...
- Die erste Kugel ist leichter als die anderen. (☞ ①-)
- Die zweite Kugel ist leichter als die anderen. (☞ ②-)
- ...

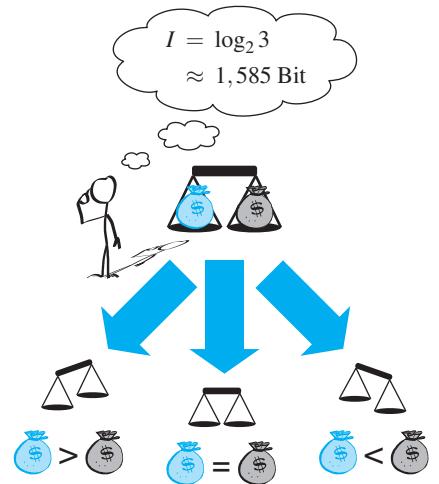
Teilen wir die 8 Kugel, wie oben beschrieben, in zwei Vierergruppen auf, so reduziert der erste Wiegeschritt die Anzahl der Möglichkeiten von 16 auf 8. In unserer formalen Terminologie entspricht dies einem Informationsgewinn von 1 Bit.

Lässt sich die Strategie verbessern? Abbildung 3.43 zeigt, dass uns die Pendelwaage eine von drei möglichen Antworten liefert, sodass nach 3 Wiegeschritten

$$3 \cdot 3 \cdot 3 = 27$$

mögliche Konstellationen unterschieden werden können. Prinzipiell könnte es daher möglich sein, das Wiegeproblem mit 8 Kugeln in 3 Schritten zu entscheiden.

Wir können dieses Argument auch informationstheoretisch formulieren. Eine Strategie ist dann optimal, wenn sie uns in jedem Schritt ein möglichst hohes Maß an Information liefert. Je größer die Information ist, die wir gewinnen, desto schneller werden die verbleibenden Möglichkeiten reduziert, und desto schneller erreichen wir die gesuchte Antwort. Dabei müssen wir darauf achten, dass die möglichen Ausgänge



**Abb. 3.43:** Jeder Wiegevorgang endet mit drei möglichen Antworten. Kommt jede Antwort gleich häufig vor, so entspricht dies einem Informationsgehalt von 1,585 Bit.

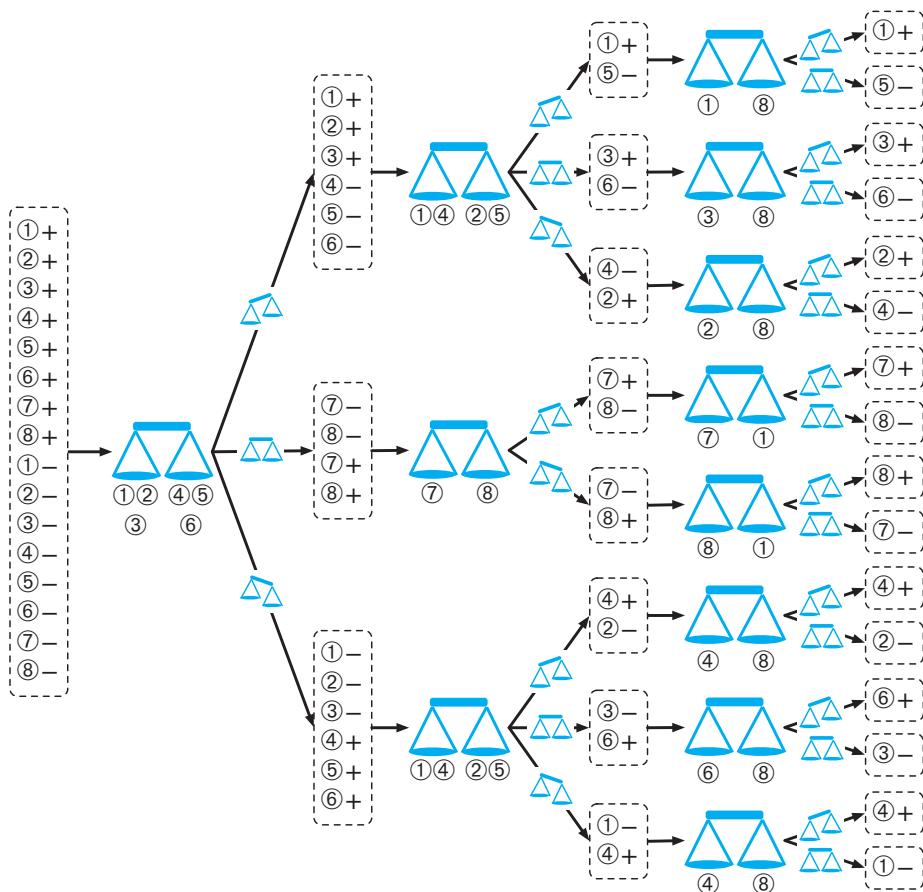


Abb. 3.44: Das Kugelproblem mit 8 Kugeln lässt sich in 3 Wiegeschritten lösen.

des Wiegevorgangs ein möglichst ausgeglichenes Maß an Information liefern, denn nur so entsteht eine balancierte Baumstruktur, in der die Pfade von der Wurzel zu den Blättern in etwa gleich lang sind.

Ein ausgeglichenes Maß an Information zu erhalten, ist gleichbedeutend mit der Aussage, dass die möglichen Ausgänge eines Wiegevorgangs mit den gleichen Wahrscheinlichkeiten eintreten, und genau diese Eigenschaft wird in der Strategie aus Abbildung 3.42 verletzt. Wenn wir im ersten Schritt jeweils vier Kugeln auf die beiden Seiten der Waage legen, wird diese jeweils mit einer Wahrscheinlichkeit von  $\frac{1}{2}$  nach links

oder rechts ausschlagen, aber mit einer Wahrscheinlichkeit von 0 in der ursprünglichen Position verharren. Die Fähigkeit der Waage, die linke und rechte Seite als gleich schwer zu erkennen, wird von dieser Strategie völlig ignoriert, und damit wird auf wichtige Information verzichtet.

Bei einer optimalen Strategie würde die Waage mit der gleichen Wahrscheinlichkeit nach links ausschlagen, nach rechts ausschlagen oder in ihrer Position verharren:

$$p(\Delta\overline{\Delta}) = \frac{1}{3}$$

$$p(\overline{\Delta}\Delta) = \frac{1}{3}$$

$$p(\overline{\Delta}\overline{\Delta}) = \frac{1}{3}$$

Die 3 möglichen Antworten, die uns die Waage in jedem Schritt liefert, entsprechen dann einem Informationsgehalt von jeweils

$$\log_2 3 \approx 1,585 \text{ Bit}$$

Eine von insgesamt 16 Möglichkeiten zu kennen, entspricht einer Information von 4 Bit, sodass mindestens

$$\left\lceil \frac{4}{1,585} \right\rceil = 3$$

Wiegeschritte notwendig sind.

Eine Strategie, die nach dem geschilderten Grundsatz konstruiert wurde, ist in Abbildung 3.44 zu sehen, und ein Blick auf die Pfadlängen zeigt, dass sie die Eigenschaft erfüllt, nach der wir gesucht haben: Sie löst das Wiegeproblem mit 8 Kugeln in 3 Schritten.

Alles in allem haben unsere Überlegungen erste Belege dafür geliefert, dass der formale Informationsbegriff sinnvoll gewählt ist, und in den nächsten Kapiteln werden Sie sehen, dass dieser Eindruck keinesfalls täuscht. Dort wird uns der Informationsbegriff auf Schritt und Tritt begleiten und seine Bedeutung weiter unter Beweis stellen.

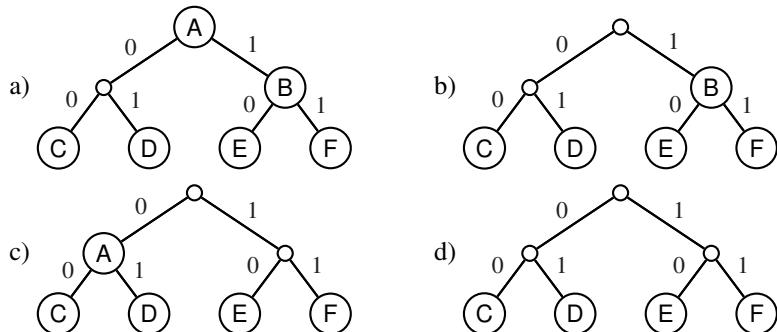
## 3.7 Übungsaufgaben

### Aufgabe 3.1



**Webcode  
3009**

Welche der folgenden Binäräbäume beschreiben einen präfixfreien Code und welche nicht?



### Aufgabe 3.2



**Webcode  
3013**

In dieser Aufgabe geht es um die nachstehenden Codes:

$\sigma$	$c_1(\sigma)$	$\sigma$	$c_1(\sigma)$	$\sigma$	$c_2(\sigma)$	$\sigma$	$c_2(\sigma)$	$\sigma$	$c_3(\sigma)$	$\sigma$	$c_3(\sigma)$
A	00	E	011	A	00	E	011	A	00	E	011
B	000	F	100	B	000	F	100	B	000	F	100
C	001	G	0000	C	001	G	0000	C	001	G	101
D	010	H	00000	D	010	H	0001	D	010	H	0000

- a) Zeigen Sie, dass keiner der abgebildeten Codes präfixfrei ist.
- b) Lässt sich die Präfixfreiheit herstellen, ohne die Codewortlängen zu verändern?

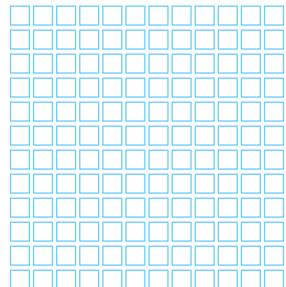
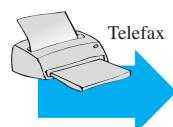
### Aufgabe 3.3



**Webcode  
3119**

In diesem Kapitel haben Sie gelernt, nach welchem Verfahren Telefaxgeräte eine eingescannte Textvorlage codieren. Ihre Aufgabe ist es, aus der unten angegebenen Bitfolge zunächst die Lauflängen zu extrahieren und anschließend das übertragene Pixelbild zu rekonstruieren. Die Übertragung beginnt mit einer Lauflänge für weiße Pixel.

001100000111110000111011  
000111011001011100111001  
100101111000111110000111  
1100011110101



Das *Microcom Network Protocol 5*, kurz MNP-5, verwendet eine spezielle Variante der Lauflängencodierung, die eine Mischform der Ansätze ist, die in diesem Kapitel vorgestellt wurden. Das Protokoll gibt vor, Symbolketten erst dann zu komprimieren, wenn darin mindestens 3 gleiche Symbole vorkommen. Eine solche Kette wird dann mit 4 Byte codiert. Die ersten drei Byte sind Wiederholungen des gleichen Zeichens und das vierte Byte ist eine Lauflänge. Diese bestimmt, wie viele weitere Wiederholungen des Zeichens folgen.

Codieren Sie die folgenden Textnachrichten auf die beschriebene Weise:

a) abcabcabcabcabc

b) abbcccddeeeeeee

c) aaabbcccddeeee

Übrigens: Das MNP-5- Protokoll hatte seine größte Praxisbedeutung Anfang der 90er-Jahre. Es wurde damals in analogen Modems eingesetzt, die Daten mithilfe künstlich erzeugter Töne über konventionelle Telefonleitungen übertrugen.

Auf Seite 176 haben Sie im Beweis von Satz 3.1 ein Verfahren kennengelernt, mit dem sich präfixfreie Codes systematisch konstruieren lassen. Ihre Aufgabe ist es, einen präfixfreien Code zu finden, dessen Codewörter die folgenden Längen aufweisen:

$$l_1 = 2, l_2 = 2, l_3 = 2, l_4 = 3, l_5 = 4, l_6 = 4, l_7 = 4$$

- a) Wie viele Zeichen muss das Codealphabet nach der Kraft'schen Ungleichung mindestens umfassen, damit ein präfixfreier Code mit den gewünschten Codewortlängen konstruiert werden kann?
- b) Führen Sie die Konstruktion mit einem passenden Codealphabet durch.

Die folgende Rechnung scheint zu belegen, dass der Algorithmus von Seite 176, entgegen dem Bewiesenen, doch nicht immer einen präfixfreien Code hervorbringt:



$l_1 = 4, l_2 = 2, l_3 = 3$	$+ 2^{-2}$	$0,0000$	0000
		$0,01$	
	$+ 2^{-3}$	$\underline{0,0100}$	01
		$0,001$	
		$\underline{0,0110}$	011

- a) Was haben wir bei der Berechnung falsch gemacht?
- b) An welcher Stelle ist die Argumentation, die wir im Beweis von Satz 3.1 verwendet haben, hier fehlerhaft?

### Aufgabe 3.4



Webcode

3175

### Aufgabe 3.5



Webcode

3274

### Aufgabe 3.6



Webcode

3290

**Aufgabe 3.7**

In dieser Aufgabe geht es um die folgende Codierung:

   
 Webcode  
3306

$$A \mapsto 01$$

$$J \mapsto 011$$

$$Z \mapsto 11$$

- a) Decodieren Sie die empfangene Bitsequenz

010110111011110111111011111101111111.

- b) Lässt sich jede empfangene Bitsequenz eindeutig decodieren?

**Aufgabe 3.8**

Bewerten Sie den Wahrheitsgehalt der folgenden Aussagen:

Richtig      Falsch

   
 Webcode  
3351

Jeder präfixfreie Code ist eindeutig decodierbar.



Jeder eindeutig decodierbare Code ist präfixfrei.



Jeder Code, der präfixfrei ist, erfüllt die Kraft'sche Ungleichung.



Jeder Code, der die Kraft'sche Ungleichung erfüllt, ist präfixfrei.


**Aufgabe 3.9**

Es seien die folgenden präfixfreien Codierungen gegeben:

   
 Webcode  
3449

$\sigma$	$c_1(\sigma)$
A	00
B	01
R	10
K	110
D	111

$\sigma$	$c_2(\sigma)$
A	0
B	10
R	110
K	1110
D	1111

$\sigma$	$c_3(\sigma)$
A	1
B	01
R	001
K	0001
D	0000

$\sigma$	$c_4(\sigma)$
A	0
B	100
R	101
K	110
D	111

- a) Codieren Sie mit  $c_1, \dots, c_4$  nacheinander die Zeichenkette ABRAKADABRA. Welche Codierung ist für diese Zeichenkette am besten geeignet?
- b) Gibt es eine präfixfreie Codierung, die ein noch besseres Ergebnis liefert?

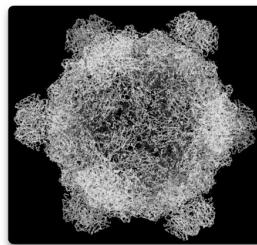
Das nebenstehende 3D-Modell zeigt φX174, das im Jahr 1977 als erstes vollständig sequenziertes Virus in die Wissenschaftsgeschichte einging. Das Genom von φX174 ist ungeheuer kompakt und besteht aus einem einzigen DNA-Strang mit insgesamt 5386 Nukleotiden. Entziffert wurde die DNA-Sequenz von der Forschungsgruppe von Frederick Sanger. Für seine bahnbrechenden Entdeckungen auf dem Gebiet der Gentechnik wurde dem englischen Biochemiker im Jahr 1980, zusammen mit Walter Gilbert, der Chemie-Nobelpreis verliehen, „*for their contributions concerning the determination of base sequences in nucleic acids*“. Für Sanger war dies bereits die zweite Ehrung aus Stockholm. Er ist einer von bis dato vier Forschern, die den Nobelpreis zweimal erhalten haben.

Der folgende Ausschnitt zeigt einen Teil des Genoms von φX174:

```

1  gagttttatc gcttccatga cgcaagaattt aacacttcg gatatttctg
51 atgagtcgaa aaattatctt gataaaggcg gaattactac tgcttggtta
101 cgaattaaat cgaagtggac tgctggcggaa aatagaaaa attcgaccta
151 tccttgcga gctcgaaag ctcttacttt gcgcacccccc gccatcaact
201 aacgatctg tcaaaaactg acgcgttggaa tgaggagaag tggcttaata
251 tgcttggcac gtgcgtcaag gactgggtta gatatgagtc acattttgtt
301 catggtagag attcttctgt tgacattttt aaagaggcggtt gattactatc
351 tgagtcccat gctgttcaac cactaatagg taagaaatca tgagtcagt
401 tactgaacaa tccgtacgtt tccagacccg tttggctctt attaagctca
451 ttcagggttc tgccgttttg gattnaaccg aagatgattt cgatttctg
501 acgagtaaca aagtggat tgctactgac cgctctcggt ctgcgtcggt
551 cggttgggat tgcgtttagt gtacgctggaa ctttggggaa taccctcggt
601 ttccctgtcc tggtagttt attgctgccc tcattgttta ttatgttcat
651 cccgtcaaca ttcaaacccg ctgtctcata atgaaaggcg ctgaatttac
701 ggaaaacattt attaaatggcg tgcggcgcc ggttaaagcc gctgaatttgc
751 tgcgtttac ctggcggtta cgcccgaggaa aacactgacgt ttttactgac
...
5301 gacgttttgg cggcgcaacc tggacgaca aatctgtca aattttatgcg
5351 cgcttcgata aaaatgttgc gcttatccaa cctgcg

```



### Aufgabe 3.10



**Webcode  
3487**



Frederick Sanger  
(1918 – 2013)

- Welche Aminosäuresequenz wird durch die farbig markierten Basen codiert?
- Sicher ist Ihnen aufgefallen, dass das letzte Triplet (tga) eines von dreien ist, die in Tabelle 3.2 nicht auftauchen. Finden Sie heraus, welche spezielle Bedeutung diesen Sequenzen zukommt.

Vielleicht ist Ihnen bei der Betrachtung von Abbildung 3.14 aufgefallen, dass die Zahlen 0 bis 9 in der Brailleschrift mit den gleichen Zeichen dargestellt werden wie die Buchstaben a bis j. Streng genommen liegt damit gar keine Codierung im Sinne von Definition 3.1 vor, da die Zuordnung nicht injektiv ist. Finde Sie heraus, wie in der Praxis mit dieser Mehrdeutigkeit umgegangen wird.

### Aufgabe 3.11



**Webcode  
3565**

**Aufgabe 3.12**
**Webcode  
3567**

Die Blindenschrift von Louis Braille ist eine Weiterentwicklung der militärischen *Nacht-schrift* (franz. *Écriture nocturne*) von Charles Barbier de La Serre [6, 60]. In der Barbier-Schrift wird jedes Symbol, genau wie bei Braille, durch zwei Spalten mit tastbaren Punkten dargestellt. Die Barbier-Symbole enthalten mit bis zu 6 Punkten pro Spalte mehr Punkte als die Braille-Symbole und stehen nicht, wie bei Braille, für Buchstaben, sondern für Laute. Um einen der 36 französischen Laute zu decodieren, mussten die Punkte in den Spalten gezählt werden. Die ermittelten Zahlen bestimmten die Zeilen- und Spaltennummer einer Matrix, in der die entsprechende Silbe nachgeschlagen werden konnte. Konkret sah die Zuordnung so aus:

	1	2	3	4	5	6	
1	a	i	o	u	é	è	1
2	an	in	on	un	eu	ou	2
3	b	d	g	j	v	z	3
4	p	t	q	ch	f	s	4
5	l	m	n	r	gn	ll	5
6	oi	oin	ian	ein	ion	ieu	6

Versuchen Sie, herauszufinden, welche drei französischen Wörter hier codiert werden:

**Aufgabe 3.13**
**Webcode  
3579**

In Tetrads-Codes ist die Durchführung arithmetischer Operationen schwieriger als in den klassischen Zahlensystemen. Die folgenden Rechnungen zeigen, dass wir im BCD-, Stibitz- oder Aiken-Code falsche Ergebnisse erhalten, wenn wir die Binärziffern in der gewohnten Art und Weise addieren.

**■ BCD-Code:**

$$\begin{array}{r}
 0100 \quad (4) \\
 + \quad 1001 \quad (9) \\
 \hline
 = \quad 1101 \quad (?)
 \end{array}
 \qquad
 \begin{array}{r}
 1001 \quad (9) \\
 + \quad 1001 \quad (9) \\
 \hline
 = \quad 0001 \quad 0010 \quad (12)
 \end{array}$$

**■ Stibitz-Code:**

$$\begin{array}{r}
 0101 \quad (2) \\
 + \quad 1001 \quad (6) \\
 \hline
 = \quad 1110 \quad (?)
 \end{array}
 \qquad
 \begin{array}{r}
 1001 \quad (6) \\
 + \quad 1100 \quad (9) \\
 \hline
 = \quad 0001 \quad 0101 \quad (?)
 \end{array}$$

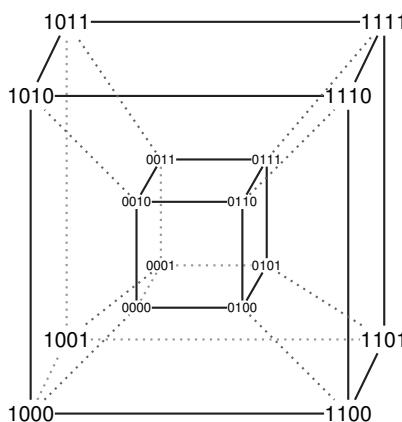
## ■ Aiken-Code:

$$\begin{array}{r}
 & 1100 & (6) \\
 + & 1100 & (6) \\
 = & 0001 & 1000 & (?) \\
 \end{array}
 \quad
 \begin{array}{r}
 & 0010 & (2) \\
 + & 0100 & (4) \\
 = & 0110 & (?) \\
 \end{array}$$

Wie kann in den genannten Codes mit dem beschriebenen Problem umgegangen werden?

Nachstehend sehen Sie die zweidimensionale Projektion eines vierdimensionalen Hyperwürfels. Die Ecken des Würfels sind mit den Elementen des Vektorraums  $\mathbb{Z}_2^4$  beschriftet und genau dann durch eine Kante miteinander verbunden, wenn sich ihre zugeordneten Bitvektoren an einer einzigen Stelle unterscheiden.

Der vierdimensionale Hyperwürfel kann dazu verwendet werden, um die Tetrade-Codes aus Abschnitt 3.4.2 grafisch zu veranschaulichen. Dabei wird eine Codierung durch den Pfad dargestellt, der durch die Codewörter  $c(0), c(1), c(2), \dots$  gegeben ist.

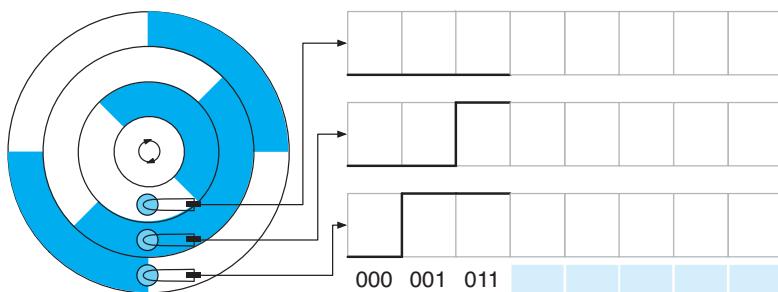
**Aufgabe 3.14**

**Webcode  
3678**

a) Tragen Sie den Pfad des erweiterten Gray-Codes aus Tabelle 3.5 in den Hyperwürfel ein.

b) An welchen bekannten Begriff aus der Graphentheorie erinnert Sie das Ergebnis?

In der abgebildeten Grafik sehen Sie eine Messscheibe, die im Uhrzeigersinn gedreht werden kann:

**Aufgabe 3.15**

**Webcode  
3691**

Die Messscheibe ist in 3 konzentrische Ringe unterteilt, die wechselweise mit unterschiedlichen Materialien beschichtet sind. Jeder Ring wird von einem separaten Sensor abgetastet und erzeugt in Abhängigkeit der Oberflächenbeschaffenheit ein binäres Signal. Wird die

Scheibe um volle  $360^\circ$  gedreht, so entstehen 8 verschiedene Bitmuster, von denen die ersten drei bereits eingezeichnet sind.

- Vervollständigen Sie das Signaldiagramm und bestimmen Sie die restlichen 5 Bitmuster.
- Welcher Ihnen bekannte Code wird hier erzeugt?

### Aufgabe 3.16



**Webcode  
3718**

In diesem Kapitel haben Sie mit dem UTF-8-Format eine weit verbreitete Codierung von Unicode-Zeichen kennengelernt.

- Wie sehen die Unicode- und die UTF-8-Bitmuster der Zeichen ‚a‘ und ‚€‘ aus?
- Decodieren Sie die im UTF-8-Format vorliegende Binärsequenz 11000001 10100001. Was fällt Ihnen auf? Wie kann das Problem behoben werden?
- Die ersten 256 Zeichen des Unicodes sind identisch mit dem Zeichensatz Latin-1 der ISO 8859-1. Entspricht damit jeder ISO-8859-1-codierte Text automatisch auch dem UTF-8-Standard?
- Gegeben sei ein beliebiger Text im Unicode. Der Text wird zunächst in eine Zeichenkette der Programmiersprache Java übersetzt und anschließend in das UTF-8-Format konvertiert. Kann die UTF-8-Codierung zu einer Vergrößerung des Datenvolumens führen? Wenn nein, warum nicht? Wenn ja, um wie viel?

### Aufgabe 3.17



**Webcode  
3779**

Nachstehend sind die Codetabellen des ungeraden und des geraden 5-Bit-Paritätscodes abgebildet.

■ Ungerader Paritätscode

$u$	$c(u)$	$u$	$c(u)$
0000	00001	1000	10000
0001	00010	1001	10011
0010	00100	1010	10101
0011	00111	1011	10110
0100	01000	1100	11001
0101	01011	1101	11010
0110	01101	1110	11100
0111	01110	1111	11111

■ Gerader Paritätscode

$u$	$c(u)$	$u$	$c(u)$
0000	00000	1000	10001
0001	00011	1001	10010
0010	00101	1010	10100
0011	00110	1011	10111
0100	01001	1100	11000
0101	01010	1101	11011
0110	01100	1110	11101
0111	01111	1111	11110

- Welcher der beiden Codes ist ein linearer Code und welcher nicht?

- b) Erzeugen Sie für den linearen Code die Generatormatrix und die Kontrollmatrix.
- c) Erzeugen Sie den dualen Code. Welche Dimension hat der zugehörige Untervektorraum?

Gegeben sei die folgende Generatormatrix:

$$G := \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

**Aufgabe 3.18**

**Webcode  
3882**

- a) Tragen Sie die von  $G$  erzeugten Codewörter in die abgedruckte Tabelle ein:

$u$	$u \cdot G$	$u$	$u \cdot G$
0000		1000	
0001		1001	
0010		1010	
0011		1011	
0100		1100	
0101		1101	
0110		1110	
0111		1111	

- b) Welchem Ihnen bekannten Code ist dieser Code ähnlich?  
 c) Wie lautet die Kontrollmatrix?  
 d) Ist der Code selbstdual?

Von der Binärzahl  $u$  wissen wir lediglich, dass sie fünf Stellen hat und aus der folgenden Menge stammt:

$$C = \{00000, 00001, 00010, 00110, 01000, 01010, 01110, 10010\}$$

Ihnen steht ein Orakel an der Seite, das Sie gezielt nach dem Wert des  $i$ -ten Bits fragen können. In dieser Aufgabe geht es darum, eine optimale Fragestrategie zu entwickeln.


**Aufgabe 3.19**

**Webcode  
3890**

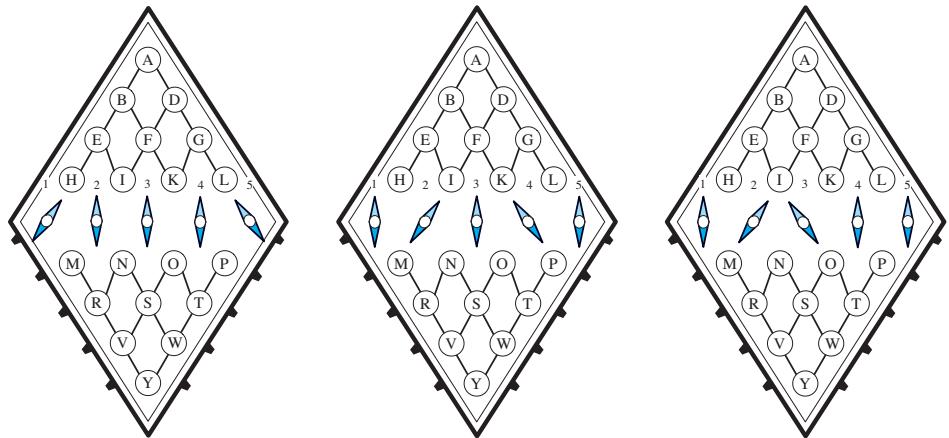
- a) Wie viele Fragen müssen wir aus informationstheoretischer Sicht mindestens stellen?  
 b) Entwickeln Sie eine Fragestrategie, die mit der ermittelten Anzahl an Fragen auskommt.

**Aufgabe 3.20**

**Webcode**  
**3904**

In Kapitel 1 haben Sie den Fünf-Nadel-Telegrafen von William Cooke und Charles Wheatstone kennengelernt. Wir wollen die Apparatur in dieser Aufgabe informationstheoretisch untersuchen und sehen uns hierfür eine Reihe von Nadelstellungen an.

- a) Lesen Sie in jedem Beispiel die Nadelstellungen von links nach rechts ab, und geben Sie an, wie viel Information, gemessen in Bits, die betrachtete Nadel liefert. Denken Sie daran: „*Information ist das, was Unsicherheit beseitigt*“. Ein Bit an Information reduziert das Maß an Unsicherheit um die Hälfte.



Informationsgehalte

1. Nadel:	<input type="text"/>
2. Nadel:	<input type="text"/>
3. Nadel:	<input type="text"/>
4. Nadel:	<input type="text"/>
5. Nadel:	<input type="text"/>
Summe:	<input type="text"/>

Informationsgehalte

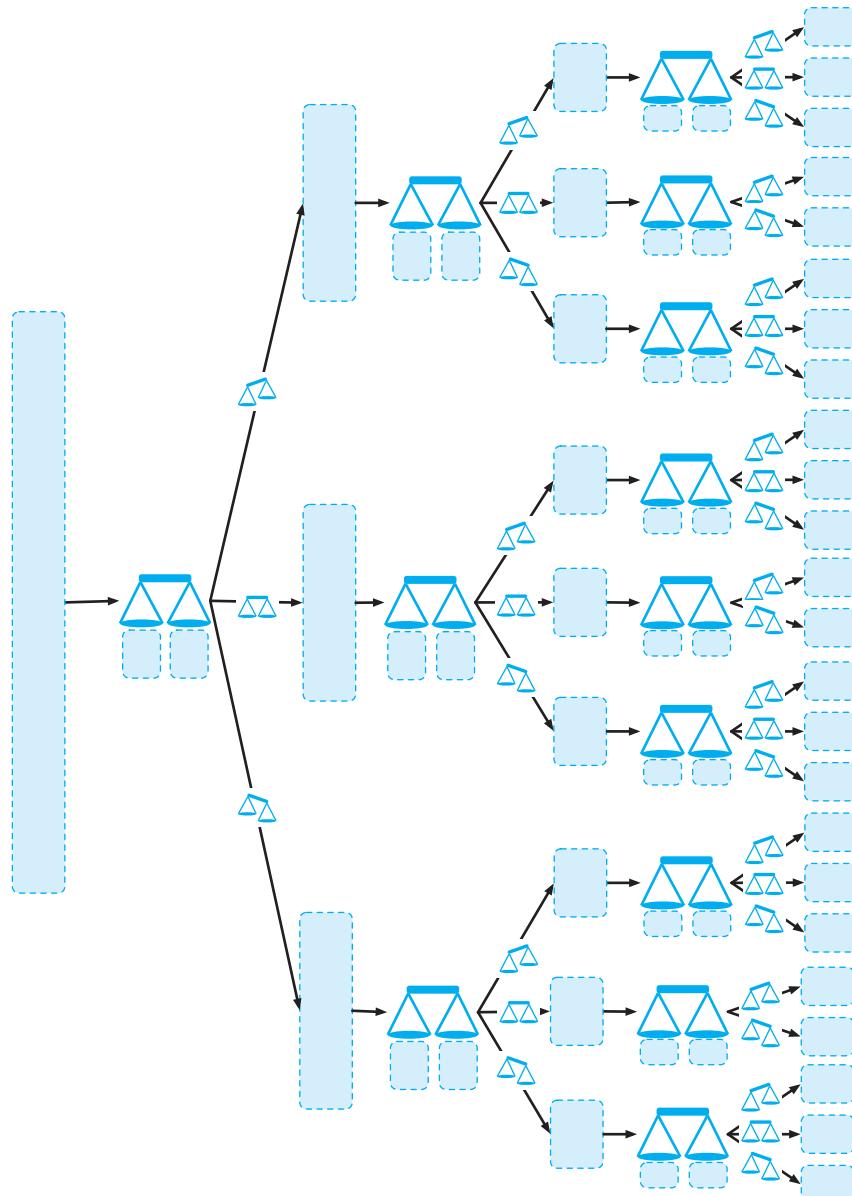
1. Nadel:	<input type="text"/>
2. Nadel:	<input type="text"/>
3. Nadel:	<input type="text"/>
4. Nadel:	<input type="text"/>
5. Nadel:	<input type="text"/>
Summe:	<input type="text"/>

Informationsgehalte

1. Nadel:	<input type="text"/>
2. Nadel:	<input type="text"/>
3. Nadel:	<input type="text"/>
4. Nadel:	<input type="text"/>
5. Nadel:	<input type="text"/>
Summe:	<input type="text"/>

- b) Hätten Sie die Summen auch ausrechnen können, ohne die Nadeln einzeln zu betrachten?
- c) Offenbar scheint die fünfte Nadel keinerlei Information in sich zu tragen. Klären Sie auf, warum dies so ist.

In diesem Kapitel haben Sie gelernt, wie sich das Kugelproblem mit 8 Kugeln durch dreimaliges Wiegen lösen lässt. Versuchen Sie, das Problem mit 12 Kugeln zu lösen, ohne die Anzahl der Wiegeschritte zu erhöhen. Ergänzen Sie hierzu das nachstehende Diagramm. Ab welchem Wert von  $n$  kann das Kugelproblem mit  $n$  Kugeln auf keinen Fall mehr durch dreimaliges Wiegen lösbar sein?

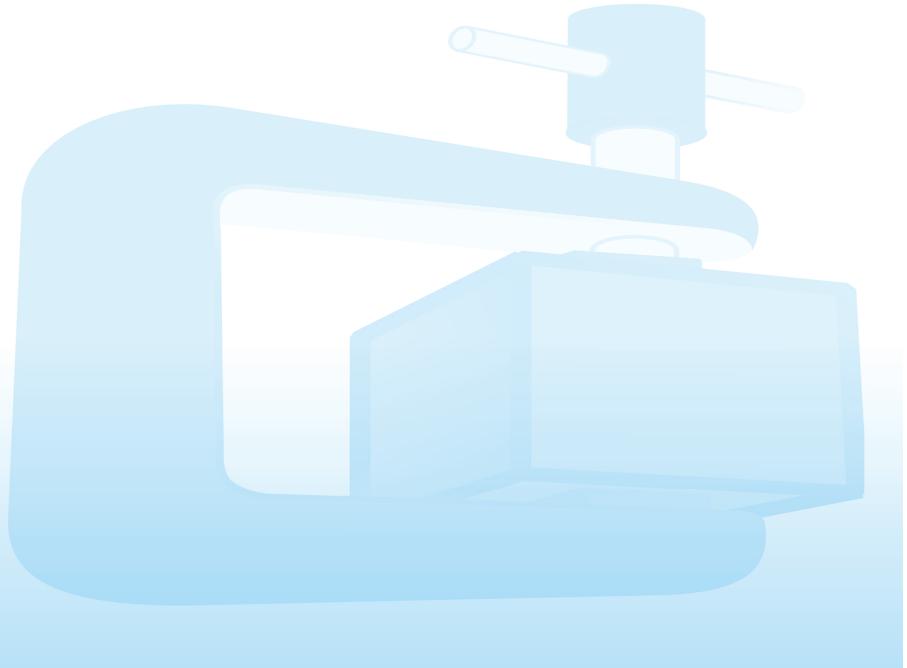
**Aufgabe 3.21****Webcode****3914**

## 4 Quellencodierung

---

In diesem Kapitel werden Sie ...

- ein formales Modell für Informationsquellen entwickeln,
- gedächtnislose und gedächtnisbehaftete Quellen auseinanderhalten,
- die Codierungen von Fano, Shannon und Huffman vergleichen,
- verstehen, wie sich diese von der arithmetischen Codierung unterscheiden,
- sich einen Überblick über die wichtigsten Substitutionscodierungen verschaffen,
- die Burrows-Wheeler-Transformation ausführen
- und mit der Move-to-front-Codierung kombinieren.



## 4.1 Motivation

In diesem Kapitel behandeln wir die Komponente, die in Shannons Kommunikationsmodell am Anfang der digitalen Datenübertragung steht: die *Daten-* oder *Informationsquelle*. Im Kern drehen sich die angestellten Untersuchungen um zwei Fragestellungen, die Shannon folgendermaßen formuliert hat:

*„How is an information source to be described mathematically, and how much information in bits per second is produced in a given source?“*

Claude Shannon [81]

Direkt im Anschluss an diese Textstelle präzisiert Shannon, was er damit genau im Sinn hat:

*„The main point at issue is the effect of statistical knowledge about the source in reducing the required capacity of the channel, by the use of proper encoding of the information.“*

Claude Shannon [81]

An diesem Zitat wollen wir uns orientieren und aus ihm den Fahrplan für die nächsten beiden Kapitel ableiten:

- In Abschnitt 4.2 entwickeln wir mehrere statistische Modelle, die für die Beschreibung von Informationsquellen geeignet sind. Wir beginnen mit der Definition von *gedächtnislosen Quellen*, die Symbole entsprechend einer statischen Wahrscheinlichkeitsverteilung emittieren. Anschließend werden wir diese Modelle zu *gedächtnisbehafteten Quellen* ausbauen, in denen sich hintereinander emittierte Symbole gegenseitig in ihren Auftrittswahrscheinlichkeiten beeinflussen.
- In den Abschnitten 4.3 bis 4.8 besprechen wir wichtige Verfahren aus dem Bereich der Datenkompression. Im Kern dieser Algorithmen steht die Idee, die statistischen Eigenschaften der Datenquelle geschickt auszunutzen, sodass die zu übertragende Nachricht in einer Form an den Sender übergeben wird, die den Informationsgehalt maximiert. Erst hierdurch wird es möglich, Information mit einer Geschwindigkeit zu übertragen, die sich am Limit der Kanalkapazität bewegt.

- In Kapitel 5 beschäftigen wir uns mit den Grenzen der Quellencodierung. Dort arbeiten wir heraus, wie die maximal erreichbare Kompressionsrate durch eine Maßzahl quantitativ erfasst werden kann und wie sich diese Zahl aus den statistischen Kenngrößen der Datenquelle berechnen lässt. Die angestellten Überlegungen werden uns bis hin zu Shannons berühmten Quellencodierungstheorem führen, das eine präzise Aussage über die maximale Anzahl an Symbole trifft, die über einen Übertragungskanal pro Sekunde verlustfrei gesendet werden können.

## 4.2 Die Informationsquelle

Zu Beginn dieses Abschnitts wollen wir uns eine intuitive Vorstellung von diskreten Informations- oder Datenquellen verschaffen. In seiner Arbeit aus dem Jahr 1948 zeichnet Shannon das folgende Bild:

*„We can think of a discrete source as generating the message, symbol by symbol. It will choose successive symbols according to certain probabilities depending, in general, on preceding choices as well as the particular symbols in question. A physical system, or a mathematical model of a system which produces such a sequence of symbols governed by a set of probabilities, is known as a stochastic process. We may consider a discrete source, therefore, to be represented by a stochastic process. Conversely, any stochastic process which produces a discrete sequence of symbols chosen from this set may be considered a discrete source.“*

Claude Shannon [81]

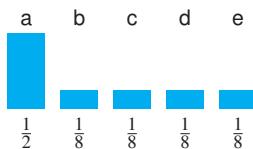
Damit ist die Grundidee skizziert: Aus der Ferne können wir uns eine Informationsquelle als eine Komponente vorstellen, die einen kontinuierlichen Strom von Symbolen eines endlichen Quellenalphabets  $\Sigma$  emittiert.



### Definition 4.1 (Diskrete Quelle)

Eine diskrete Quelle  $X$  mit dem Quellenalphabet  $\Sigma = \{\sigma_1, \dots, \sigma_n\}$  emittiert einen unendlichen Strom von Symbolen aus der Menge  $\Sigma$ . Jedes Symbol besitzt eine Auftrittswahrscheinlichkeit  $> 0$ .

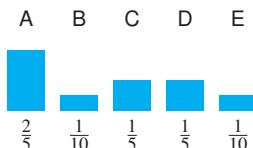
Quelle 1



Typische Sequenz:

abaaecadaeadaabab  
ceadaabaaceadaaa  
abaaceadac ...

Quelle 2



Typische Sequenz:

DAAEEDCBACADAEAC  
ADBCADAACEABDAC  
AACDABEDAC ...

Abb. 4.1: Beispiele gedächtnisloser Quellen

Beachten Sie, dass wir in unserem theoretischen Modell davon ausgehen, dass die Quelle kontinuierlich sendet und damit eine potenziell unendlich lange Sequenz produziert. Viele Datenquellen, die wir in der Realität beobachten können, senden aber nur kurze Sequenzen. Der Unterschied klingt marginal, hat aber einschneidende Konsequenzen für die Interpretation der weiter unten hergeleiteten Ergebnisse. Nahezu alle Theoreme der Informations- und Codierungstheorie machen Grenzwertaussagen, in denen die Länge der betrachteten Symbolsequenzen gedanklich gegen unendlich strebt. Die Ergebnisse sind deshalb stets als Näherungsgrößen zu betrachten, denen wir uns mit zunehmender Nachrichtenlänge zwar asymptotisch annähern, die wir aber nicht unbedingt exakt erreichen.

## 4.2.1 Gedächtnislose Quellen

Die einfachsten diskreten Datenquellen sind jene, die in der Informationstheorie als *gedächtnislos* (*memoryless*) bezeichnet werden. Zwei solche Quellen sind exemplarisch in Abbildung 4.1 zu sehen.



### Definition 4.2 (Gedächtnislose Quelle)

Eine diskrete Quelle über dem Alphabet  $\Sigma = \{\sigma_1, \dots, \sigma_n\}$  heißt

- gedächtnislos, wenn sie die Symbole  $\sigma_1, \dots, \sigma_n$ 
  - stochastisch unabhängig voneinander
  - mit den Wahrscheinlichkeiten  $p(\sigma_1), \dots, p(\sigma_n)$  emittiert.

Eine gedächtnislose Quellen emittiert ihre Symbole demnach zu jedem beliebigen Zeitpunkt mit den gleichen, konstanten Wahrscheinlichkeiten. Dabei ist es bedeutungslos, welche Symbole vorher emittiert wurden, es existieren also keinerlei Kontextabhängigkeiten.

Eine gedächtnislose Quelle mit  $\Sigma = \{0, 1\}$  wird auch als *Bernoulli-Quelle* bezeichnet. Die einfachste Bernoulli-Quelle ist in Abbildung 4.2 zu sehen. Sie emittiert die Symbole 0 und 1 mit der gleichen Wahrscheinlichkeit und ist damit nichts anderes als ein perfekter Zufallszahlengenerator.

Die angesprochene Bernoulli-Quelle wollen wir noch ein wenig intensiver betrachten und richten unseren Blick auf die beiden Beispielsequenzen in Abbildung 4.2. Von einem statistischen Standpunkt aus gesehen, und genau diesen nimmt die Shannon'sche Informationstheorie

ein, sind beide Zahlensequenzen kaum zu unterscheiden; sie erfüllen die statistischen Eigenschaften von echten Zufallszahlen nahezu perfekt. Das Wörtchen „nahezu“ ist hier wichtig, denn tatsächlich ist nur die erste Folge wirklich zufällig (Abbildung 4.3). Sie wurde aus den Ziffern einer Zufallszahl generiert, die dem 1955 erschienenen Buch *A Million Random Digits with 100,000 Normal Deviates* entstammt [21]. Dieses sehr spezielle Werk enthält auf mehr als 600 dicht beschriebenen Seiten insgesamt 1.000.000 Ziffern einer experimentell ermittelten Zahlenfolge.

Die zweite Folge wurde dagegen systematisch generiert. Sie besteht aus den Nachkommaziffern der Zahl  $\pi$ , gerechnet modulo 2, und ist damit alles andere als eine Zufallszahl. Da die Ziffern der Zahl  $\pi$  aber fast wie in einer echten Zufallszahl verteilt sind, dürfen wir die Folge dennoch als eine typische Sequenz unserer Bernoulli-Quelle ansehen. Damit sind wir an einem zentralen Punkt angekommen: In der Shannon'schen Informationstheorie spielt ausschließlich die statistische Verteilung der emittierten Symbole eine Rolle, und nicht die Art und Weise, wie die Datenströme innerhalb der Datenquelle erzeugt werden.

## 4.2.2 Markov-Quellen

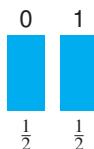
Alle bisher betrachteten Datenquellen waren gedächtnislos, d.h., die Wahrscheinlichkeit, mit der wir ein bestimmtes Zeichen beobachten konnten, war unabhängig davon, welche Zeichen vorher emittiert wurden. In der Tat sind nur wenige Datenquellen, die in der Praxis eine Rolle spielen, von dieser Art. Betrachten wir beispielsweise einen in deutsch verfassten Text, so treten die Buchstabenpaare ‚au‘, ‚ei‘ oder ‚ss‘ viel häufiger auf als das Buchstabenpaar ‚yy‘. Ähnliche Eigenschaften gelten für die englische Sprache. Dort können wir in Texten, die ohne Kunstworte auskommen, mit Sicherheit vorhersagen, dass auf den Buchstaben ‚q‘ der Buchstabe ‚u‘ folgt. Solche Kontextabhängigkeiten lassen sich elegant mithilfe von *Transitionswahrscheinlichkeiten* oder *Bigrammwahrscheinlichkeiten* beschreiben:



### Definition 4.3 (Wahrscheinlichkeitsbegriffe)

- Die *Transitionswahrscheinlichkeit*  $p_\sigma(\tau)$  ist die Wahrscheinlichkeit, dass auf das Symbol  $\sigma$  das Symbol  $\tau$  folgt.
- Die *Bigrammwahrscheinlichkeit*  $p(\sigma, \tau)$  ist die Wahrscheinlichkeit, dass die Textsequenz  $\sigma\tau$  emittiert wird.

Quelle 3



Typische Sequenzen:

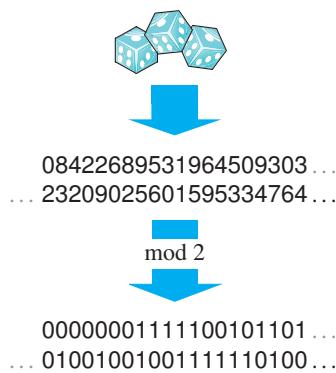
```

000000011110010110101001001 ...
... 001111101001100011000110110 ...
... 010101110101111011111010001 ...
... 0101110001001011100011111000 ...
... 111101010001010011101110011 ...
... 1011110011000011011110010100 ...
... 1010011101100101111001111110 ...
... 1110110001000101111000001110 ...
... 10001000111010110100 ...
101110011101111010000011010 ...
... 11100000111101111110100011 ...
... 010011010101000000000110000 ...
... 010001100111001100100001110 ...
... 0010000101100000111010001110 ...
... 1111000100001111010000100101 ...
... 110101101111000001001101101 ...
... 0110000001011100111100010001 ...
... 10000111001011010110 ...

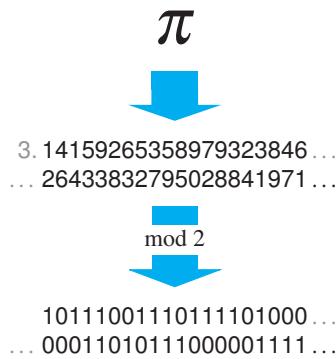
```

Abb. 4.2: Beispiel einer Bernoulli-Quelle

■ Sequenz 1



■ Sequenz 2



**Abb. 4.3:** Nur die erste Binärsequenz ist zufällig. Sie wurde aus den Ziffern einer 1955 publizierten Zahl generiert, die im Rahmen eines Zufallsexperiments ermittelt wurde.

Zwischen der Transitionswahrscheinlichkeit  $p_\sigma(\tau)$  und der Bigrammwahrscheinlichkeit  $p(\sigma, \tau)$  besteht ein enger Zusammenhang. Erstere ist die Wahrscheinlichkeit, dass die Quelle das Zeichen  $\tau$  emittiert, unter der Bedingung, dass vorher  $\sigma$  emittiert wurde. Für die Berechnung der bedingten Wahrscheinlichkeit  $p_\sigma(\tau)$  wird also nicht mehr der gesamte Ereignisraum betrachtet, sondern nur noch der Fall, dass zunächst das Zeichen  $\sigma$  emittiert wurde. Die Beschränkung des Ereignisraums können wir mathematisch so ausdrücken:

$$p_\sigma(\tau) = \frac{p(\sigma, \tau)}{p(\sigma)} \quad (4.1)$$

In Definition 4.1 haben wir explizit gefordert, dass die Auftrittswahrscheinlichkeit der Quellsymbole niemals 0 sein darf und somit jedes Symbol mit der Wahrscheinlichkeit 1 irgendwann einmal emittiert wird. Diese Voraussetzung ist hauptsächlich aus formalen Gründen notwendig; unter anderem stellt sie sicher, dass die rechte Seite von (4.1) stets wohldefiniert ist.

Kennen wir die Bigrammwahrscheinlichkeiten  $p(\sigma, \tau)$ , so können wir  $p(\sigma)$ , d. h. die Wahrscheinlichkeit, dass die Datenquelle das Symbol  $\sigma$  emittiert, über eine der beiden folgenden Formeln ausrechnen:

$$p(\sigma) = \sum_{\tau} p(\sigma, \tau) \quad (4.2)$$

$$= \sum_{\tau} p(\tau, \sigma) \quad (4.3)$$

Kombinieren wir (4.1) mit (4.2), so erhalten wir

$$p_\sigma(\tau) = \frac{p(\sigma, \tau)}{\sum_{\tau} p(\sigma, \tau)} \quad (4.4)$$

Diese Formel macht deutlich, wie die Transitionswahrscheinlichkeiten  $p_\sigma(\tau)$  aus den Bigrammwahrscheinlichkeiten  $p(\sigma, \tau)$  berechnet werden können. Gleichsam zeigt sie einen einfachen Weg auf, um sie direkt aus einer vorgegebenen Nachricht zu erzeugen:

- Im ersten Schritt wird die Nachricht Zeichen für Zeichen analysiert und für jedes Bigramm  $\sigma\tau$  in einer zweidimensionalen Tabelle gezählt, wie oft es vorkommt. Anschließend werden alle Tabelleneinträge durch die Länge der Nachricht dividiert. Das Ergebnis sind die Bigrammwahrscheinlichkeiten  $p(\sigma, \tau)$ . Abbildung 4.4 demonstriert das Gesagte an dem gleichen Beispiel, das Shannon in seiner Originalarbeit aus dem Jahr 1948 benutzt hat [81].

- Im zweiten Schritt werden aus den Bigrammwahrscheinlichkeiten  $p(\sigma, \tau)$  die Transitionswahrscheinlichkeiten  $p_\sigma(\tau)$  berechnet. Wir erhalten sie nach (4.4), indem wir für jede Zeile unserer Tabelle

- die Summe der Wahrscheinlichkeiten bilden
- und danach die Zeileneinträge durch diesen Wert dividieren.

Dies entspricht einer Normierung der Zeileneinträge bezüglich der Zeilensumme. Führen wir die Berechnung für unser Beispiel durch, so erhalten wir das in Abbildung 4.5 gezeigte Ergebnis.

Eine alternative Berechnungsformel entsteht, indem (4.1) nicht mit (4.2), sondern mit (4.3) kombiniert wird. Wir erhalten dann

$$p_\sigma(\tau) = \frac{p(\sigma, \tau)}{\sum_\tau p(\tau, \sigma)}$$

Im Gegensatz zu (4.4) werden die Elemente der  $n$ -ten Zeile jetzt bezüglich der kumulierten Wahrscheinlichkeiten aus der  $n$ -ten Spalte und nicht aus der  $n$ -ten Zeile normiert. Es spielt also keine Rolle, ob wir die Zeileneinträge entsprechend der  $n$ -ten Zeilensumme oder der  $n$ -ten Spaltensumme normieren; beide sind nach (4.2) und (4.3) identisch.

Als letzte Rechenübung wollen wir noch zwei elementare Aussagen über die Wahrscheinlichkeitssummen herleiten. Einerseits folgt aus

$$\sum_\sigma p(\sigma) = 1$$

und (4.2) unmittelbar

$$\sum_{\sigma, \tau} p(\sigma, \tau) = 1$$

Andererseits muss auch die Summe

$$\sum_\tau p_\sigma(\tau)$$

gleich 1 sein, da sie der Wahrscheinlichkeit entspricht, dass auf das Symbol  $\sigma$  irgendein anderes Symbol folgt. Tatsächlich können wir diesen Zusammenhang mit wenigen Federstrichen ableiten. Es ist

$$\sum_\tau p_\sigma(\tau) \stackrel{(4.1)}{=} \sum_\tau \frac{p(\sigma, \tau)}{p(\sigma)} = \frac{1}{p(\sigma)} \sum_\tau p(\sigma, \tau) \stackrel{(4.2)}{=} \frac{p(\sigma)}{p(\sigma)} = 1$$

Als Nächstes wollen wir klären, ob unser statistisches Modell für die Beschreibung von Datenquellen geeignet ist, die natürlichsprachliche Texte emittieren. Konkret geht es dabei um die Frage, ob wir die

ABBABABABABABABBBBABB ...  
... BBBBABABABABBBACACA ...  
... BBABBBBBBABBABACBBBABA ...



$p(\sigma, \tau)$	$\tau = A$	$\tau = B$	$\tau = C$
$\sigma = A$	0	$\frac{4}{15}$	$\frac{1}{15}$
$\sigma = B$	$\frac{8}{27}$	$\frac{8}{27}$	0
$\sigma = C$	$\frac{1}{27}$	$\frac{4}{135}$	$\frac{1}{135}$

**Abb. 4.4:** Berechnung der Bigrammwahrscheinlichkeiten durch die statistische Analyse einer typischen Nachricht

$p(\sigma, \tau)$	$\tau = A$	$\tau = B$	$\tau = C$
$\sigma = A$	0	$\frac{4}{15}$	$\frac{1}{15}$
$\sigma = B$	$\frac{8}{27}$	$\frac{8}{27}$	0
$\sigma = C$	$\frac{1}{27}$	$\frac{4}{135}$	$\frac{1}{135}$

$$p_\sigma(\tau) = \frac{p(\sigma, \tau)}{\sum_\tau p(\sigma, \tau)}$$

$p_\sigma(\tau)$	$\tau = A$	$\tau = B$	$\tau = C$
$\sigma = A$	0	$\frac{4}{5}$	$\frac{1}{5}$
$\sigma = B$	$\frac{1}{2}$	$\frac{1}{2}$	0
$\sigma = C$	$\frac{1}{2}$	$\frac{2}{5}$	$\frac{1}{10}$

**Abb. 4.5:** Die zeilenweise Normierung der Tabelleneinträge führt zu den Transitionswahrscheinlichkeiten.

## Quelle 1



XFOML RXKHRJFFUJ ZLPWCFWK-CYJ FFJEYVKQSGXYD QPAAMKB-ZAACIBZLHJQD

## Quelle 2



OCRO HLI RGWR NMIELWIS EU LL  
NBNSESEBYA TH EEI ALHENHTTPA  
OOBTVA NAH BRL

## Quelle 3



ON IE ANTSOUTINYS ARE T INCTORE  
ST BE S DEAMY ACHIN D ILONASIVE  
TUCOWWE AT TEASONARE FUSO TI-  
ZIN ANDY TOBE SEACE CTISBE

## Quelle 4



IN NO IST LAT WHEY CRATICT FROU-  
RE BIRS GROCID PONDENOME OF  
DEMONSTURES OF THE REPTAGIN IS  
REGOACTIONA OF CRE

## Quelle 5



REPRESENTING AND SPEEDILY IS AN  
GOOD APT OR COME CAN DIFFERENT  
NATURAL HERE HE THE A IN CAME  
THE TO OF TO EXPERT GRAY COME  
TO FURNISHES THE LINE MESSAGE  
HAD BE THESE

## Quelle 6



THE HEAD AND IN FRONTAL AT-  
TACK ON AN ENGLISH WRITER THAT  
THE CHARACTER OF THIS POINT IS  
THEREFORE ANOTHER METHOD FOR  
THE LETTERS THAT THE TIME OF  
WHO EVER TOLD THE PROBLEM FOR  
AN UNEXPECTED

**Abb. 4.6:** Ergebnis von Shannons Approximationsexperiment [81]

statistischen Eigenschaften der natürlichen Sprache durch ein Modell annähern können, das auf der Wahrscheinlichkeitsverteilung von  $n$ -Grammen beruht.

In seiner Arbeit aus dem Jahr 1948 hat Shannon überzeugend dargelegt, dass diese Frage ruhigen Gewissens bejaht werden darf und wir bereits für geringe Werte von  $n$  gute Approximationen erreichen. Das Ergebnis seines Experiments ist in Abbildung 4.6 zusammengefasst. Dort sind die Anfänge von 6 Nachrichten zu sehen, die aus verschiedenen Datenquellen stammen. Die erste Datenquelle ist gedächtnislos und emittiert alle Buchstaben mit der gleichen Wahrscheinlichkeit (*zero-order approximation*). Die zweite Quelle ist ebenfalls gedächtnislos und emittiert die Zeichen mit den Wahrscheinlichkeiten, wie sie für englische Texte typisch sind (*first-order approximation*). Die dritte und die vierte Quelle sind gedächtnisbehaftet und emittieren die Symbole entsprechend den Bigrammwahrscheinlichkeiten bzw. Trigrammwahrscheinlichkeiten englischer Texte (*second-order approximation* bzw. *third-order approximation*). Die restlichen Quellen approximieren die englische Sprache auf der Wortebene. Während die fünfte Quelle die Wörter, entsprechend ihren Auftrittswahrscheinlichkeiten, unabhängig voneinander auswählt (*first-order word approximation*), basiert die sechste Quelle auf den Bigrammwahrscheinlichkeiten von Wortkombinationen (*second-order word approximation*).

Vor allem die Ausgabe der letzten Quelle macht deutlich, dass sich die natürliche Sprache auf dem eingeschlagenen Weg gut approximieren lässt. Shannon hat damit empirisch gezeigt, dass die statistischen Modelle geeignet sind, um praxisrelevante Anwendungsfälle zu beschreiben. Beachten Sie, dass dabei nur die Syntax der Sprache eine Rolle spielt und nicht die Semantik. Inhaltlich ergibt die Nachricht der sechsten Quelle genauso wenig einen Sinn wie die Nachrichten der anderen Quellen.

## Zustandsdiagramme

Eine Datenquelle bezeichnen wir in diesem Buch als *Markov-Quelle*, wenn sich ihr Emissionsverhalten mithilfe eines Zustandsdiagramms darstellen lässt. Welche Art von Diagrammen wir hierbei im Sinn haben, machen die in Abbildung 4.7 gezeigten Beispiele deutlich.

Die Zustände eines Diagramms bilden das Gedächtnis der Quelle und sind, abgesehen von den trivialen Diagrammen mit nur einem Zustand, mit Sequenzen von Quellsymbolen markiert. Diese Sequenzen repräsentieren die Symbole, die von der Quelle zuletzt emittiert wurden.

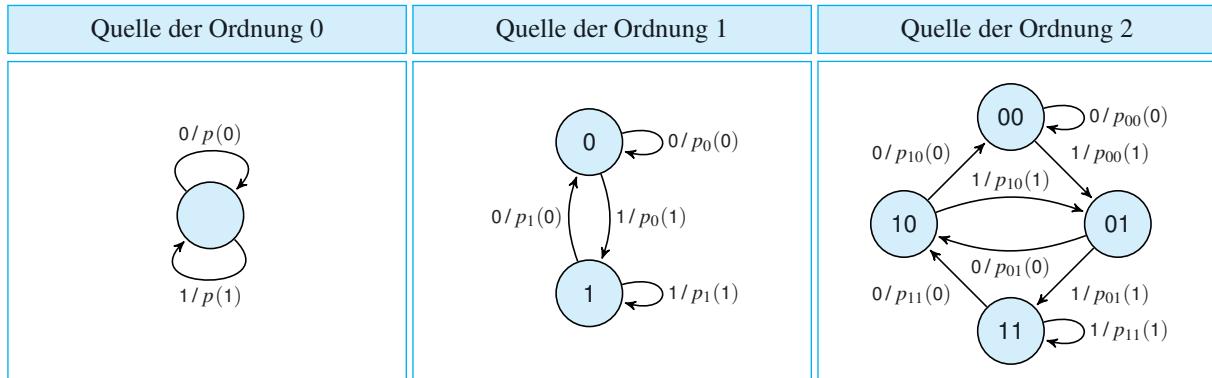


Abb. 4.7: Binäre Markov-Quellen verschiedener Ordnungen

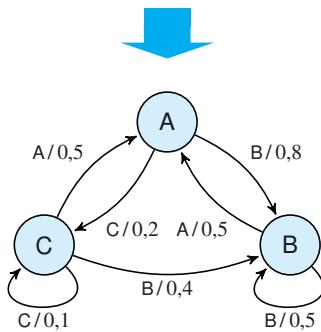
Das bedeutet, dass die mittlere und die rechte Beispielquelle in Abbildung 4.7 in der Lage sind, 1 bzw. 2 Symbole in die Vergangenheit zu blicken, und sich die Auftrittswahrscheinlichkeiten der Symbole entsprechend dieser Historie verändern können. Die linke Beispielquelle besitzt diese Flexibilität nicht. Sie verfügt über genau einen Zustand und ist damit nicht in der Lage, Symbole kontextabhängig zu emittieren: Sie ist das, was wir in diesem Buch als eine gedächtnislose Quelle bezeichnen.

Allgemein nennen wir eine Markov-Quelle, bei der die Auftrittswahrscheinlichkeit eines Symbols in Abhängigkeit der letzten  $n$  emittierten Symbole variiert, eine Quelle der Ordnung  $n$ . Gedächtnislose Quellen finden in dieser Nomenklatur auf natürliche Weise ihren Platz: Sie sind die Markov-Quellen der Ordnung 0.

Haben wir die Transitionswahrscheinlichkeiten einer Datenquelle in einer tabellarischen Darstellung vor uns, so können wir diese eins zu eins in ein Zustandsdiagramm übersetzen. Hierzu müssen wir lediglich für jedes Symbol einen Zustand erzeugen und für jeden Eintrag  $p_\sigma(\tau) > 0$  eine mit  $\tau/p_\sigma(\tau)$  markierte Kante von Zustand  $\sigma$  nach Zustand  $\tau$  führen. Übersetzen wir die Tabelle aus Abbildung 4.5 auf diese Weise, so erhalten wir das Zustandsdiagramm aus Abbildung 4.8.

An dieser Stelle wird klar, weshalb wir die bedingte Wahrscheinlichkeit  $p_\sigma(\tau)$  als Transitionswahrscheinlichkeit bezeichnen.  $p_\sigma(\tau)$  entspricht der Wahrscheinlichkeit, dass die Datenquelle, wenn sie sich im Zustand  $\sigma$  befindet, in den Zustand  $\tau$  übergeht.

$p_{\sigma}(\tau)$	$\tau = A$	$\tau = B$	$\tau = C$
$\sigma = A$	0	$\frac{4}{5}$	$\frac{1}{5}$
$\sigma = B$	$\frac{1}{2}$	$\frac{1}{2}$	0
$\sigma = C$	$\frac{1}{2}$	$\frac{2}{5}$	$\frac{1}{10}$



**Abb. 4.8:** Shannons Beispiel einer Markov-Quelle der Ordnung 1 [81]

## Ergodizität

An dieser Stelle wollen wir zurückblicken und unsere bisher erarbeiteten Ergebnisse auf den Prüfstand stellen. Wir haben dargelegt, dass sich eine Markov-Quelle aus einer hinreichend langen Nachricht modellieren lässt, indem zunächst die Bigrammwahrscheinlichkeiten bestimmt, danach die Transitionswahrscheinlichkeiten berechnet und diese am Ende in ein Zustandsdiagramm übersetzt werden. Es ist ein bedeutendes Merkmal dieser Vorgehensweise, dass wir für die Analyse eine einzige Nachricht verwendet haben und nicht etwa eine Schar vieler Nachrichten. Wir sind stillschweigend davon ausgegangen, dass eine einzige Nachricht ausreicht, um die statistischen Eigenschaften der Datenquelle zu erfassen. Zufallsprozesse mit dieser Eigenschaft tragen in der Wahrscheinlichkeitstheorie einen besonderen Namen: Sie heißen *ergodisch*.



### Definition 4.4 (Ergodizität)

Ein Zufallsprozess heißt *ergodisch*, wenn die statistischen Eigenschaften in jeder Stichprobe dieselben sind.

Modelliert jedes Zustandsdiagramm eine ergodische Datenquelle? Die Beispiele in Abbildung 4.9 machen deutlich, dass wir diese Frage verneinen müssen. Beide Beispiele wollen wir genauer ansehen:

- Das erste Zustandsdiagramm besitzt zwei isolierte Teilbereiche, die nach dem Betreten nicht mehr verlassen werden können. In unserem Beispiel führt dies dazu, dass die Datenquelle genau zwei Nachrichten emittieren kann: Die erste ist eine Folge von Cs und die zweite eine Folge von Bs. Die statistischen Eigenschaften beider Nachrichten sind verschieden, sodass die Datenquelle nicht ergodisch sein kann.
- Das zweite Zustandsdiagramm besitzt die Eigenschaft, dass jeder Zyklus, d. h. jeder Pfad durch den Graphen, der einen Zustand mit sich selbst verbindet, eine gerade Anzahl an Kanten durchläuft. Der Zustand A besitzt beispielsweise die folgenden Zyklen:

$A \rightarrow C \rightarrow A$	(☞ Zykluslänge 2)
$A \rightarrow B \rightarrow A$	(☞ Zykluslänge 2)
$A \rightarrow C \rightarrow A \rightarrow B \rightarrow A$	(☞ Zykluslänge 4)
$A \rightarrow B \rightarrow A \rightarrow C \rightarrow A$	(☞ Zykluslänge 4)
...	

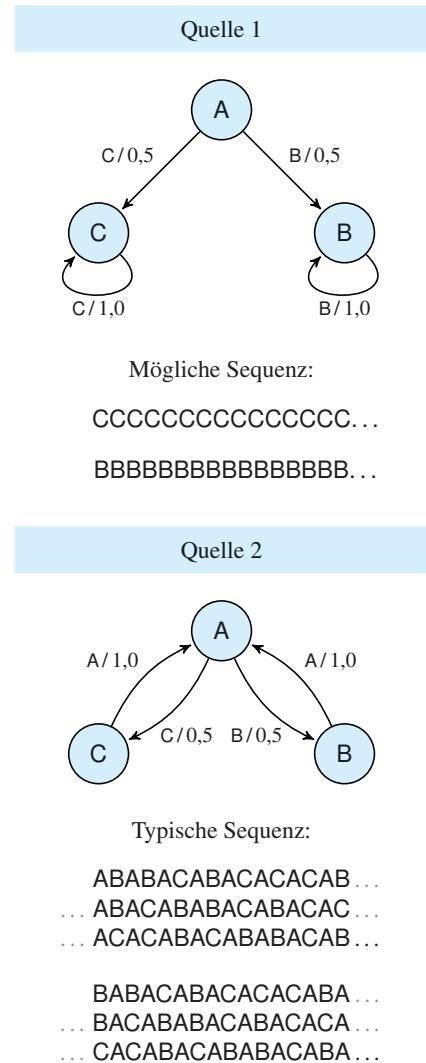
Dies führt dazu, dass die Symbole in bestimmten Mustern auftreten, die sich in Abhängigkeit des Startzustands verändern. Starten wir beispielsweise im Zustand C, so wird das erste emittierte Symbol ein A sein, genauso wie das dritte, fünfte, siebte und so fort. An diesen Positionen treffen wir mit der Wahrscheinlichkeit 1 auf das Symbol A. Starten wir dagegen im Zustand A, so wird eine Nachricht emittiert, die diese statistische Eigenschaft nicht besitzt. In der jetzt emittierten Nachricht treffen wir an den eben genannten Positionen mit der Wahrscheinlichkeit 0 auf das Symbol A. Damit ist auch diese Quelle nicht ergodisch, da nicht alle Stichproben die gleichen statistischen Eigenschaften besitzen.

Das Zustandsdiagramm aus Abbildung 4.8 besitzt die geschilderten Probleme nicht. Es existieren dort keine isolierten Teilbereiche im Sinne des ersten Beispiels, und die Zykluslängen weisen für keinen Zustand einen gemeinsamen Teiler  $> 1$  auf, wie es im zweiten Beispiel der Fall war. Es lässt sich zeigen, dass diese Eigenschaften sogar hinreichend sind, um eine Datenquelle als ergodisch zu identifizieren [81].

Im Rest dieses Buchs werden wir immer davon ausgehen, dass die betrachteten Datenquellen ergodisch sind. Dies versetzt uns in die bequeme Lage, aus jeder hinreichend langen Nachricht auf die statistischen Eigenschaften der Quelle schließen zu können.

## 4.3 Datenkompression

Mit dem Wissen, das wir im letzten Abschnitt erworben haben, sind wir gut gerüstet, um uns dem eigentlichen Gegenstand dieses Kapitels zu stellen: der *Quellencodierung*. Bevor wir uns in die Details der verschiedenen Codierungsverfahren vertiefen, wollen wir zunächst den Grundgedanken offenlegen, den alle Algorithmen, so unterschiedlich sie auch sind, miteinander teilen. Abstrakt gesprochen wollen wir mit der Quellencodierung eine möglichst gute Auslastung des Übertragungskanals erreichen, d. h., wir wollen Information mit einer Geschwindigkeit übertragen, die sich am Limit der Kanalkapazität bewegt. Dies gelingt durch die Minimierung der Redundanz oder, was dasselbe ist, durch die Maximierung der Information. Damit können wir den Grundgedanken folgendermaßen auf den Punkt bringen:



**Abb. 4.9:** Nichtergodische Quellen



Die Quellencodierung verfolgt das Ziel, die zu übertragende Nachricht in einer Form an den Sender zu übergeben, die einen möglichst hohen Informationsgehalt aufweist.

Damit ist klar: Die Algorithmen der Quellencodierung sind allesamt Algorithmen aus dem Bereich der *Datenkompression*. Auf der obersten Ebene lassen sie sich in zwei verschiedene Klassen einteilen (Abbildung 4.10):

#### ■ Verlustfreie Kompressionsverfahren

Eine Codierung heißt verlustfrei, wenn sich aus der codierten Zeichensequenz die Originalnachricht unverändert wiederherstellen lässt. In Definition 3.1 haben wir explizit gefordert, dass eine Codierung injektiv sein muss, zwei unterschiedliche Nachrichten also immer auch unterschiedlich codiert werden. Das bedeutet, dass wir in diesem Buch ausschließlich verlustfreie Codierungen betrachten.

#### ■ Verlustbehaftete Kompressionsverfahren

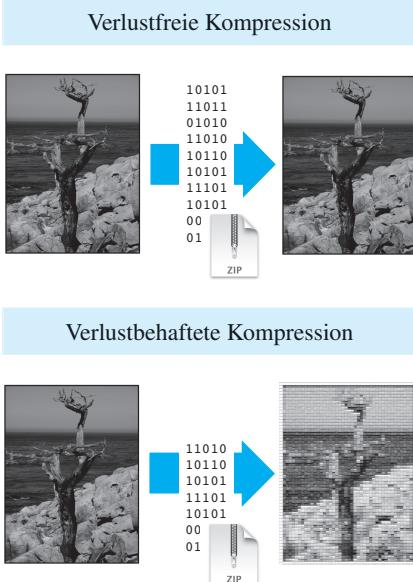
Wird auf die Forderung der Injektivität verzichtet, so umfasst der Codierungsbegriff auch solche Abbildungen, die unterschiedlichen Nachrichten manchmal die gleiche Sequenz von Codesymbolen zuordnen. Mit solchen Codierungen lassen sich hohe Kompressionsraten erreichen, allerdings ist die Originalnachricht nicht mehr in jedem Fall eindeutig rekonstruierbar.

Eingesetzt werden verlustbehaftete Codierungen vor allem im Audio- und Videobereich. Hier werden bewusst jene Signalelemente entfernt, die das menschliche Hör- oder Sehorgan nicht wahrnehmen kann.

Im Bereich der verlustfreien Kompressionsverfahren lassen sich zwei wichtige Untergruppen identifizieren (Abbildung 4.11):

#### ■ Entropiecodierungen

Die Algorithmen aus dieser Gruppe verfolgen die Idee, eine Nachricht mit einer längenvariablen Codierung zu komprimieren, die häufig vorkommende Zeichen mit kurzen und selten vorkommende Zeichen mit langen Codewörtern belegt. Der wichtigste Vertreter dieser Gruppe ist die Huffman-Codierung, die wir zusammen mit ihren Vorläufern in Abschnitt 4.4 besprechen.



**Abb. 4.10:** Verlustfreie Kompressionsverfahren können die ursprüngliche Nachricht originalgetreu wiederherstellen, erreichen dafür aber geringere Kompressionsraten. Verlustbehaftete Kompressionen werden vor allem im Audio- und Videobereich eingesetzt. Sie verzichten auf die Speicherung von akustischen oder visuellen Details, die unsere Sinne nicht wahrnehmen können.

Weiterentwicklungen der Huffman-Codierung sind beispielsweise die arithmetische Codierung und die ANS-Codierung, die Sie in den Abschnitten 4.5 und 4.6 kennenlernen werden. Auch wenn sie nicht mehr zeichenweise arbeiten, und damit streng genommen nicht mehr in die Gruppe der Entropiecodierungen fallen, basieren sie auf der gleichen Idee, die Codewörter aus den Auftrittswahrscheinlichkeiten der Quellsymbole abzuleiten.

### ■ Substitutionscodierungen

Die Algorithmen aus dieser Gruppe komprimieren eine Nachricht, indem Teilsequenzen durch Referenzen in ein *Verzeichnis* oder *Wörterbuch (dictionary)* dargestellt werden. Substitutionscodierungen lassen sich in statische und dynamische Verfahren aufteilen. Statistische Verfahren verwenden ein Verzeichnis, das vorab festgeschrieben wird und während der Codierung unverändert bleibt. Dynamische Verfahren erstellen das Verzeichnis während der Codierung und passen sich damit individuell an die zu codierende Nachricht an.

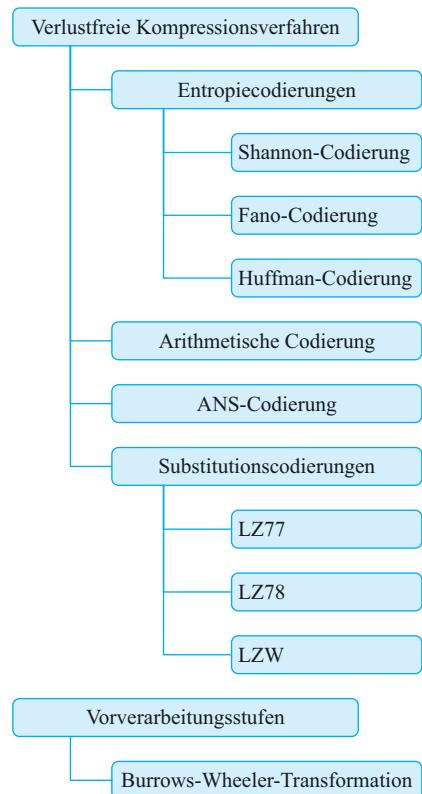
Zu den bekanntesten Substitutionscodierungen gehören die Verfahren LZ77, LZ78 und LZW, die von Abraham Lempel, Jacob Ziv und Terry Welch entwickelt wurden. Alle drei Verfahren sind Gegenstand von Abschnitt 4.7.

Viele Kompressionsalgorithmen, die in der Praxis eine Rolle spielen, sind Mischformen der genannten Verfahren, die zusätzlich mit Vor- und Nachverarbeitungsstufen kombiniert werden. Beispielsweise verwenden viele Substitutionscodierungen einen nachgeschalteten Huffman-Encoder, um die Wörterbuchreferenzen kompakt darzustellen.

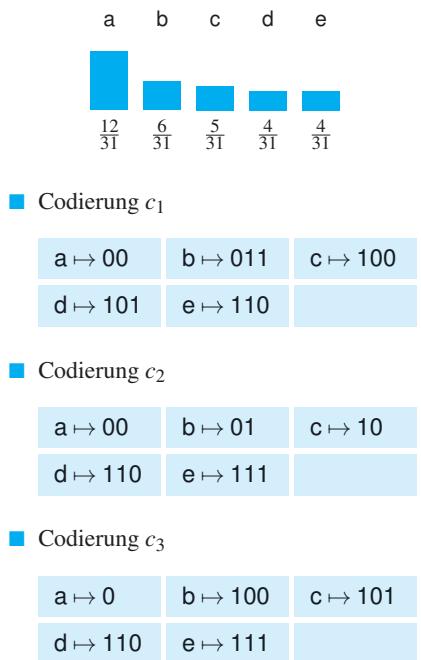
Vorverarbeitungsstufen haben in erster Linie die Aufgabe, die Nachrichten in eine für das gewählte Kompressionsverfahren passende Form zu bringen. Ein trickreicher Algorithmus aus diesem Bereich ist die *Burrows-Wheeler-Transformation*, die wir detailliert in Abschnitt 4.8 besprechen.

## Bewertung von Kompressionsverfahren

Bevor wir die verschiedenen Kompressionsalgorithmen im Detail betrachten, wollen wir uns Gedanken darüber machen, wie sich deren Güte messen lässt. Wird eine Computerdatei z. B. mit dem Programm **zip** komprimiert, so wird als Gütemaß gerne der Begriff der *Kompressionsrate* benutzt. Gemeint ist damit das Größenverhältnis zwischen der



**Abb. 4.11:** Verlustfreie Kompressionsverfahren in der Übersicht



**Abb. 4.12:** Entropiecodierungen

komprimierten und der nicht komprimierten Datei:

$$\text{Kompressionsrate } L = \frac{\left| \begin{array}{c} \text{ZIP} \\ \hline \text{DOC} \end{array} \right|}{\left| \begin{array}{c} \text{DOC} \\ \hline \text{DOC} \end{array} \right|} \quad (4.5)$$

Ein Quotient von 0,5 drückt beispielsweise aus, dass die komprimierte Datei nur noch halb so groß ist wie die ursprüngliche.

Die Berechnungsvorschrift (4.5) wollen wir noch ein wenig formaler erfassen. Ist  $u$  eine Nachricht, die von einer Datenquelle emittiert wird, so ist  $|u|$  die Anzahl der darin enthaltenen Symbole und  $|c(u)|$  die Anzahl der Symbole, aus denen die komprimierte Nachricht besteht. Die rechte Seite von (4.5) wird dann zu

$$\frac{|c(\mathbf{u})|}{|\mathbf{u}|} \quad (4.6)$$

Für die meisten Datenquellen und Codierungen, die in der Praxis eine Rolle spielen, wird sich dieser Quotient stabilisieren, wenn wir für  $u$  immer längere Nachrichten wählen und dabei auf die korrekte statistische Verteilung der Symbole achten.

Das bedeutet, dass wir den Quotienten

$$L = \lim_{|\mathbf{u}| \rightarrow \infty} \frac{|c(\mathbf{u})|}{|\mathbf{u}|} \quad (4.7)$$

als Maßzahl für die Kompressionsgüte heranziehen können.

Im Falle von Entropiecodierungen können wir das Gesagte noch weiter präzisieren. Diese Codierungen arbeiten zeichenweise und versuchen, eine Nachricht dadurch zu komprimieren, dass sie häufig vorkommende Symbole auf kurze Codewörter und selten vorkommende Symbole auf lange Codewörter abbilden. Drei Codierungen, die diesen Ansatz verfolgen, sind in Abbildung 4.12 zu sehen.

Für alle drei Codierungen können wir die Kompressionsrate approximieren, indem wir eine typische Nachricht wählen und den Quotienten (4.6) für immer längere Anfangsstücke ausrechnen. Für die Datenquelle, die ihre Symbole mit den in Abbildung 4.12 gezeigten Wahrscheinlichkeiten emittiert, könnte eine typische Nachricht beispielsweise so beginnen:

baecaadceabadaacebbacaceadbaabd ...

Diese Nachricht führt zu der folgenden Approximation:

### ■ Codierung $c_1$

011	 $L = \frac{3}{1} = 3,00$
01100	 $L = \frac{5}{2} = 2,50$
01100110	 $L = \frac{8}{3} \approx 2,67$
01100110100	 $L = \frac{11}{4} = 2,75$
...	
0110011010000001011001100001100 ... 101	 $L = \frac{81}{31} \approx 2,61$

### ■ Codierung $c_2$

01	 $L = \frac{2}{1} = 2,00$
0100	 $L = \frac{4}{2} = 2,00$
0100111	 $L = \frac{7}{3} \approx 2,33$
010011110	 $L = \frac{9}{4} = 2,25$
...	
0100111100000110101110001001100 ... 110	 $L = \frac{70}{31} \approx 2,26$

### ■ Codierung $c_3$

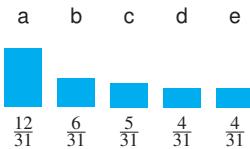
100	 $L = \frac{3}{1} = 3,00$
1000	 $L = \frac{4}{2} = 2,00$
1000111	 $L = \frac{7}{3} \approx 2,33$
1000111101	 $L = \frac{10}{4} = 2,50$
...	
1000111101001101011110100011000 ... 110	 $L = \frac{69}{31} \approx 2,23$

Die Beispiele machen klar, welche Bedeutung dem Grenzwert (4.7) hier zukommt. Er gibt an, wie viele Symbole im Mittel für die Codierung eines Symbols der Originalnachricht verwendet werden.

Sind für jedes Symbol  $\sigma_i$  die Auftrittswahrscheinlichkeit  $p_i$  und die Codewortlänge  $l_i$  bekannt, so können wir die *mittlere Codewortlänge* direkt ausrechnen. Sie entspricht dann der gewichteten Summe

$$p_1 l_1 + p_2 l_2 + \dots + p_n l_n$$

Diesen Zusammenhang benutzen wir als Legitimation für die folgende, formale Definition des Begriffs der *mittleren Codewortlänge*.



### Codierung $c_1$

$a \mapsto 00$	$b \mapsto 011$	$c \mapsto 100$
$d \mapsto 101$	$e \mapsto 110$	

$$\begin{aligned}
 L &= 2 \cdot \frac{12}{31} + 3 \cdot \frac{6}{31} + 3 \cdot \frac{5}{31} + 3 \cdot \frac{4}{31} + 3 \cdot \frac{4}{31} \\
 &= \frac{24}{31} + \frac{18}{31} + \frac{15}{31} + \frac{12}{31} + \frac{12}{31} \\
 &= \frac{81}{31} \approx 2,61290
 \end{aligned}$$

### Codierung $c_2$

$a \mapsto 00$	$b \mapsto 01$	$c \mapsto 10$
$d \mapsto 110$	$e \mapsto 111$	

$$\begin{aligned}
 L &= 2 \cdot \frac{12}{31} + 2 \cdot \frac{6}{31} + 2 \cdot \frac{5}{31} + 3 \cdot \frac{4}{31} + 3 \cdot \frac{4}{31} \\
 &= \frac{24}{31} + \frac{12}{31} + \frac{10}{31} + \frac{12}{31} + \frac{12}{31} \\
 &= \frac{70}{31} \approx 2,25806
 \end{aligned}$$

### Codierung $c_3$

$a \mapsto 0$	$b \mapsto 100$	$c \mapsto 101$
$d \mapsto 110$	$e \mapsto 111$	

$$\begin{aligned}
 L &= 1 \cdot \frac{12}{31} + 3 \cdot \frac{6}{31} + 3 \cdot \frac{5}{31} + 3 \cdot \frac{4}{31} + 3 \cdot \frac{4}{31} \\
 &= \frac{12}{31} + \frac{18}{31} + \frac{15}{31} + \frac{12}{31} + \frac{12}{31} \\
 &= \frac{69}{31} \approx 2,22581
 \end{aligned}$$

Abb. 4.13: Mittlere Codewortlängen



### Definition 4.5

Es sei  $X$  eine Datenquelle, die ihre Symbole  $\sigma_1, \dots, \sigma_n$  mit den Wahrscheinlichkeiten  $p_1, \dots, p_n$  emittiert. Die gewichtete Summe

$$L(X, c) := \sum_{i=1}^n p_i l_i$$

heißt *mittlere Codewortlänge*.  $l_i$  ist die Länge des Codeworts  $c(\sigma_i)$ .

Geht aus dem Kontext hervor, welche Quelle  $X$  bzw. welche Codierung  $c$  gemeint ist, so schreiben wir anstatt  $L(X, c)$  nur noch  $L(c)$ ,  $L(X)$  oder schlicht  $L$ .

In Abbildung 4.13 sind die exakten Werte von  $L$  für unsere drei Beispielcodierungen zu sehen. Die geringste mittlere Codewortlänge weist die Codierung  $c_3$  auf; sie wird eine Nachricht, in denen die Symbole mit den angegebenen Wahrscheinlichkeiten vorkommen, am stärksten komprimieren.

Nachdem wir mit der mittleren Codewortlänge eine Maßzahl an die Hand bekommen haben, mit der sich die Güte einer Entropiecodierung quantitativ messen lässt, sind wir gut gewappnet, um die nächsten Abschnitte zu meistern. Dort werden wir mehrere Verfahren betrachten, mit denen sich eine Entropiecodierung für eine vorgegebene Datenquelle systematisch konstruieren lässt.

## 4.4 Entropiecodierungen

### 4.4.1 Shannon-Codierung

Eines der ersten Verfahren zur Erzeugung längenvariabler Codes ist in der Arbeit von Claude Shannon aus dem Jahr 1948 beschrieben. Wir wollen seiner historischen Bedeutung Rechnung tragen und uns zunächst den Originalwortlaut ansehen:

„Arrange the messages of length  $N$  in order of decreasing probability and suppose their probabilities are  $p_1 \geq p_2 \geq p_3 \dots \geq p_n$ . Let  $P_s = \sum_1^{s-1} p_i$ ; that is  $P_s$  is the cumulative

■ Beispiel 1		$i$	$p_i$	$\log_2 \frac{1}{p_i}$	$\lceil \log_2 \frac{1}{p_i} \rceil$	$P_i := \sum_{j=1}^{i-1} p_j$	Codewort
a	$\frac{1}{2}$	a	$\frac{1}{2}$	1	1	$0 = 0, \underline{0}00000\dots$	0
b	$\frac{1}{8}$	b	$\frac{1}{8}$	3	3	$\frac{1}{2} = 0, \underline{1}00000\dots$	100
c	$\frac{1}{8}$	c	$\frac{1}{8}$	3	3	$\frac{5}{8} = 0, \underline{10}1000\dots$	101
d	$\frac{1}{8}$	d	$\frac{1}{8}$	3	3	$\frac{6}{8} = 0, \underline{11}0000\dots$	110
e	$\frac{1}{8}$	e	$\frac{1}{8}$	3	3	$\frac{7}{8} = 0, \underline{111}000\dots$	111

■ Beispiel 2		$i$	$p_i$	$\log_2 \frac{1}{p_i}$	$\lceil \log_2 \frac{1}{p_i} \rceil$	$P_i := \sum_{j=1}^{i-1} p_j$	Codewort
a	$\frac{12}{31}$	a	$\frac{12}{31}$	1,37	2	$0 = 0, \underline{00}0000\dots$	00
b	$\frac{6}{31}$	b	$\frac{6}{31}$	2,37	3	$\frac{12}{31} = 0, \underline{01}1000\dots$	011
c	$\frac{5}{31}$	c	$\frac{5}{31}$	2,63	3	$\frac{18}{31} = 0, \underline{10}101\dots$	100
d	$\frac{4}{31}$	d	$\frac{4}{31}$	2,95	3	$\frac{23}{31} = 0, \underline{101}111\dots$	101
e	$\frac{4}{31}$	e	$\frac{4}{31}$	2,95	3	$\frac{27}{31} = 0, \underline{110}111\dots$	110

Abb. 4.14: Shannon-Codierung in Aktion

probability up to, but not including,  $p_s$ . We first encode into a binary system. The binary code for message  $s$  is obtained by expanding  $P_s$  as a binary number. The expansion is carried out to  $m_s$  places, where  $m_s$  is the integer satisfying:

$$\log_2 \frac{1}{p_s} \leq m_s < 1 + \log_2 \frac{1}{p_s}.$$

Claude Shannon [81]

Zunächst fällt auf, dass Shannon ganz allgemein von Nachrichten der Länge  $N$  („messages of length  $N$ “) und nicht von Symbolen spricht. Um jedes einzelne Symbol mit einem bestimmten Codewort zu belegen, brauchen wir aber lediglich  $N = 1$  zu setzen. In diesem Fall entsprechen  $p_1, \dots, p_n$  den Wahrscheinlichkeiten, mit denen die Datenquelle ihre Symbole emittiert, und wir erhalten das, wonach wir suchen: ein Verfahren für die Konstruktion präfixfreier längenvariabler Codes.

Abbildung 4.14 demonstriert das Verfahren am Beispiel zweier Datenquellen mit dem Symbolalphabet  $\Sigma = \{a, b, c, d, e\}$ . Um die Codewörter

abzuleiten, können wir uns eins zu eins an Shannons verbaler Beschreibung orientieren:

- Im ersten Schritt müssen die Symbole der Datenquelle nach absteigender Wahrscheinlichkeit sortiert werden. In unseren Beispielen wurde die Reihenfolge bereits passend gewählt, sodass hier nichts zu tun ist.
- Im zweiten Schritt werden die kumulierten Summen  $P_i$  nach dem folgenden Schema berechnet:

$$\begin{aligned} P_1 &= 0 \\ P_{i+1} &= P_i + p_i \end{aligned} \quad (4.8)$$

Beachten Sie dabei, dass die Wahrscheinlichkeit  $p_i$  in der Summe  $P_{i+1}$ , aber nicht in der Summe  $P_i$  enthalten ist.

- Im dritten Schritt werden die Codewörter abgeleitet. Wir erhalten das  $i$ -te Codewort, indem wir die ersten  $m_i$  Nachkommabits der kumulierten Summe  $P_i$  ablesen. Die Codewortlänge  $m_i$  ist dabei eindeutig bestimmt: Sie ist der Wert  $\log_2 \frac{1}{p_i}$ , aufgerundet zu der nächstgrößeren natürlichen Zahl  $\lceil \log_2 \frac{1}{p_i} \rceil$ .

Ein Blick auf die Beispiele macht klar, dass die Shannon-Codierung unserer Erwartung gerecht wird: Häufiger auftretende Symbole werden mit kürzeren und seltener auftretende Symbole mit längeren Codewörtern belegt. Hierfür verantwortlich ist die Berechnungsvorschrift für  $m_i$ . Wird die Auftrittswahrscheinlichkeit  $p_i$  größer, so verkleinert sich der Wert  $\log_2 \frac{1}{p_i}$  und damit auch die Anzahl der Bits, aus denen das Codewort gebildet wird.

Shannons Verfahren produziert für unsere Beispielquellen jeweils eine präfixfreie Codierung. Die zweite kennen wir bereits. Sie ist die Codierung  $c_1$ , die wir in Abbildung 4.12 als Beispiel verwendet haben.

Wir wollen uns Gewissheit darüber verschaffen, dass die Präfixfreiheit kein Zufall war, und Shannons Verfahren in jedem erdenklichen Fall einen präfixfreien längenvariablen Code produziert. Hierzu müssen wir zeigen, dass kein Codewort der Anfang eines anderen ist.

Wir können hier ganz ähnlich argumentieren, wie auf Seite 176 im Beweis von Satz 3.1. Dort haben wir die Richtung von rechts nach links durch die Angabe eines Code-Konstruktionsverfahrens bewiesen, dem die gleiche Kernidee zugrunde liegt.

Zunächst folgt aus (4.8) die Beziehung

$$P_{i+j} = P_{i+j-1} + p_{i+j-1} \geq P_{i+j-1} + p_{i+j} \geq P_i + p_{i+j} \quad (1 \leq j \leq n-i)$$

die wir folgendermaßen umschreiben können:

$$P_{i+j} - P_i \geq p_{i+j} \quad (4.9)$$

Wir nehmen jetzt an, der erzeugte Code sei nicht präfixfrei. Dann existieren zwei Summen  $P_i$  und  $P_{i+j}$ , sodass  $P_{i+j}$  in den ersten  $m_{i+j}$  Nachkommastellen mit  $P_i$  übereinstimmt. Wegen

$$\sum_i p_i = 1$$

sind die Vorkommaanteile von  $P_i$  und  $P_{i+j}$  beide gleich 0. Das bedeutet, dass  $P_i$  und  $P_{i+j}$  so nahe beieinander liegen, dass die Differenz weniger als  $2^{-m_{i+j}}$  beträgt:

$$P_{i+j} - P_i < 2^{-m_{i+j}} \quad (4.10)$$

Wegen

$$2^{-m_{i+j}} \leq 2^{-\log_2 \frac{1}{p_{i+j}}} = 2^{\log_2 p_{i+j}} = p_{i+j}$$

können wir (4.10) folgendermaßen umschreiben:

$$P_{i+j} - P_i < p_{i+j} \quad (4.11)$$

Dies steht im Widerspruch zu (4.9), sodass wir die Annahme, der Code sei nicht präfixfrei, verwerfen müssen.

## 4.4.2 Fano-Codierung

Auf eine ähnliche Art der Codierung war der MIT-Professor Robert M. Fano gestoßen [32]. Sein 1949 publiziertes Verfahren war Shannon bereits bekannt und wird in dessen Arbeit aus dem Jahr 1948 mit den folgenden Worten zusammengefasst:

*„[Fanos] method is to arrange the messages of length N in order of decreasing probability. Divide this series into two groups of as nearly equal probability as possible. If the message is in the first group its first binary digit will be 0, otherwise 1. The groups are similarly divided into subsets of nearly equal probability and the particular subset determines the second binary digit. This process is continued until each subset contains only one message.“*

Claude Shannon [81]

■ Beispiel 1

	a	b	c	d	e	
	$\frac{1}{2}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	

a	$\frac{1}{2}$	0		0
b	$\frac{1}{8}$	1	0	0
c	$\frac{1}{8}$	1	0	1
d	$\frac{1}{8}$	1	1	0
e	$\frac{1}{8}$	1	1	1

■ Beispiel 2

	a	b	c	d	e	
	$\frac{12}{31}$	$\frac{6}{31}$	$\frac{5}{31}$	$\frac{4}{31}$	$\frac{4}{31}$	

a	$\frac{12}{31}$	0	0		00	
b	$\frac{6}{31}$	0	1		01	
c	$\frac{5}{31}$	1	0		10	
d	$\frac{4}{31}$	1	1	0		110
e	$\frac{4}{31}$	1	1	1		111

Abb. 4.15: Fano-Codierung

Abbildung 4.15 demonstriert die Fano-Codierung an den beiden Beispielen aus dem vorangegangenen Abschnitt. Genau wie bei der Shannon-Codierung werden vorab die Symbole entsprechend ihren Auftrittswahrscheinlichkeiten sortiert. Danach folgen mehrere Partitionierungsschritte. In jedem Schritt werden die Symbole so durch eine Trennlinie in zwei Partitionen eingeteilt, dass die Summe der Wahrscheinlichkeiten möglichst ausgewogen ist. Im ersten Beispiel gelingt dies perfekt, da die Summen in beiden Partitionen zu jedem Zeitpunkt übereinstimmen. Im zweiten Beispiel ist dies nicht mehr der Fall. Dort müssen wir mit einer gewissen Unausgewogenheit der Wahrscheinlichkeitssummen leben.

Die Aufteilung wird so lange wiederholt, bis nur noch Partitionen mit einem einzigen Symbol übrig sind. Danach können wir die Codewörter ablesen, indem wir für jedes Symbol nachschauen, welchen Partitionen es jeweils zugeordnet wurde, und für die obere Partition eine 0 und für die untere Partition eine 1 notieren. Auf diese Weise trägt jeder Partitionierungsschritt genau ein Bit zum Codewort eines Symbols bei.

Die folgenden beiden Eigenschaften der Fano-Codierung sind leicht einzusehen:

- Zu jeder Zeit sind die Bitfolgen in den verschiedenen Partitionen paarweise verschieden. Das bedeutet, dass ein Codewort niemals der Anfang eines anderen Codeworts sein kann und die Fano-Codierung folglich immer einen präfixfreien Code erzeugt.
- Der Algorithmus sorgt dafür, dass die Symbole mit hohen Auftrittswahrscheinlichkeiten kleine Partitionen und die Symbole mit niedrigen Auftrittswahrscheinlichkeiten große Partitionen bilden. Je kleiner eine Partition ist, desto schneller terminiert das Verfahren. Diese Steuerungseigenschaft sorgt bei der Fano-Codierung dafür, dass häufiger auftretende Symbole mit kürzeren und seltener auftretende Symbole mit längeren Codewörtern versehen werden. Einen ganz ähnlichen Steuerungsmechanismus hatten wir bereits bei der Shannon-Codierung erkannt.

Vergleichen wir die Ergebnisse mit jenen aus dem vorherigen Abschnitt, so können wir zwei interessante Eigenschaften beobachten. Einerseits erzeugen beide Codierungsverfahren für das erste Beispiel exakt den gleichen Code. Andererseits macht das zweite Beispiel deutlich, dass dies nicht immer der Fall ist. Für diese liefert die Fano-Codierung tatsächlich ein besseres Ergebnis als die Shannon-Codierung: Fanos Algorithmus hat die Codierung  $c_2$  aus Abbildung 4.12 hervorgebracht, die

mit  $L \approx 2,258$  eine geringere mittlere Codewortlänge aufweist als die Codierung  $c_1$ , die Shannons Algorithmus produziert.

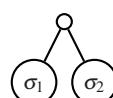
Aus Abschnitt 4.3 wissen wir bereits, dass auch die Codierung  $c_2$  nicht optimal ist. Dort hatten wir mit  $c_3$  eine Codierung angegeben, die eine noch geringere mittlere Codewortlänge aufweist. Damit lässt sich an dieser Stelle ein gemischtes Fazit ziehen. Auf der positiven Seite können wir mit der Fano-Codierung ein Verfahren unser Eigen nennen, mit dem sich präfixfreie, längenvariable Codes auf einfache Weise erzeugen lassen. Auf der negativen Seite müssen wir verbuchen, dass die Fano-Codierung zwar in vielen, aber nicht in allen Fällen optimale Ergebnisse liefert.

Diese Beobachtung wirft die Frage auf, ob ein systematisches Verfahren existiert, das *immer* einen optimalen präfixfreien Code erzeugt. Im nächsten Abschnitt werden wir diese Frage bejahen. Am Ende werden wir mit der *Huffman-Codierung* einen Algorithmus in Händen halten, der in allen Fällen ein optimales Ergebnis liefert und genauso einfach auszuführen ist wie die gerade vorgestellte Fano-Codierung.

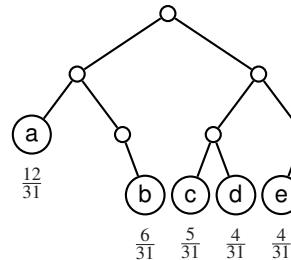
#### 4.4.3 Huffman-Codierung

Wir werden in diesem Abschnitt ein Verfahren entwickeln, das für jede gedächtnislose Quelle einen optimalen längenvariablen Code, d. h. einen Code mit minimaler durchschnittlicher Codewortlänge, erzeugt. Zum Erfolg wird uns die simple Tatsache verhelfen, dass sich jede Codierung in Form eines Codebaums beschreiben lässt. Wie die Bäume der Codes aussehen, die wir im vorherigen Abschnitt als Beispiele verwendet haben, zeigt Abbildung 4.16. Die Pfade von der Wurzel zu den Blättern definieren die Codewörter, wobei eine Kante nach links einer 0 und eine Kante nach rechts einer 1 entspricht. Anhand der Baumdarstellung können wir durch bloßes Hinschauen erkennen, dass die abgebildeten Codes präfixfrei sind: Sie besitzen diese Eigenschaft, weil kein innerer Knoten mit einem Quellsymbol markiert ist.

Wir wollen nun erarbeiten, wie sich der Codebaum eines optimalen Codes direkt konstruieren lässt. Zunächst betrachten wir den Fall für eine Datenquelle mit zwei Symbolen  $\sigma_1$  und  $\sigma_2$  und den Auftrittswahrscheinlichkeiten  $p_1$  und  $p_2$ . Für eine solche Datenquelle können wir den Baum eines optimalen Codes ohne nachzudenken hinschreiben:

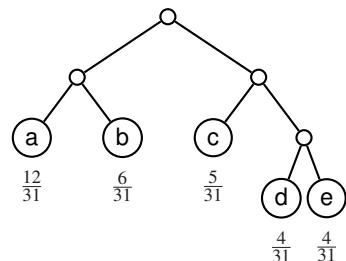


#### Shannon-Codierung



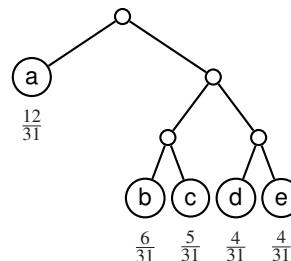
a ↠ 00	b ↠ 011	c ↠ 100
d ↠ 101	e ↠ 110	

#### Fano-Codierung



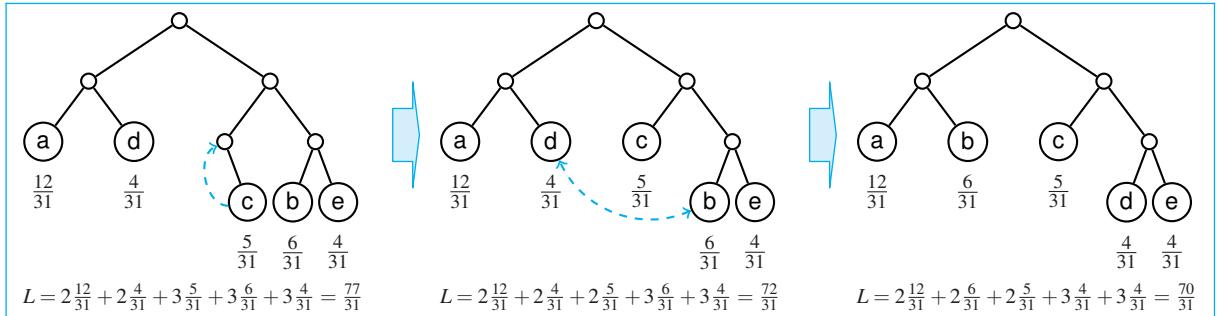
a ↠ 00	b ↠ 01	c ↠ 10
d ↠ 110	e ↠ 111	

#### Optimal



a ↠ 0	b ↠ 100	c ↠ 101
d ↠ 110	e ↠ 111	

Abb. 4.16: Baumdarstellung der Entropiecodierungen aus Abbildung 4.12



**Abb. 4.17:** Ist ein Baum nicht saturiert (links) oder sind die Blätter seiner längsten Pfade nicht mit den am seltensten vorkommenden Symbolen markiert (Mitte), so kann er keinen optimalen Code repräsentieren. In diesen Fällen können wir durch geringfügige Modifikationen einen Code erzeugen, der eine geringere mittlere Codewortlänge aufweist.

In den nun folgenden Überlegungen stehen  $\sigma_1, \sigma_2, \dots, \sigma_n$  für die Symbole einer Datenquelle  $X$  und  $p_1, p_2, \dots, p_n$  für die Wahrscheinlichkeiten, mit denen diese Symbole emittiert werden. Wir nehmen an, die Wahrscheinlichkeiten seien monoton fallend sortiert, es ist also

$$p_1 \geq p_2 \geq \dots \geq p_n$$

Ferner sei  $l_i$  die Länge des Codeworts, das unser Verfahren dem Symbol  $\sigma_i$  zuordnet. Unser Ziel ist es, einen optimalen präfixfreien Code zu erzeugen, also einen Code, der die mittlere Codewortlänge

$$L = p_1 l_1 + p_2 l_2 + \dots + p_{n-1} l_{n-1} + p_n l_n \quad (4.12)$$

minimiert.

Unser Verfahren basiert auf zwei elementaren Eigenschaften, die der Codebaum eines optimalen Codes auf jeden Fall erfüllen muss:

- Der Codebaum eines optimalen Codes ist ein *saturierter* Binärbaum, d.h., jeder innere Knoten besitzt zwei Nachfolger. Warum dies so sein muss, ist leicht einzusehen: Gäbe es einen inneren Knoten mit nur einem Nachfolger, so ließe sich durch die Elimination dieses Knotens sofort ein Code mit einer geringeren mittleren Codewortlänge erzeugen (Abbildung 4.17 links). Aus der Saturiertheit folgt, dass es mindestens zwei längste Pfade gibt.
- Betrachten wir die Symbole  $\sigma_{n-1}$  und  $\sigma_n$  mit den beiden geringsten Auftrittswahrscheinlichkeiten, so wissen wir, dass sie an den Blättern zweier längstmöglicher Pfade annotiert sein müssen. Wäre dies nicht der Fall (Abbildung 4.17 Mitte), so könnten wir die Symbole



Es ist anzunehmen, dass es Professor Robert Fano im Jahr 1951 nicht ganz ernst meinte, als er seinen Studenten am MIT die Aufgabe stellte, eine Strategie für die Konstruktion optimaler präfixfreier Codes zu erarbeiten. Fano verriet seinen Studenten nicht, dass dieses Problem damals noch ungelöst war und sich sowohl er als auch der große Claude Shannon vergeblich daran versucht hatten. Mit der Shannon-Codierung und der Fano-Codierung, die wir in den Abschnitten 4.4.1 und 4.4.2 besprochen haben, existierten zu jener Zeit lediglich Verfahren, die zwar einfach durchzuführen waren, aber nicht in jedem Fall eine optimale Lösung produzierten. Betrachten wir präfixfreie Codes im Lichte der Kraft'schen Ungleichung, so ist ein optimaler Code nichts anderes als die Lösung eines ganzzahligen Optimierungsproblems, und genau von dieser Sichtweise ließen sich Fano und Shannon leiten. Für gewöhnlich sind ganzzahlige Optimierungspro-

bleme schwer zu handhaben, und so rechneten weder Fano noch Shannon damit, dass ein einfaches Konstruktionsverfahren für optimale Codes überhaupt existieren würde. Einer von Fanos Studenten war David A. Huffman. Auch er konnte die gestellte Aufgabe nicht auf Anhieb lösen, doch die erfrischende Unbefangenheit, mit der er nach einer Antwort suchte, führte ihn schließlich zum Erfolg. Anders als sein Lehrer ging er das Problem nicht als Optimierungsproblem an. Er näherte sich von einer völlig anderen Seite und überlegte, welche strukturellen Eigenschaften die Codebäume präfixfreier Codierungen besitzen müssen. Huffman war kurz davor aufzugeben, als ihn der entscheidende Geistesblitz ereilte. Er entdeckte das nach ihm benannte Verfahren, mit dem sich optimale Codebäume bottom-up aufbauen lassen. In [88] wird er mit den folgenden Worten zitiert: „*It was the most singular moment of my life. [...] There was the absolute lightning of sudden realization.*“

an den Blättern austauschen und kämen so zu einem besseren Code (Abbildung 4.17 rechts).

Mit dem erarbeiteten Wissen können wir einen optimalen Codebaum bereits ein Stück weit aufbauen. Wir wissen aus dem eben Gesagten, dass er das folgende Fragment enthalten muss:



Wir wollen versuchen, mehr über den Rest des Codebaums herauszubekommen. In Abbildung 4.18 ist dies derjenige Teil, der als Dreieck symbolisiert ist. Bezeichnen wir die Länge des Pfads, der die Wurzel mit dem Vaterknoten von  $\sigma_{n-1}$  und  $\sigma_n$  verbindet, mit  $l'$ , so können wir Gleichung (4.12) wie folgt umschreiben:

$$\begin{aligned} L &= p_1 l_1 + p_2 l_2 + \dots + p_{n-1} (l' + 1) + p_n (l' + 1) \\ &= p_1 l_1 + p_2 l_2 + \dots + (p_{n-1} + p_n) l' + p_{n-1} + p_n \end{aligned}$$

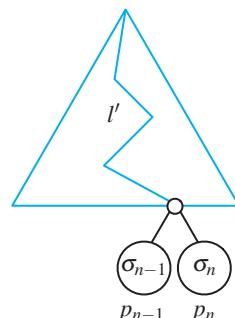
Es gilt also

$$L = L' + (p_{n-1} + p_n) \quad (4.14)$$

mit

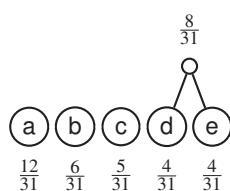
$$L' := p_1 l_1 + p_2 l_2 + \dots + (p_{n-1} + p_n) l' \quad (4.15)$$

Aus Gleichung (4.14) folgt, dass  $L$  minimal ist, wenn  $L'$  minimal ist. Nach (4.15) ist  $L'$  die mittlere Codewortlänge einer Codierung für eine

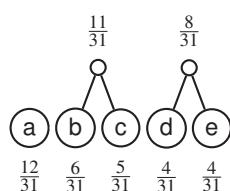


**Abb. 4.18:** Sind  $\sigma_{n-1}$  und  $\sigma_n$  die Symbole mit den geringsten Auftrittswahrscheinlichkeiten, so muss ein optimaler Codebaum die dargestellte Form besitzen.

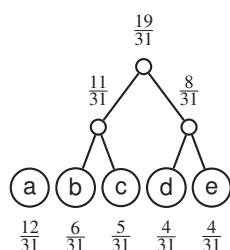
■ 1. Knotenbildung



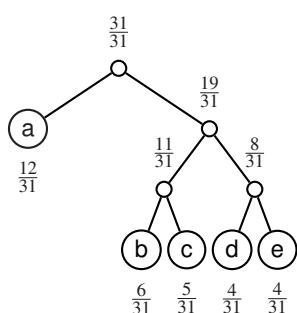
■ 2. Knotenbildung



■ 3. Knotenbildung



■ 4. Knotenbildung



**Abb. 4.19:** Schrittweise Durchführung der Huffman-Codierung

Datenquelle  $X'$ , die sich von unserer ursprünglichen Quelle  $X$  nur dadurch unterscheidet, dass die Symbole  $\sigma_{n-1}$  und  $\sigma_n$  gedanklich durch ein neues Symbol  $\sigma'$  ersetzen wurden, das mit der Auftrittswahrscheinlichkeit  $p_{n-1} + p_n$  emittiert wird. Das bedeutet, dass wir einen optimalen Code für unsere ursprüngliche Datenquelle erhalten, indem wir das Dreieck in Abbildung 4.18 durch den Codebaum eines optimalen Codes für die Datenquelle  $X'$  ersetzen.

$X'$  emittiert ein Symbol weniger als  $X$ , d. h., wir haben es geschafft, das Problem für eine Datenquelle mit  $n$  Symbolen auf ein Problem für eine Datenquelle mit  $n - 1$  Symbolen zurückzuführen. Gehen wir dieses Problem auf die gleiche Weise an, so bleibt irgendwann eine Datenquelle mit nur noch 2 Symbolen übrig. Dies ist der triviale Basisfall, den wir weiter oben bereits abgehandelt haben.

Damit sind wir am Ziel. Wir haben uns das *Huffman-Verfahren* erarbeitet, das der amerikanische Computerwissenschaftler David A. Huffman im Jahr 1951 entwickelte. Zusammengefasst liest es sich wie folgt:



### Huffman-Codierung

- Erzeuge für jedes Symbol der Datenquelle ein Blatt und markiere es mit seiner Auftrittswahrscheinlichkeit.
- Bestimme diejenigen zwei Knoten, die mit den niedrigsten Auftrittswahrscheinlichkeiten  $p_i$  und  $p_j$  markiert sind.
- Erzeuge für die ausgewählten Knoten einen gemeinsamen Vaterknoten und markiere diesen mit der Summe  $p_i + p_j$ .
- Wiederhole die Konstruktion, bis der Codebaum vollständig aufgebaut ist.

Abbildung 4.19 demonstriert die vollständige Durchführung des Huffman-Algorithmus am Beispiel der Datenquelle aus Abbildung 4.12. Als Ergebnis erhalten wir jenen Baum, den wir weiter oben schon vorab als optimal bezeichnet hatten.

In unserem Beispiel waren die beiden Knoten mit den geringsten Wahrscheinlichkeiten stets eindeutig bestimmt, sodass wir bei der Konstruktion des Huffman-Baums an keiner Stelle eine Wahlmöglichkeit hatten. Dass dies nicht immer der Fall ist, belegt das Beispiel in Abbildung 4.20. Hier ist die Auswahl der Knoten mit den geringsten Wahrscheinlichkeiten nicht eindeutig, und je nachdem, wie wir uns entscheiden, entstehen verschiedene Huffman-Bäume. Die beiden abgebilde-

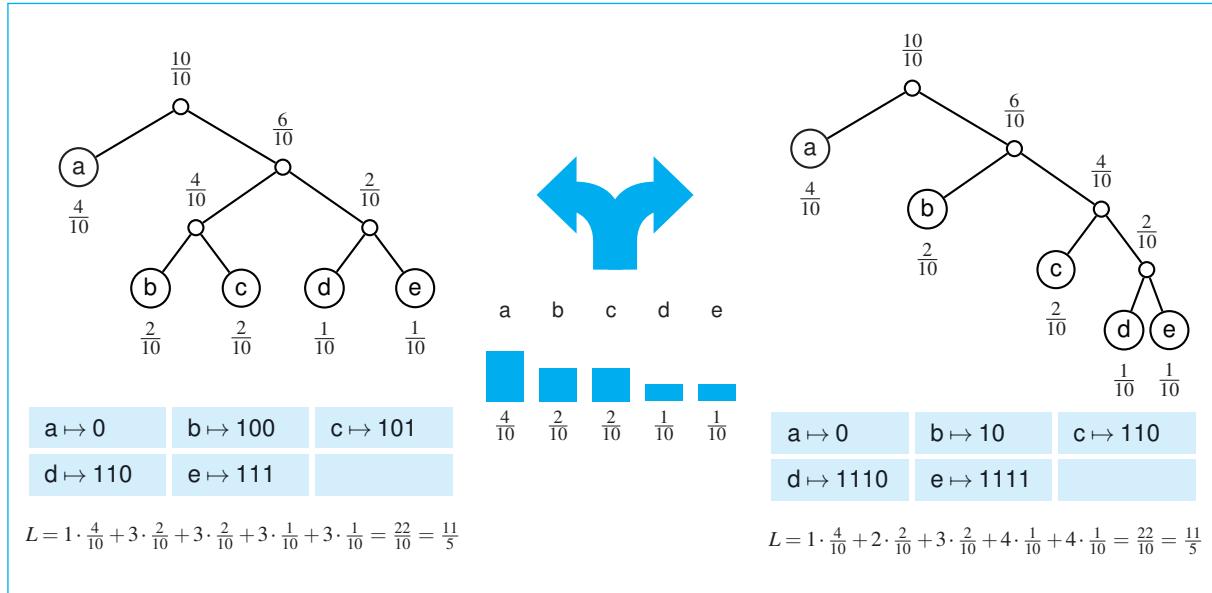


Abb. 4.20: Nicht immer ist die Huffman-Codierung eindeutig. Dennoch ist sichergestellt, dass stets optimale Codes entstehen.

ten Bäume sind nicht isomorph zueinander, sodass sich die Codewörter nicht nur in der Reihenfolge der Nullen und Einsen, sondern auch in der Länge unterscheiden. Dennoch sind beide Codes gleichwertig, wie die Berechnung der mittleren Codewortlängen belegt. Beide Codes wenden im Mittel  $L = \frac{11}{5}$  Bits für die Codierung eines Zeichens auf.

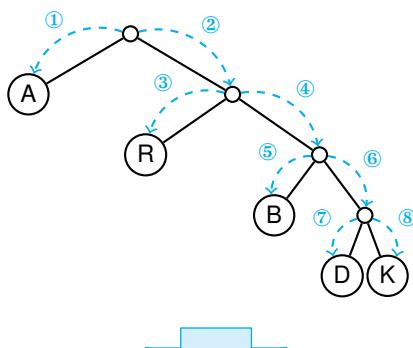
#### 4.4.4 Decodierung präfixfreier Codes

Alle Codes, die wir in den vorangegangenen Abschnitten erzeugt haben, waren präfixfrei. Mit ihnen geht die Codierung einer Nachricht besonders einfach von der Hand, da wir nichts weiter machen müssen, als die Codewörter der einzelnen Zeichen aneinanderzuhängen. Legen wir die Codierung

$$A \mapsto 0 \quad R \mapsto 10 \quad B \mapsto 110 \quad D \mapsto 1110 \quad K \mapsto 1111 \quad (4.16)$$

zugrunde, dann würde die Nachricht ABRAKADABRA beispielsweise so dargestellt werden:

$$01101001111011100110100 \quad (4.17)$$



	Kante	Ausgabe
01101001111011100110100	① A	
01101001111011100110100	②	
01101001111011100110100	④	
01101001111011100110100	⑤ B	
01101001111011100110100	②	
01101001111011100110100	③ R	
01101001111011100110100	① A	
01101001111011100110100	②	
01101001111011100110100	④	
01101001111011100110100	⑥	
01101001111011100110100	⑧ K	
01101001111011100110100	① A	
01101001111011100110100	②	
01101001111011100110100	④	
01101001111011100110100	⑥	
01101001111011100110100	⑦ D	
01101001111011100110100	① A	
01101001111011100110100	②	
01101001111011100110100	④	
01101001111011100110100	⑤ B	
01101001111011100110100	②	
01101001111011100110100	③ R	
01101001111011100110100	① A	

**Abb. 4.21:** Decodierung eines Bitstroms durch Traversierung des Codebaums. Es ist das intuitiv naheliegendste, aber nicht das schnellste Verfahren.

Die Präfixfreiheit garantiert, dass sich die ursprüngliche Nachricht eindeutig aus dem Bitstrom rekonstruieren lässt. Für die Decodierung existieren zwei prinzipiell unterschiedliche Algorithmen, die sich in Laufzeit und Platzverbrauch unterscheiden.

### Baumtraversierung

Am einfachsten gelingt die Decodierung, indem der Bitstrom anhand des Codebaums zurückübersetzt wird. Hierzu wird an der Wurzel gestartet und der binäre Datenstrom Bit für Bit bearbeitet. Bei einer 0 wird der linke Pfad ausgewählt und bei einer 1 der rechte. Ist ein Blatt erreicht, wird das dort vorgefundene Symbol ausgegeben und zur Wurzel zurückgekehrt. Sind noch nicht alle Bits bearbeitet, wird der Vorgang von dort wiederholt (Abbildung 4.21).

### Einsatz einer Decodiertabelle

Die Übersetzung lässt sich durch den Einsatz einer Decodiertabelle beschleunigen. Eine solche Tabelle enthält  $2^l$  Einträge, wobei  $l$  die Anzahl der Bits des längsten Codeworts ist. Jeder Tabelleneintrag enthält ein Ausgabezeichen  $\sigma$  und einen Offset  $i$ . Für die Decodierung wird ein Schieberegister der Länge  $l$  eingesetzt, das nacheinander mit den Datenbits gefüllt wird. Jedes Mal, wenn das Register voll ist, wird sein Inhalt als Index interpretiert, die Decodiertabelle an der betreffenden Stelle ausgelesen und das dort verzeichnete Symbol  $\sigma$  ausgegeben. Danach werden  $i$  Bits nachgeschoben und die beschriebenen Schritte so lange wiederholt, bis alle Bits bearbeitet sind.

Am Beispiel der Codierung (4.16) wollen wir herausarbeiten, wie die beschriebene Decodiertabelle konkret aussieht. In unserem Beispiel setzt sich das längste Codewort aus 4 Bits zusammen, sodass wir eine Tabelle mit  $2^4 = 16$  Einträgen erstellen müssen. Jede Zeile entspricht dann einem bestimmten Binärmuster der Länge 4, und jedes Binärmuster beginnt mit genau einem Codewort  $c(\sigma)$ . In der Tabelle notieren wir das Symbol  $\sigma$  und merken uns zusätzlich die Codewortlänge; dieser Wert ist der oben beschriebene Offset. Als Ergebnis erhalten wir die in Abbildung 4.22 links dargestellte Decodiertabelle.

Führen wir das oben beschriebene Verfahren aus, so erhalten wir aus dem Datenstrom (4.17) tatsächlich die Originalnachricht zurück. Beachten Sie, dass das Schieberegister in den letzten beiden Verarbeitungsschritten nicht mehr vollständig gefüllt ist. Ein Blick auf die Tabelle macht aber schnell klar, dass wir dieses Problem getrost ignorieren können: Wir erhalten immer das richtige Zeichen, egal, ob wir die leerlaufenden Registerstellen durch eine 0 oder eine 1 ersetzen.

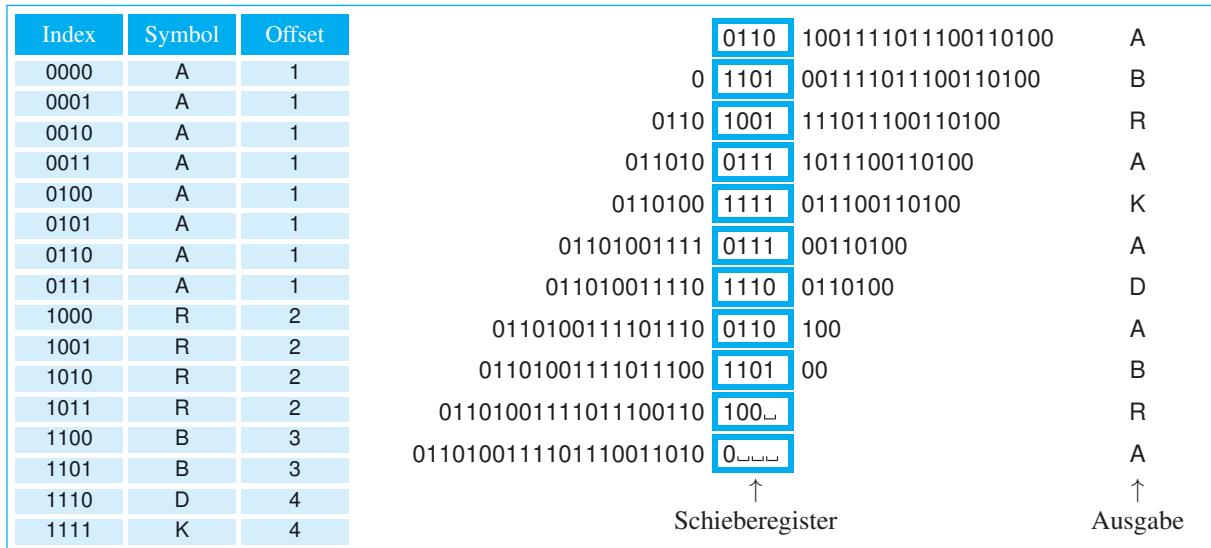


Abb. 4.22: Rückgewinnung einer präfixfrei codierten Nachricht mithilfe einer Decodiertabelle

In der beschriebenen Weise ist das Verfahren nur marginal effizienter als die zuerst beschriebene Baumtraversierung. Dies ändert sich, wenn wir die Decodiertabelle durch eine optimierte Variante ersetzen, wie sie in Abbildung 4.23 zu sehen ist. Die neue Tabelle nutzt aus, dass wir in vielen Fällen mehrere Symbole auf einmal vorhersagen können. Aufpassen müssen wir jetzt am Ende der Decodierung. Beim Leerlaufen des Schieberegisters muss darauf geachtet werden, keine Symbole auszugeben, die in der ursprünglichen Nachricht gar nicht enthalten sind.

Vergleichen wir die vorgestellten Verfahren in Bezug auf ihre Geschwindigkeit und ihren Speicherplatzverbrauch, so erhalten wir ein differenziertes Bild. Zunächst hat die Beispielrechnung gezeigt, dass wir die Decodierung über den Einsatz einer Tabelle erheblich beschleunigen können. Anstatt jedes der 23 Bits einzeln zu bearbeiten, reichen 11 Schieberegisterschritte aus, um die ursprüngliche Nachricht wiederherzustellen. Durch die Verwendung der optimierten Tabelle lässt sich diese Zahl sogar noch weiter, auf nur 7 Schieberegisterschritte, reduzieren. Erkauft wird der Geschwindigkeitszuwachs durch einen erhöhten Speicherplatzbedarf. Da die Decodiertabelle exponentiell mit der Bitbreite des längsten Codeworts wächst, ist das Verfahren bei größeren Codewortlängen nicht mehr anwendbar. In diesen Fällen muss auf das Traversierungsverfahren zurückgegriffen werden, das lediglich für die Ablage des Codebaums Speicherplatz benötigt.

Index	Symbol	Offset			
0000	AAAA	4			
0001	AAA	3			
0010	AAR	4			
0011	AA	2			
0100	ARA	4			
0101	AR	3			
0110	AB	4			
0111	A	1			
1000	RAA	4			
1001	RA	3			
1010	RR	4			
1011	R	2			
1100	BA	4			
1101	B	3			
1110	D	4			
1111	K	4			

The diagram shows the decoding process. A sequence of bits is processed through a shifter register (Schieberegister) with four stages. The output of the shifter register is then mapped to a symbol using the table on the left. The final output is the 'Ausgabe'.

Abb. 4.23: Beschleunigte Decodierung mithilfe einer optimierten Tabelle

## 4.5 Arithmetische Codierung

Die bisher betrachteten Kompressionsalgorithmen haben uns vor Augen geführt, mit welcher Strategie Entropiedecodierungen arbeiten. Sie versuchen, eine Nachricht zu verkürzen, indem häufiger auftretende Symbole mit kürzeren Codewörtern und seltener auftretende Symbole mit längeren Codewörtern belegt werden. Prinzipienbedingt weisen alle Entropiedecodierungen einen Makel auf, der auf ihre Eigenschaft zurückgeht, Nachrichten zeichenweise zu codieren. Diese Eigenschaft hat zur Folge, dass ein Codewort immer aus einer ganzzahligen Anzahl Bits besteht. Das bedeutet, dass die Codewortlänge eines Symbols durch eine natürliche Zahl approximiert werden muss und damit häufig ein wenig neben dem Optimum liegt. Dieser Rundungsfehler sorgt dafür, dass die Kompressionsraten in den meisten Fällen unter dem theoretisch Möglichen bleiben.

Die *arithmetische Codierung* löst das Problem, indem sie eine Nachricht nicht mehr Zeichen für Zeichen bearbeitet, sondern als Ganzes codiert. Das bedeutet, dass sich die einzelnen Bits des codierten Datenstroms, anders als bei den klassischen Entropiedecodierungen, nicht mehr eindeutig einem bestimmten Nachrichtensymbol zuordnen lassen; es findet eine Verwischung der scharfen Grenzen statt, die in den klassischen Entropiedecodierungen immer vorhanden sind.

Konkret funktioniert die arithmetische Codierung so, dass einer vorgelegten Nachricht ein reelles Intervall  $[a; b] \subseteq [0; 1)$  zugeordnet wird, und anschließend in diesem Intervall nach der Zahl mit der kürzesten Binärdarstellung gesucht wird. Die binären Nachkommastellen dieser Zahl sind dann das Ergebnis der arithmetischen Codierung; sie formen zusammen das Codewort (Abbildung 4.24).

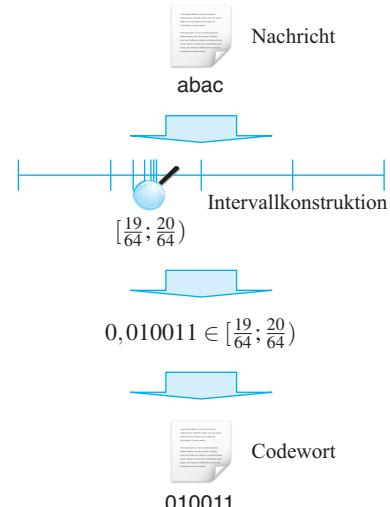
Die Konstruktion der Intervalle ist für die Qualität des produzierten Ergebnisses wesentlich. Sie hat so zu erfolgen, dass Nachrichten mit einer höheren Auftrittswahrscheinlichkeit größere Intervalle zugeordnet werden als Nachrichten mit einer niedrigeren Auftrittswahrscheinlichkeit. Der Grund hierfür ist naheliegend: Je größer das konstruierte Intervall ist, desto eher finden wir darin eine Zahl vor, die nur wenige Nachkommastellen aufweist, und desto kürzer wird die produzierte Binärsequenz. Das bedeutet, dass wir auf ein optimales Ergebnis hoffen können, wenn die Größenverhältnisse zweier Intervalle in dem gleichen Größenverhältnis stehen wie die Auftrittswahrscheinlichkeiten der Nachrichten, die durch diese Intervalle repräsentiert werden.

Für gedächtnislose Quellen, und auf solche wollen wir uns in diesem Abschnitt beschränken, ist die Konstruktion der Intervalle einfach. Sind die Auftrittswahrscheinlichkeiten der Quellsymbole sowie die Länge der zu codierenden Nachricht bekannt, so können wir die Intervalle in einem iterativen Prozess erzeugen, wie er in Abbildung 4.25 zu sehen ist. Wir beginnen die Konstruktion mit dem halboffenen Intervall  $[0; 1)$  und unterteilen es im ersten Schritt in so viele Partitionen, wie das Quellenalphabet Elemente umfasst. Dabei wird jedem Symbol ein Intervall zugeordnet, dessen Länge proportional zu der Auftrittswahrscheinlichkeit des Symbols ist.

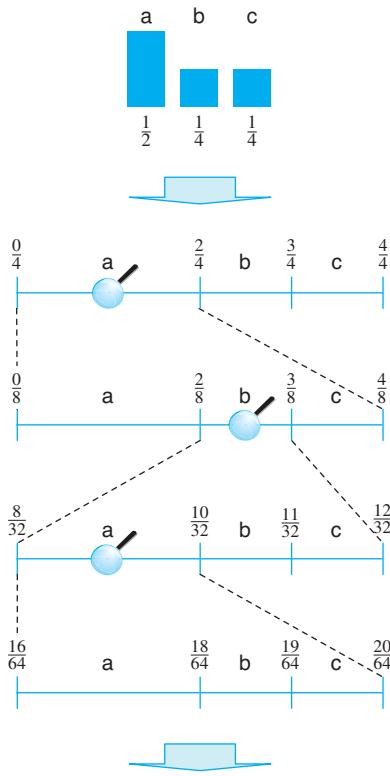
Die Reihenfolge der Intervalle spielt für die Qualität des Ergebnisses keine Rolle; es ist lediglich wichtig, dass der Encoder und der Decoder die gleiche Anordnung verwenden. In unserem Beispiel wurde eine lexikografische Anordnung der Symbole gewählt.

Im nächsten Schritt müssen wir dasjenige Intervall auswählen, das dem ersten Nachrichtensymbol entspricht, und die Partitionierung wiederholen. Wird dieser Prozess so lange ausgeführt, bis alle Symbole bearbeitet sind, so gelangen wir zu einem Intervall  $[a; b]$ , das die zu codierende Nachricht repräsentiert. In einer etwas präziseren Formulierung können wir sagen: Jede reelle Zahl  $c \in [a; b]$  ist ein Repräsentant der zu codierenden Nachricht.

Um das Codewort zu erhalten, müssen wir den Repräsentanten mit der kürzesten Binärdarstellung ausfindig machen. Dass wir einen solchen



**Abb. 4.24:** Arbeitsweise der arithmetischen Codierung



Jede Zahl aus dem Intervall  $[\frac{19}{64}, \frac{20}{64})$  repräsentiert die Nachricht abac.

**Abb. 4.25:** Schrittweise Intervallberechnung für die Nachricht abac

Repräsentanten auf jeden Fall finden können, veranschaulicht Abbildung 4.26. Die Grafik zeigt, dass die Binärzahlen der Form

$$0, b_1 b_2 b_3 \dots b_n$$

ein immer dichter werdendes Netz über das Intervall  $[0;1)$  legen, wenn wir die Zahl  $n$ , d. h. die Anzahl der Nachkommaziffern, sukzessive erhöhen. Irgendwann wird dieses Netz so dicht, dass wir in jedem noch so kleinen Intervall einen Repräsentanten finden.

In unserem Beispiel hat die Suche für  $n = 6$  ein Ende. Mit der Zahl

$$0,010011$$

erreichen wir das Intervall  $[\frac{19}{64}, \frac{20}{64})$  und können das Ergebnis dann unmittelbar ablesen: Die arithmetische Codierung übersetzt die Nachricht abac in die Bitfolge 010011.

Wir wollen an dieser Stelle ein wenig genauer darüber nachdenken, wie die Länge der produzierten Bitfolge mit der Auftrittswahrscheinlichkeit der codierten Nachricht zusammenhängt. Konkret verbirgt sich hinter dieser Überlegung die folgende Frage: Wie dicht muss das Gitter das Intervall  $[0;1)$  überziehen, um ein vorgelegtes Intervall  $[a;b)$  mit Sicherheit zu treffen?

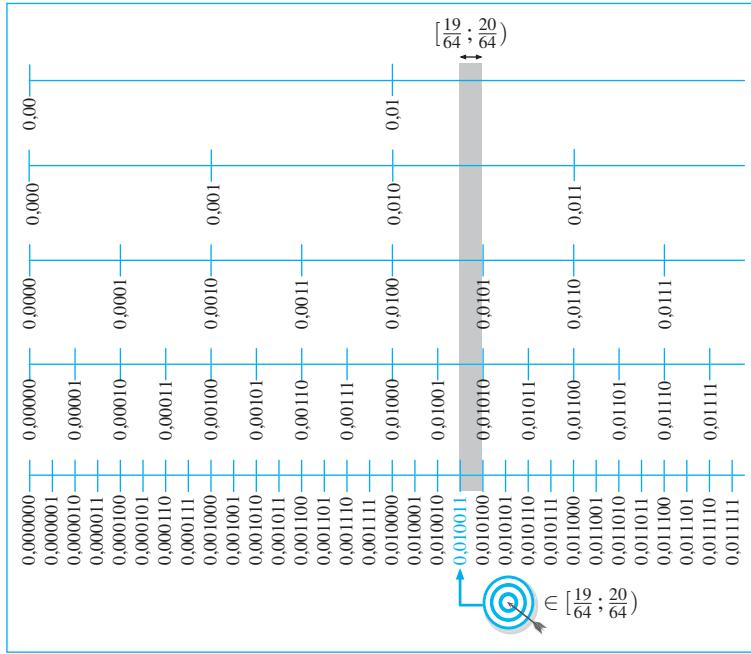
Um der Antwort näher zu kommen, überlegen wir zunächst, wie breit das Intervall ist, das die arithmetische Codierung mit einer Nachricht assoziiert. Hierzu sei  $\Sigma = \{\sigma_1, \dots, \sigma_n\}$  ein Quellenalphabet und  $\mathbf{u} \in \Sigma^N$  eine typische Nachricht der Länge  $N$ , die von einer gedächtnislosen Datenquelle emittiert wird. Die Wahrscheinlichkeiten, mit denen die Symbole  $\sigma_1, \dots, \sigma_n$  von der Datenquelle emittiert werden, bezeichnen wir, wie üblich, mit  $p_1, \dots, p_n$ .

Setzen wir für  $N$  sehr große Zahlen ein, so wird  $\mathbf{u}$  ungefähr  $p_1 N$ -mal das Symbol  $\sigma_1$  enthalten,  $p_2 N$ -mal das Symbol  $\sigma_2$  und so fort. Damit können wir die Länge des Intervalls  $[a;b)$  sehr genau approximieren. Es ist

$$\begin{aligned} b - a &\approx \underbrace{p_1 \cdot \dots \cdot p_1}_{p_1 N\text{-mal}} \cdot \underbrace{p_2 \cdot \dots \cdot p_2}_{p_2 N\text{-mal}} \cdot \dots \cdot \underbrace{p_n \cdot \dots \cdot p_n}_{p_n N\text{-mal}} \\ &= p_1^{p_1 N} \cdot p_2^{p_2 N} \cdot \dots \cdot p_n^{p_n N} = \prod_i p_i^{p_i N} \end{aligned} \quad (4.18)$$

Die Abschätzung wird mit steigenden Werten von  $N$  immer genauer.

Als Nächstes wollen wir bestimmen, wie dicht das Gitter ist, das durch die Binärzahlen mit  $l$  Nachkommaziffern aufgespannt wird. Ein Blick



**Abb. 4.26:** Um die Bitsequenz für die Beispielnachricht abac zu bestimmen, müssen wir innerhalb des Intervalls  $[\frac{19}{64}; \frac{20}{64})$  die Zahl mit der kürzesten Binärdarstellung ausfindig machen. In unserem Beispiel ist dies die Zahl 0,010011. Folglich ist 010011 das Ergebnis der arithmetischen Codierung.

auf Abbildung 4.26 reicht aus, um die Gitterbreite zu erkennen: Sie beträgt

$$2^{-l}$$

Wir treffen das Intervall  $[a; b)$  mit Sicherheit, wenn die Gitterbreite kleiner oder gleich der Intervalllänge  $b - a$  ist. Das bedeutet, dass wir mit jedem Wert für  $l$  auf der sicheren Seite sind, der die folgende Beziehung erfüllt:

$$2^{-l} \leq \prod_i p_i^{p_i N}$$

Logarithmieren wir beide Seiten, so erhalten wir

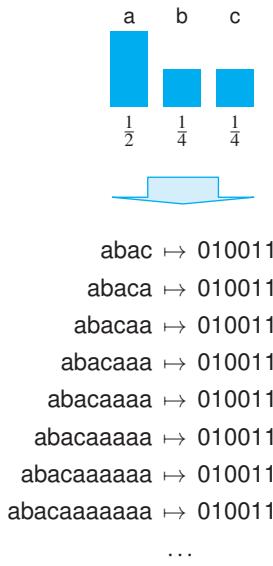
$$-l \leq \sum_i p_i N \log_2 p_i,$$

und dies können wir weiter umformen zu

$$l \geq N \sum_i p_i \log_2 \frac{1}{p_i} \quad (4.19)$$

Der kleinste ganzzahlige Wert, der (4.19) erfüllt, ist

$$\lceil N \sum_i p_i \log_2 \frac{1}{p_i} \rceil$$



**Abb. 4.27:** Für alle dargestellten Nachrichten produziert die arithmetische Codierung die gleiche Bitsequenz. Ohne das Wissen über die Nachrichtenlänge ist die ursprüngliche Symbolsequenz nicht eindeutig rekonstruierbar.

Wählen wir für  $l$  genau diesen Wert, so gilt die Beziehung

$$N \sum_i p_i \log_2 \frac{1}{p_i} \leq l \leq 1 + N \sum_i p_i \log_2 \frac{1}{p_i} \quad (4.20)$$

Formen wir diese Gleichung noch minimal um, so erhalten wir ein wichtiges Ergebnis über die mittlere Codewortlänge  $L$ . Diese gibt an, mit wie vielen Bits ein Symbol der Nachricht durchschnittlich codiert wird, und lässt sich in unserem Fall so ausdrücken:

$$L = \frac{l}{N}$$

Dividieren wir alle Seiten der Ungleichung (4.20) durch  $N$ , so ergibt sich die Beziehung, die wir suchen:

$$\sum_i p_i \log_2 \frac{1}{p_i} \leq L \leq \frac{1}{N} + \sum_i p_i \log_2 \frac{1}{p_i}$$

Diese Ungleichung zeigt, dass die arithmetische Codierung zu einer mittleren Codewortlänge führt, die sich immer näher an die Grenze

$$\sum_i p_i \log_2 \frac{1}{p_i} \quad (4.21)$$

herantastet. Prägen Sie sich den Wert (4.21) gut ein, denn in Kapitel 5 werden wir offenlegen, dass er eine Schranke bildet, die kein Kompressionsverfahren nach unten durchbrechen kann. Die Tatsache, dass sich die mittlere Codewortlänge der arithmetischen Codierung diesem Wert annähert, bedeutet, dass das Verfahren optimal ist, d. h. mit zunehmender Nachrichtenlänge dem theoretischen Optimum zustrebt.

## Decodierung

Ein arithmetischer Decoder arbeitet nach dem gleichen Grundprinzip wie ein arithmetischer Encoder. Sind das Quellenalphabet und die Auftrittswahrscheinlichkeiten der verschiedenen Symbole bekannt, so kann er empfängerseitig die gleiche Intervallkonstruktion durchführen, die der Encoder senderseitig vollzogen hat. Ein Problem kann der Decoder allerdings nicht lösen: Da die empfangene Bitsequenz keinerlei Information über die Länge der Originalnachricht in sich trägt, kann er nicht entscheiden, nach wie vielen Schritten die Intervallbildung gestoppt werden muss. Dieses Problem ist elementar und auch an unserem Beispiel gut zu beobachten. Abbildung 4.27 zeigt, dass die arithmetische Codierung für unendlich viele Nachrichten die gleiche Bitsequenz

erzeugt. Streng genommen ist die arithmetische Codierung in der bisher präsentierten Form also gar keine Codierung im Sinne von Definition 3.1.

Um die arithmetische Codierung praktisch einzusetzen, kommen wir nicht umhin, dem Decoder die Länge der Originalnachricht mitzuteilen. Dies kann auf zwei prinzipiell unterschiedliche Weisen erfolgen:

### ■ Übermittlung der Nachrichtenlänge

Die einfachste Möglichkeit besteht darin, die Nachrichtenlänge getrennt zu codieren und dem eigentlichen Bitstrom voranzustellen. Das asymptotische Verhalten der mittleren Codewortlänge bleibt hierdurch unbeeinflusst. Für die Codierung der Länge  $N$  sind  $\log_2 N$  Bits ausreichend, sodass sich die mittlere Codewortlänge um  $\frac{\log_2 N}{N}$  vergrößert. Mit zunehmender Nachrichtenlänge  $N$  strebt dieser Wert gegen 0.

### ■ Verwendung eines Endezeichens

Alternativ kann das Quellenalphabet um ein dediziertes Symbol ergänzt werden, das der Markierung des Nachrichtenendes dient. Wird dieses Symbol, wir nennen es hier kurzerhand „#“, empfängerseitig erkannt, so kann der Decoder die Intervallbildung an dieser Stelle abbrechen und das bisher berechnete Ergebnis anzeigen. Auch dieser Ansatz hat keinen Einfluss auf die asymptotische Entwicklung der mittleren Codewortlänge. Die Ergänzung des Quellenalphabets hat zur Folge, dass die mittlere Codewortlänge, gemäß (4.21), gegen den Wert

$$p_{\#} \log_2 \frac{1}{p_{\#}} + \sum_i p_i \log_2 \frac{1}{p_i}$$

strebt, wobei  $p_{\#}$  die Auftrittswahrscheinlichkeit des Symbols „#“ ist. Für immer größer werdende Werte von  $N$  konvergiert  $p_{\#}$  gegen 0, und im Übungsteil auf Seite 337 werden Sie zeigen, dass dann auch das Produkt  $p_{\#} \log \frac{1}{p_{\#}}$  gegen 0 strebt. Folgerichtig hat die Hinzunahme eines dedizierten Endezeichens keinerlei Auswirkung auf die asymptotische Entwicklung der mittleren Codewortlänge.



Zu Beginn dieses Abschnitts haben wir erwähnt, dass die arithmetische Codierung der Huffman-Codierung qualitativ überlegen ist, und dennoch wird die Huffman-Codierung in der Praxis ungleich häufiger eingesetzt. Wie passt dies zusammen?

Für die vergleichsweise geringe Verbreitung der arithmetischen Codierung sind zwei Gründe maßgebend. Zum einen ist die Programmierung von effizienten Encodern und Decodern aufwendig: Für die Darstellung der Zahlen aus dem Intervall  $[0; 1)$  wird eine Genauigkeit benötigt, die mit wachsender Nachrichtenlänge kontinuierlich zunimmt. Es muss daher entweder auf komplizierte Zahlfomate mit einer beliebigen Genauigkeit zurückgegriffen werden, oder die Implementierung muss so trickreich erfolgen, dass immer nur der relevante Bitabschnitt gespeichert wird. Beide Varianten sind mit einem Aufwand verbunden, der den eines Huffman-Encoders deutlich übersteigt. Dieser kann sich darauf beschränken, vergleichsweise einfache Baumstrukturen zu erzeugen.

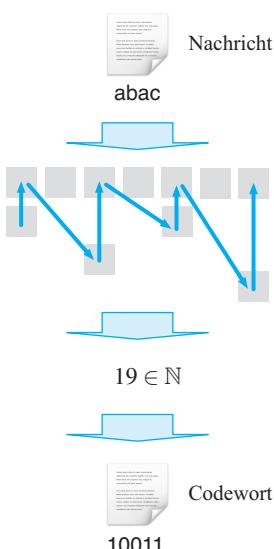
Zum anderen verhindern rechtliche Aspekte den breitflächigen Einsatz der arithmetischen Codierung. Mehrere Algorithmen zur effizienten Codierung und Decodierung sind patentrechtlich geschützt und damit nicht ohne die Zahlung von Lizenzgebühren nutzbar. Beide Gründe haben in der Vergangenheit dazu geführt, dass die Huffman-Codierung eine weitaus höhere Verbreitung besitzt. Problematisch ist dies nicht, da der qualitative Vorteil der arithmetischen Codierung in den meisten Fällen gering ist und sich der zusätzliche Aufwand daher nur selten lohnt.

## 4.6 ANS-Codierung

In den letzten beiden Abschnitten haben wir mit der Huffman-Codierung und der arithmetischen Codierung zwei Verfahren kennen gelernt, die nicht nur vollkommen unterschiedlich arbeiten, sondern auch sehr unterschiedliche Stärken und Schwächen aufweisen. Im direkten Vergleich der beiden ist die Huffman-Codierung die deutlich schnellere und zudem viel einfacher zu implementieren. Den Preis zahlen wir in einer Kompressionsrate, die im Allgemeinen etwas unter dem theoretischen Optimum liegt. Hier entfaltet die arithmetische Codierung ihr volles Potenzial. Die von ihr erzielte Kompressionsrate tastet sich mit steigender Nachrichtenlänge immer weiter an das theoretische Optimum heran.

In diesem Abschnitt werden wir mit der ANS-Codierung (*Asymmetric Numeral Systems*) ein Kompressionsverfahren besprechen, das die Vorteile der beiden anderen in einem hohen Maß in sich vereint: Sie arbeitet fast so effizient wie die Huffman-Codierung und erzielt dabei eine Kompressionsrate, die sich, analog zur arithmetischen Codierung, am Rand des theoretischen Optimums bewegt.

Die ANS-Codierung basiert dabei auf einer ganz ähnlichen Idee wie die arithmetische Codierung: Eine Nachricht wird durch den Encoder auf eine einzige Zahl abgebildet und deren Binärrepräsentation als das Codewort aufgefasst. Anders als bei der arithmetischen Codierung ist diese Zahl aber keine rationale, sondern eine natürliche (Abbildung 4.28). Die Berechnung kann dabei auf unterschiedliche Art und Weise erfolgen, was zu einer Unterteilung in mehrere ANS-Untervarianten führt. Hierzu gehören:



**Abb. 4.28:** Arbeitsweise der ANS-Codierung

### ■ uABS (*Uniform Asymmetric Binary Systems*)

Diese Variante ist auf die Verwendung eines zweielementigen Quellenalphabets optimiert und liefert auch für kurze Nachrichten sehr gute Ergebnisse. Je länger die codierten Nachrichten werden, desto näher rückt die Kompressionsrate an das theoretische Optimum heran. In Abschnitt 4.6.2 werden wir uns näher mit dieser Variante beschäftigen.

### ■ rANS (*Range Asymmetric Numeral Systems*)

Die Grundidee der uABS-Codierung wird bei dieser Variante auf Quellenalphabete mit einer beliebigen Anzahl an Elementen übertragen. Die rANS-Variante ist einfacher zu erlernen als die uABS-Codierung, weshalb wir sie aus didaktischen Gründen vor der uABS-Codierung, in Abschnitt 4.6.1, besprechen.

### ■ tANS (*Table Asymmetric Numeral Systems*)

Diese Variante basiert auf der Idee, die Leistung der klassischen ANS-Codierungen zu steigern, indem die Berechnungen in Form von Tabellenzugriffen durchgeführt werden. Mit ihr lassen sich ANS-Decoder implementieren, die ähnlich effizient arbeiten wie schnelle Huffman-Decoder.



Die ANS-Codierung fasziniert aus mehreren Gründen. Neben ihrer Eigenschaft, die positiven Merkmale der Huffman-Codierung und der arithmetischen Codierung zu vereinen, ist es das Datum ihrer Entdeckung. Die ANS-Codierung wurde im Jahr 2009 von Jarosław Duda publiziert und gehört damit zu den jüngsten bedeutenden Entdeckungen im Bereich der theoretischen Informatik [26]. Sie fiel in eine Zeit, in der das Gebiet der Quellencodierung von den meisten Experten als weitgehend abgearbeitet angesehen wurde und daher kaum noch jemand mit einer bahnbrechenden Entdeckung rechnete.

In den Folgejahren konnte die ANS-Codierung ihre Praxistauglichkeit mehrfach unter Beweis stellen und in etlichen Bereichen die bis dato eingesetzten Verfahren nach und nach verdrängen. Den Anfang machte die Firma Facebook im Jahr 2015 durch die Verwendung von ANS in dem Open-Source-Kompressor *Zstandard* [31]. Etwa zur gleichen Zeit hatte Apple die ANS-Codierung in den *LZFSE*-Algorithmus integriert, der eine ähnliche Kompressionsstrategie verfolgt und ebenfalls frei verfügbar ist [57]. Im Jahr 2017 fand die ANS-Codierung Einzug in den freien *JPEG-XL*-Standard [66], und seit 2019 wird sie von Google in ihrem JPEG-ähnlichen Kompressionsstandard *pik* verwendet [36]. Microsoft erkannte das Potenzial des neuen Verfahrens um die gleiche Zeit.

Duda hatte keine Rechte an seinem Verfahren beansprucht und von Anfang an versucht, es von patentrechtlichen Beschränkungen freizuhalten. Doch dieser Plan schlug fehl. Als erste Firma hatte Google versucht, das freie Verfahren zu patentieren, der Antrag wurde im Jahr 2018 jedoch abgewiesen. Microsoft hatte mit einem ähnlichen Antrag Erfolg und ist seit Anfang 2022 im Besitz eines Patens auf die rANS-Technologie.

## 4.6.1 rANS-Codierung

Weiter oben haben wir bereits vorweggenommen, dass die ANS-Codierung eine Nachricht in Form einer einzigen natürlichen Zahl repräsentiert. Wie so etwas möglich ist, wollen wir uns an einem konkreten Beispiel erarbeiten. Um die Dinge an dieser Stelle nicht unnötig zu verkomplizieren, gehen wir zunächst von einem binären Quellenalphabet  $\Sigma = \{0, 1\}$  aus und stellen uns die Frage, wie die Nachricht

$$u = 01101$$

auf möglichst einfache Weise auf eine natürliche Zahl abgebildet werden kann. Die Antwort liegt für dieses Beispiel auf der Hand. Die Abbildung gelingt ohne Mühe, indem die Nachricht ganz einfach als die Binärrepräsentation der gesuchten Zahl aufgefasst wird, in unserem Beispiel also der Zahl 13.

Algorithmisch können wir die Umwandlung in Form einer Rekursion formulieren. Wir beginnen mit der Zahl  $x_0 = 0$  und wenden im Anschluss daran sukzessive die *Codierungsfunktion*

$$x_i := C(u_i, x_{i-1}) := 2x_{i-1} + u_i \quad (4.22)$$

an, wobei  $u_i \in \{0, 1\}$  das  $i$ -te Symbol der zu codierenden Nachricht ist. In unserem Beispiel gilt also:

$$u_1 = 0, u_2 = 1, u_3 = 1, u_4 = 0, u_5 = 1$$

Der gebräuchlichen Terminologie folgend, bezeichnen wir den Wert von  $x$  als den *Zustand* des Encoders. Demnach startet der Encoder im *Startzustand* 0 und berechnet daraus durch die Anwendung der Codierungsfunktion  $C$  eine Reihe von *Folgezuständen*. Für unser Beispiel läuft die vollständige Codierung so ab, wie es links in Abbildung 4.29 zu sehen ist.

Codierung	Decodierung
$u = u_1 u_2 u_3 u_4 u_5 = 01101$  $x_1 = C(u_1, x_0) = C(0, 0) = 2 \cdot 0 + 0 = 0$ $x_2 = C(u_2, x_1) = C(1, 0) = 2 \cdot 0 + 1 = 1$ $x_3 = C(u_3, x_2) = C(1, 1) = 2 \cdot 1 + 1 = 3$ $x_4 = C(u_4, x_3) = C(0, 3) = 2 \cdot 3 + 0 = 6$ $x_5 = C(u_5, x_4) = C(1, 6) = 2 \cdot 6 + 1 = 13$  $v = x_5 = 1101$	$v = x_5 = 1101$  $D(x_5) = D(13) = (u_5, x_4)$ mit $\begin{cases} u_5 = 13 \bmod 2 = 1 \\ x_4 = \lfloor \frac{13}{2} \rfloor = 6 \end{cases}$ $D(x_4) = D(6) = (u_4, x_3)$ mit $\begin{cases} u_4 = 6 \bmod 2 = 0 \\ x_3 = \lfloor \frac{6}{2} \rfloor = 3 \end{cases}$ $D(x_3) = D(3) = (u_3, x_2)$ mit $\begin{cases} u_3 = 3 \bmod 2 = 1 \\ x_2 = \lfloor \frac{3}{2} \rfloor = 1 \end{cases}$ $D(x_2) = D(1) = (u_2, x_1)$ mit $\begin{cases} u_2 = 1 \bmod 2 = 1 \\ x_1 = \lfloor \frac{1}{2} \rfloor = 0 \end{cases}$ $D(x_1) = D(0) = (u_1, x_0)$ mit $\begin{cases} u_1 = 0 \bmod 2 = 0 \\ x_0 = \lfloor \frac{0}{2} \rfloor = 0 \end{cases}$  $u = u_1 u_2 u_3 u_4 u_5 = 01101$

Abb. 4.29: Codierung und Decodierung der Nachricht 01101 mit den Formeln (4.22) und (4.23).

Die Decodierung ist genauso einfach möglich, durch die sukzessive Anwendung der folgenden *Dekodierungsfunktion*:

$$D(x_i) := (u_i, x_{i-1}) \text{ mit } \begin{cases} u_i := x_i \bmod 2 \\ x_{i-1} := \lfloor \frac{x_i}{2} \rfloor \end{cases} \quad (4.23)$$

Die rechte Seite in Abbildung 4.29 zeigt Schritt für Schritt, wie die Decodierung für unser Beispiel abläuft. Beachten Sie, dass der Dekoder die Nachrichtensymbole in umgekehrter Reihenfolge rekonstruiert. Das zuletzt codierte Zeichen ist das zuerst rekonstruierte. Diese Eigenschaft unterscheidet die ANS-Codierung von fast allen in diesem Buch besprochenen Codierungen.

Als Nächstes wollen wir das Erarbeitete in Form einer *Codierungstabelle* darstellen, wie sie im Kontext der ANS-Codierung gerne verwendet wird:

$x_i$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...
$u_i = 0$	0		1		2		3		4		5		6		7		8		9		...
$u_i = 1$		0		1		2		3		4		5		6		7		8		9	...

Abb. 4.30: Codierungstabelle, abgeleitet aus der Iterationsvorschrift (4.22)

det wird. Wie diese Tabelle für unser Beispiel aussieht, ist in Abbildung 4.30 zu sehen.

In der oberen Zeile sind alle Zustände vermerkt, die der ANS-Encoder annehmen kann. Dies sind potenziell alle natürlichen Zahlen, d. h., wir müssen uns die Tabelle so vorstellen, als sei sie nach rechts unbegrenzt. Wir werden diese Zeile nutzen, um in jeder Iteration den Folgezustand abzulesen. Die unteren beiden Zeilen markieren den aktuellen Zustand des Decoders, wobei jede Zeile eines der beiden möglichen Symbole 0 und 1 aus dem Quellenalphabet  $\Sigma$  repräsentiert.

Mit Hilfe dieser Tabelle können wir die Codierung grafisch ausführen. Hierfür nehmen wir an, wir befänden uns am Anfang der  $i$ -ten Iteration, d. h.,  $x_{i-1}$  ist der aktuelle Zustand des Encoders und  $u_i$  das nächste zu verarbeitende Zeichen. Jetzt führen wir nacheinander zwei Schritte aus. Im ersten Schritt springen wir in die Zeile, die das Zeichen  $u_i$  repräsentiert, und zwar an jene Stelle, die mit dem aktuellen Zustand  $x_{i-1}$  markiert ist. Im zweiten Schritt bewegen wir uns nach oben in die erste Zeile, wo wir den Folgezustand  $x_i$  ablesen können. Abbildung 4.31 zeigt, wie wir auf diese Weise die weiter oben rein rechnerisch durchgeführte Codierung grafisch nachvollziehen können. Die Decodierung erfolgt analog. Wie in Abbildung 4.32 zu sehen ist, müssen wir lediglich alle Pfeile in der umgekehrten Richtung durchlaufen.

Als Nächstes wenden wir uns der Frage zu, um wieviele Bits sich das Codewort in jeder Iteration verlängert hat. Für die von uns gewählte Codierungsfunktion (4.22) ist die Antwort trivial. Da wir mit ihr die Symbole unserer Nachricht schlachtweg aneinander hängen, verlängert sich das Codewort in jeder Iteration um genau 1 Bit. In Kapitel 5 werden wir zu der Erkenntnis gelangen, dass dieses Vorgehen optimal ist, wenn die beiden Symbole unseres Quellenalphabets gleich oft vorkommen und keinerlei Kontextabhängigkeiten bestehen.

In unserem Beispiel sind diese Voraussetzungen aber nicht erfüllt. In der Nachricht 01101 kommt das Symbol 0 zweimal und das Symbol 1

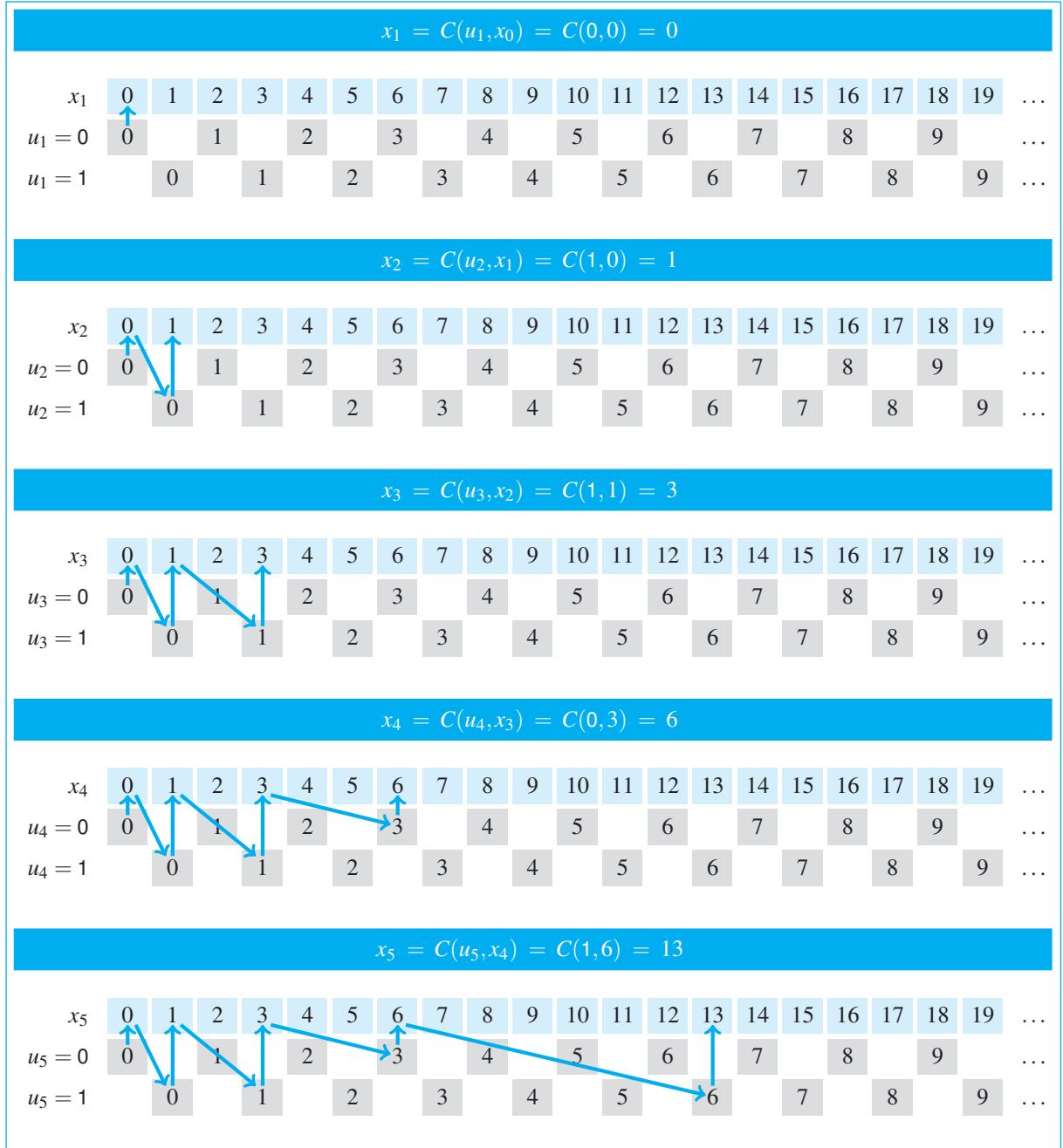


Abb. 4.31: Grafische Durchführung der Codierung aus Abbildung 4.29 (links)

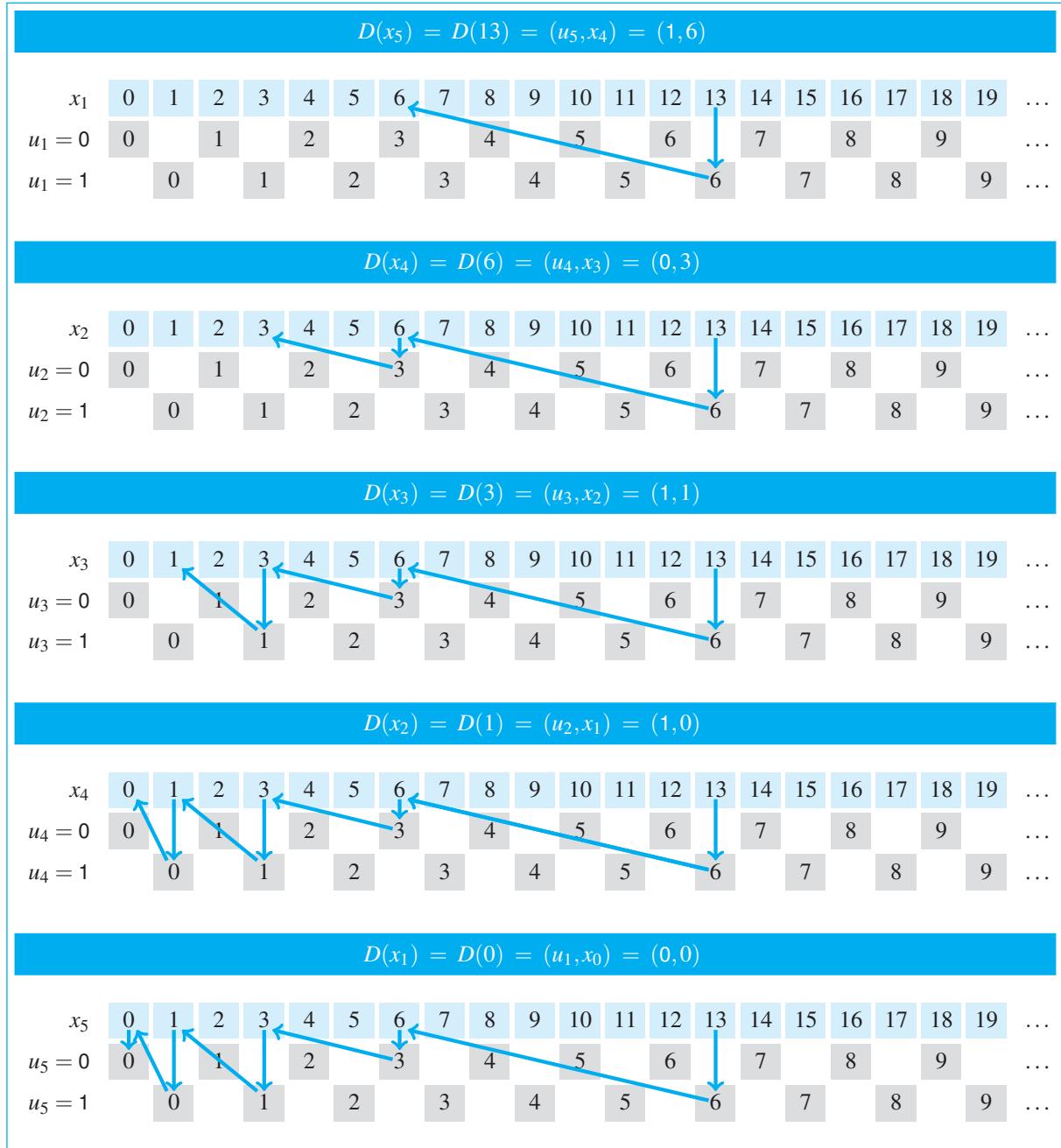


Abb. 4.32: Grafische Durchführung der Decodierung aus Abbildung 4.29 (rechts)

dreimal vor. Wir haben es demnach mit der in folgenden Wahrscheinlichkeitsverteilung zu tun:

$$p(0) = \frac{2}{5}, \quad p(1) = \frac{3}{5}$$

In Kapitel 5 werden wir erarbeiten, dass wir in diesem Fall ein optimales Ergebnis erhalten, wenn die Nachricht im  $i$ -ten Iterationsschritt um durchschnittlich

$$\log_2 \frac{1}{p(u_i)}$$

Bits verlängert wird. Für  $p(u_i) = \frac{1}{2}$  ergibt sich aus dieser Formel das bereits Gesagte: Die Verlängerung um 1 Bit liefert in diesem Fall ein optimales Ergebnis.

Da die Symbole in unserer Beispieldnachricht unterschiedlich häufig auftreten, ist die Funktion (4.22) nicht geeignet. Was wir stattdessen benötigen, ist eine Formel, die das Codewort für jede verarbeitete 0 um ungefähr

$$\log_2 \frac{1}{p(0)} = \log_2 \frac{5}{2} \approx 1,16 \text{ Bits}$$

und jede verarbeitete 1 um ungefähr

$$\log_2 \frac{1}{p(1)} = \log_2 \frac{5}{3} \approx 0,77 \text{ Bits}$$

verlängert.

Wir wollen herausarbeiten, wie die Codierungsfunktion  $C$  geändert werden muss, um die gewünschte Eigenschaft zu erfüllen. Hierfür müssen wir uns lediglich daran erinnern, dass das Ergebnis der ANS-Codierung die Binärrepräsentation des final erreichten Zustands ist. Um eine Zahl  $x$  binär zu repräsentieren, benötigen wir ungefähr  $\log_2 x$  Bits. Möchten wir also, dass sich das Codewort in jeder Iteration um ungefähr  $\log_2 \frac{1}{p(u_i)}$  Bits verlängert, dann muss die gewählte Codierungsfunktion die Beziehung

$$\log_2 x_i - \log_2 x_{i-1} \approx \log_2 \frac{1}{p(u_i)}$$

erfüllen, was dasselbe ist wie:

$$\log_2 C(u_i, x_{i-1}) - \log_2 x_{i-1} \approx \log_2 \frac{1}{p(u_i)}$$

Diese Beziehung lässt sich mit wenigen Federstrichen nach  $C(u_i, x_{i-1})$  auflösen:

$$C(u_i, x_{i-1}) \approx \frac{x_{i-1}}{p(u_i)} \quad (4.24)$$

Codierung	Decodierung
$u = u_1 u_2 u_3 u_4 u_5 = 01101$  $x_1 = C(u_1, x_0) = C(0, 0) = \left\lfloor \frac{0}{2} \right\rfloor \cdot 5 + 0 \bmod 2 = 0$ $x_2 = C(u_2, x_1) = C(1, 0) = \left\lfloor \frac{0}{3} \right\rfloor \cdot 5 + 2 + 0 \bmod 3 = 2$ $x_3 = C(u_3, x_2) = C(1, 2) = \left\lfloor \frac{2}{3} \right\rfloor \cdot 5 + 2 + 2 \bmod 3 = 4$ $x_4 = C(u_4, x_3) = C(0, 4) = \left\lfloor \frac{4}{2} \right\rfloor \cdot 5 + 4 \bmod 2 = 10$ $x_5 = C(u_5, x_4) = C(1, 10) = \left\lfloor \frac{10}{3} \right\rfloor \cdot 5 + 2 + 10 \bmod 3 = 18$  $v = 18 = 10010$	$v = 18 = 10010$  $D(x_5) = D(18) = (u_5, x_4)$ mit $\begin{cases} u_5 = 1 \\ x_4 = \left\lfloor \frac{18}{5} \right\rfloor \cdot 3 - 2 + 3 = 10 \end{cases}$ $D(x_4) = D(10) = (u_4, x_3)$ mit $\begin{cases} u_4 = 0 \\ x_4 = \left\lfloor \frac{10}{5} \right\rfloor \cdot 2 + 0 = 4 \end{cases}$ $D(x_3) = D(4) = (u_3, x_2)$ mit $\begin{cases} u_3 = 1 \\ x_2 = \left\lfloor \frac{4}{5} \right\rfloor \cdot 3 - 2 + 4 = 2 \end{cases}$ $D(x_2) = D(2) = (u_2, x_1)$ mit $\begin{cases} u_2 = 1 \\ x_2 = \left\lfloor \frac{2}{5} \right\rfloor \cdot 3 - 2 + 2 = 0 \end{cases}$ $D(x_1) = D(0) = (u_1, x_0)$ mit $\begin{cases} u_1 = 0 \\ x_0 = \left\lfloor \frac{0}{5} \right\rfloor \cdot 2 + 0 = 0 \end{cases}$  $u = u_1 u_2 u_3 u_4 u_5 = 01101$

Abb. 4.33: Codierung und Decodierung der Nachricht 01101

Aus zwei Gründen können wir diese Formel aber nicht direkt verwenden. Zum einen muss die Codierungsfunktion  $C$  auf eine natürliche Zahl abbilden. Zum anderen muss sie die Forderung der Injektivität erfüllen, da nur dann eine verlustfreie Decodierung möglich ist. Beides ist hier nicht gegeben.

Trotzdem sind wir der Ziellinie schon recht nahe, da sich beide Forderungen durch eine geringfügige Anpassung der Codierungsfunktion erfüllen lassen. Eine geeignete Iterationsvorschrift entsteht beispielsweise durch die folgende Modifikation:

$$C(u_i, x_{i-1}) := \begin{cases} \left\lfloor \frac{x_{i-1}}{2} \right\rfloor \cdot 5 + x_{i-1} \bmod 2 & \text{falls } u_i = 0 \\ \left\lfloor \frac{x_{i-1}}{3} \right\rfloor \cdot 5 + 2 + x_{i-1} \bmod 3 & \text{falls } u_i = 1 \end{cases} \quad (4.25)$$

Für unsere Beispielnachricht erzeugt diese Formel den links in Abbildung 4.33 dargestellten Codierungsverlauf.



Wahrscheinlich haben Sie bemerkt, dass sich das Codewort durch die Anwendung der neuen Codierungsfunktion (4.25) sogar verlängert hat. Anstatt vier Bits (Codeword 1101) benötigen wir zur Darstellung der Nachricht 01101 jetzt fünf Bits (Codewort 10010). Dies ist kein Fehler, sondern der Tatsache geschuldet, dass wir als Beispiel eine wahrlich kurze Nachricht gewählt haben. Alle Aussagen, die wir über die Güte einer Codierung treffen, sind aber asymptotische Aussagen, d. h. mathematische Grenzwertbetrachtungen, in denen wir die Nachrichtenlänge gedanklich gegen Unendlich streben lassen.

	Segment 1					Segment 2					Segment 3					Segment 4					
$x_i$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...
$u_i = 0$	0	1				2	3				4	5				6	7				...
$u_i = 1$		0	1	2				3	4	5			6	7	8			9	10	11	...

Abb. 4.34: Codierungstabelle, abgeleitet aus der Iterationsvorschrift (4.26)

Die Decodierung ist mit der folgenden Funktion möglich:

$$D(x_i) := (u_i, x_{i-1}) = \begin{cases} (0, \lfloor \frac{x_i}{5} \rfloor \cdot 2 + x_i \bmod 5) & \text{falls } x_i \bmod 5 < 2 \\ (1, \lfloor \frac{x_i}{5} \rfloor \cdot 3 - 2 + x_i \bmod 5) & \text{falls } x_i \bmod 5 \geq 2 \end{cases}$$

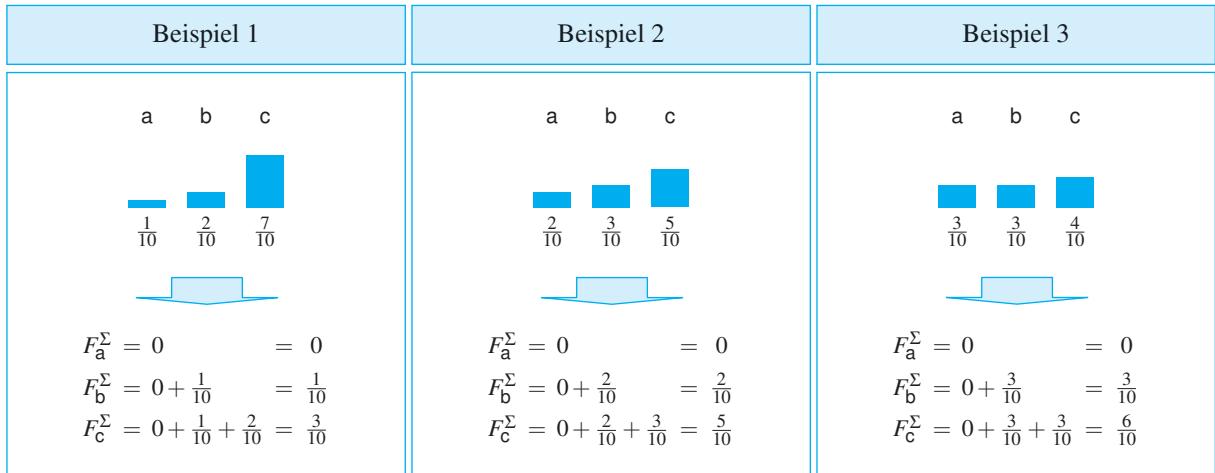
Die rechte Seite in Abbildung 4.33 zeigt, dass wir durch die sukzessive Anwendung dieser Iterationsvorschrift tatsächlich unsere ursprüngliche Nachricht zurück erhalten.

Mit einem Blick auf die Codierungstabelle in Abbildung 4.34 lässt sich sehr anschaulich die Konstruktionsidee verstehen, die sich hinter diesen Funktionen verbirgt. Die in unserem Beispiel auftretenden Wahrscheinlichkeiten  $p(0) = \frac{2}{5}$  und  $p(1) = \frac{3}{5}$  haben den gemeinsamen Nenner 5, was die Codierungstabelle in sich wiederholende Segmente der Länge 5 zerfällt. Jedes dieser Segmente ist so aufgebaut, dass wir darin jedes Symbol des Quellenalphabets wiederfinden, und zwar in einer Häufigkeit, die dessen Auftrittswahrscheinlichkeit entspricht.

Die asymmetrische Verteilung der Quellsymbole wirkt sich unmittelbar auf die Codewortlänge aus. Wird in unserem Beispiel das häufiger vorkommende Symbol 1 verarbeitet, so führt dies dazu, dass der Zustand des Encoders langsamer wächst als bei der Verarbeitung des seltener vorkommenden Symbols 0. Je kleiner der Zustand des Encoders am Ende ist, desto weniger Bits werden für dessen Binärrepräsentation benötigt und desto kürzer ist das produzierte Codewort. Damit ist der Steuerungsmechanismus offengelegt.

Eine zentrale Eigenschaft der rANS-Codierung ist es, dass die Konstruktion nicht nur für binäre, sondern für beliebige Code-Alphabete funktioniert. Um zu verstehen, wie die oben durchgeföhrten Rechnungen verallgemeinert werden können, nehmen wir an, das Quellenalphabet  $\Sigma$  enthalte die Symbole  $\sigma_1, \dots, \sigma_n$ , die mit den Wahrscheinlichkeiten

$$p(\sigma_i) = \frac{F_{\sigma_i}}{M}$$



**Abb. 4.35:** Berechnung von  $F_{\sigma_i}^\Sigma$  am Beispiel eines dreielementigen Quellenalphabets  $\Sigma = \{a, b, c\}$

auftreten. In der englischen Literatur wird der Zähler  $F_{\sigma_i}$  als der *frequency count* des Symbols  $\sigma_i$  bezeichnet. Eng damit verbunden ist der *cumulative frequency count*  $F_{\sigma_i}^\Sigma$ , der folgendermaßen definiert ist:

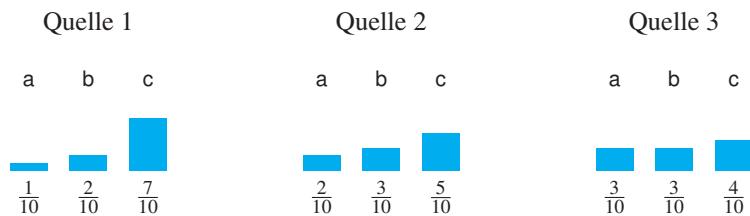
$$F_{\sigma_i}^\Sigma := F_{\sigma_1} + \dots + F_{\sigma_{i-1}}$$

Für das erste Symbol ist der Wert  $F_{\sigma_i}^\Sigma$  immer gleich 0. Für alle anderen wird er berechnet, indem die Zähler der Auftrittswahrscheinlichkeiten aller Vorgängersymbole addiert werden. Abbildung 4.35 demonstriert die Berechnung von  $F_{\sigma_i}^\Sigma$  anhand mehrerer Beispiele.

Mit der geleisteten Vorarbeit sind wir in der Lage, die oben verwendete Codierungs vorschrift (4.25) folgendermaßen zu verallgemeinern:

$$C(u_i, x_{i-1}) := \left\lfloor \frac{x_{i-1}}{F_{u_i}} \right\rfloor \cdot M + F_{u_i}^\Sigma + x_{i-1} \bmod F_{u_i} \quad (4.26)$$

Abbildung 4.36 zeigt anhand mehrerer Beispiele, wie diese Formel die natürlichen Zahlen in Segmente zerfallen lässt, in denen jedes Quellsymbol genauso häufig vorkommt, wie es seiner Auftrittswahrscheinlichkeit entspricht. Die Segmente sind dabei so aufgebaut, dass jede natürliche Zahl genau einem der Quellsymbole zugeordnet ist; wir finden also niemals eine Spalte in der Codierungstabelle, die mit mehr als einem Quellsymbol markiert ist. Diese Eigenschaft gestattet uns, die durchgeführte Codierung eindeutig zurückzuverfolgen, oder, in einer anderen Formulierung: Die Nachricht eindeutig zu dekodieren.



#### ■ Quelle 1

	Segment 1										Segment 2										...
$x_i$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...
$u_i = a$	0										1										...
$u_i = b$		0	1									2	3								...
$u_i = c$			0	1	2	3	4	5	6				7	8	9	10	11	12	13		...

#### ■ Quelle 2

	Segment 1										Segment 2										...
$x_i$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...
$u_i = a$	0	1									2	3									...
$u_i = b$			0	1	2								3	4	5						...
$u_i = c$				0	1	2	3	4						5	6	7	8	9		...	

#### ■ Quelle 3

	Segment 1										Segment 2										...
$x_i$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...
$u_i = a$	0	1	2								3	4	5								...
$u_i = b$			0	1	2								3	4	5						...
$u_i = c$				0	1	2	3							4	5	6	7				...

Abb. 4.36: rANS-Codierungstabellen für ein dreielementiges Quellenalphabet  $\Sigma = \{a, b, c\}$

## 4.6.2 uABS-Codierung

Die rANS-Codierung ist mit einem Nachteil verbunden, über den wir weiter oben etwas ungeniert hinweggegangen sind. Er geht auf die Auftrettswahrscheinlichkeiten der Symbole zurück, die wir in der Form

$$p(\sigma_i) = \frac{F_{\sigma_i}}{M}$$

angegeben hatten. Wir haben damit implizit gefordert, dass alle Auftrettswahrscheinlichkeiten als ein Bruch mit einem gemeinsamen Nenner  $M$  dargestellt werden können. Zwar können wir jede Wahrscheinlichkeitsverteilung durch die Vergrößerung von  $M$  beliebig genau approximieren, doch dies hat Nachteile in der Praxis. Da mit  $M$  auch die Größe der Segmente steigt, innerhalb derer die Symbole des Quellenalphabets entsprechend ihrer Auftrettswahrscheinlichkeit verteilt werden, nähert sich das Verfahren asymptotisch viel langsamer dem Optimum an. Das bedeutet, dass wir für kürzere Nachrichten damit rechnen müssen, eine suboptimale Kompressionsrate zu erzielen.

Ist  $\Sigma$  ein binäres Alphabet, wie es in unserem Anfangsbeispiel der Fall war, so können wir dieses Problem mit der uABS-Codierung umgehen. Die Idee dieser Codierung besteht darin, die natürlichen Zahlen nicht mehr länger in Segmente einer bestimmten Länge zu zergliedern, sondern die Symbole möglichst gleichmäßig auf dem Zahlenstrahl zu verteilen. Konkret bedeutet dies das Folgende: Ist  $\Sigma = \{0, 1\}$  ein binäres Quellenalphabet und die Auftrettswahrscheinlichkeit  $p(0)$  des Symbols 0 gleich  $p$ , dann ist  $p(1) = 1 - p$  die Auftrettswahrscheinlichkeit des Symbols 1. In diesem Fall erhalten wir genau dann eine gleichmäßige Verteilung der Symbole, wenn wir in einem Segment der Länge  $N$  immer ungefähr  $p \cdot N$  mal das Symbol 0 vorfinden und ungefähr  $(1 - p) \cdot N$  mal das Symbol 1.

Eine solche Verteilung liefert uns beispielsweise die folgende Codierungsfunktion:

$$C(u_i, x_{i-1}) := \begin{cases} \left\lceil \frac{x_{i-1}+1}{1-p} \right\rceil - 1 & \text{falls } u_i = 0 \\ \left\lfloor \frac{x_{i-1}}{p} \right\rfloor & \text{falls } u_i = 1 \end{cases} \quad (4.27)$$

Abbildung 4.37 zeigt am Beispiel verschiedener Wahrscheinlichkeitsverteilungen, wie die von dieser Formel erzeugten Codierungstabellen aussehen.

Abschließend wollen wir die Codierungstabellen in eine kompaktere Form überführen, die im Zusammenhang mit der ANS-Codierung eben-

Quelle 1

$$\begin{matrix} 0 & 1 \\ \frac{3}{7} & \frac{4}{7} \end{matrix}$$

Quelle 2

$$\begin{matrix} 0 & 1 \\ \frac{3}{10} & \frac{7}{10} \end{matrix}$$

Quelle 3

$$\begin{matrix} 0 & 1 \\ \frac{5}{13} & \frac{8}{13} \end{matrix}$$

Quelle 4

$$\begin{matrix} 0 & 1 \\ \frac{11}{17} & \frac{6}{17} \end{matrix}$$

■ Quelle 1

$x_i$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...
$u_i = 0$	0	0	1		2	3		4		5	6	7		8		9		10		11	...
$u_i = 1$	0		1	2		3	4		5		6	7		8		9		10		11	...

■ Quelle 2

$x_i$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...
$u_i = 0$	0	0	1		2	3		4	5	6		7	8		9	10		11	12	13	...
$u_i = 1$	0		1		2		3		4		5		6		7		8		9		...

■ Quelle 3

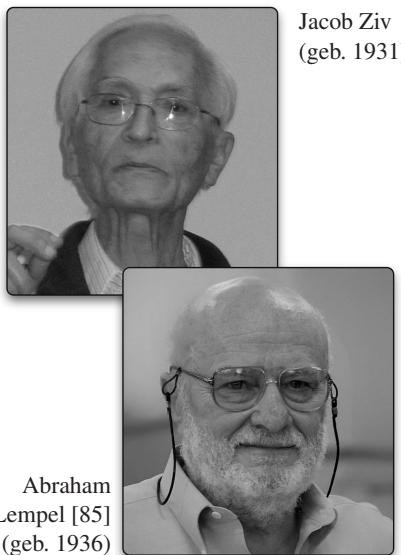
$x_i$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...
$u_i = 0$	0		0	1	2		3		4	5		6	7		8		9	10		11	...
$u_i = 1$	0		1		2		3		4		5		6		7		8		9		...

■ Quelle 4

$x_i$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...
$u_i = 0$		0		1		2		3		4		5		6		7		8		9	...
$u_i = 1$	0	1		2	3		4	5		6	7		8	9		10		11	12		...

Abb. 4.37: uABS-Codierungstabellen für verschiedene Wahrscheinlichkeitsverteilungen

	Quelle 1		Quelle 2		Quelle 3		Quelle 4		
	0	1	0	1	0	1	0	1	
Quelle 1									$\frac{3}{7}$
Quelle 2									$\frac{3}{10}$
Quelle 3									$\frac{5}{13}$
Quelle 4									$\frac{11}{17}$
									$\frac{4}{7}$
									$\frac{7}{10}$
									$\frac{8}{13}$
									$\frac{6}{17}$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$
									$\dots$



**Abb. 4.39:** Jacob Ziv und Abraham Lempel sind die geistigen Väter der Substitutionscodierung.

## 4.7 Substitutionscodierungen

Substitutionscodierungen basieren auf der Idee, Symbolfragmente, die in einer Nachricht mehrfach vorkommen, in Form von Referenzen zu speichern. Die Kompressionsraten, die auf diese Weise erzielt werden können, hängen dabei stark von der Struktur der vorgelegten Nachricht ab: Kehren lange Symbolfragmente häufig wieder, so produzieren Substitutionscodierer in der Regel kompakte Ausgaben. Sind Wiederholungen selten oder die wiederkehrenden Fragmente kurz, so sind die Kompressionsraten meistens gering. Unter anderem sind Substitutionscodierungen für die Kompression natürlichsprachlicher Texte geeignet. Dort ist die Wahrscheinlichkeit hoch, dass sich Worte, Wortfolgen oder ganze Satzpassagen regelmäßig wiederholen.

Trotz ihrer genauso einsichtigen wie naheliegenden Grundidee wurde die erste Substitutionscodierung deutlich später entwickelt als die zahlreichen Entropiecodierungen, die Ende der 40er-Jahre entstanden sind. Den Grundstein legten Jacob Ziv und Abraham Lempel im Jahr 1977 in der Mai-Ausgabe des Magazins *IEEE Transactions on Information Theory* (Abbildung 4.39 und 4.40). Das dort publizierte LZ77-

IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. IT-23, NO. 3, MAY 1977

337

**A Universal Algorithm for Sequential Data Compression**

JACOB ZIV, FELLOW, IEEE, AND ABRAHAM LEMPEL, MEMBER, IEEE

**Abstract—**A universal algorithm for sequential data compression is presented. Its performance is investigated with respect to a nonprobabilistic model of constrained sources. The compression ratio achieved by the proposed universal code uniformly approaches the lower bounds on the compression ratios attainable by block-to-variable codes and variable-to-block codes designed to match a completely specified source.

I. INTRODUCTION

IN MANY situations arising in digital communications and data processing, the encountered strings of data display various structural regularities or are otherwise subject to certain constraints, thereby allowing for storage and time-saving techniques of data compres-

then show that the efficiency of our universal code with no *a priori* knowledge of the source approaches those bounds.

The proposed compression algorithm is an adaptation of a simple copying procedure discussed recently [10] in a study on the complexity of finite sequences. Basically, we employ the concept of encoding future segments of the source-output via maximum-length copying from a buffer containing the recent past output. The transmitted codeword consists of the buffer address and the length of the copied segment. With a predetermined initial load of the buffer and the information contained in the codewords, the source data can readily be reconstructed at the decoding end of the process.

**Abb. 4.40:** Mit dem LZ77-Verfahren legten Jacob Ziv und Abraham Lempel den Grundstein für die Entwicklung zahlreicher Substitutionscodierungen. Viele bekannte Softwarewerkzeuge, die wir im Computeralltag zur Kompression von Datenbeständen verwenden, basieren auf dem gleichen oder einem ähnlichen Arbeitsprinzip.

*Verfahren* markiert den Anfang einer längeren Reihe von Substitutionscodierungen, die in den Folgejahren um die Verfahren LZSS, LZ78, LZW und mehrere andere Derivate ergänzt wurde.

### 4.7.1 Lempel-Ziv-77-Kompression

*„Basically, we employ the concept of encoding future segments of the source-output via maximum-length copying from a buffer containing the recent past output. The transmitted codeword consists of the buffer address and the length of the copied segment.“*

Jacob Ziv, Abraham Lempel [108]

Die LZ77-Kompression wird mit einem zweigeteilten Schieberegister durchgeführt, wie es schematisch in Abbildung 4.41 zu sehen ist. Der linke Teil ist das *Suchfenster (search window)* und der rechte Teil der *Vorschaupuffer (look-ahead buffer)*. Später wird im Suchfenster jener Ausschnitt der Nachricht gespeichert, in dem nach übereinstimmenden Fragmenten gesucht wird, und der Vorschaupuffer wird diejenigen Symbole enthalten, die als Nächstes bearbeitet werden.

Um eine Nachricht zu komprimieren, wird sie im ersten Schritt so weit von rechts in das Register geschoben, bis der Vorschaupuffer vollständig gefüllt ist. Das Suchfenster enthält zu diesem Zeitpunkt noch keine Nachrichtensymbole und wird mit einem vorher festgelegten Symbol des Quellenalphabets  $\Sigma$  initialisiert. Abbildung 4.42 (oben) demonstriert das Gesagte anhand der folgenden Nachricht:

001010210210212021021200

Dies ist die gleiche Nachricht, die in der Originalarbeit aus dem Jahr 1977 verwendet wurde. Für die Initialisierung des Suchfensters hatten Jacob Ziv und Abraham Lempel das Symbol 0 gewählt, und auch unser Schieberegister ist mit diesem Symbol vorbeschrieben.

Der LZ77-Algorithmus führt nun mehrere Iterationen durch, in denen er jeweils versucht, ein möglichst großes Anfangsstück des Vorschaupuffers durch eine Referenz in das Suchfenster abzudecken. Jede Referenz wird dabei durch ein Tripel der Form

$< i, n, \sigma >$

codiert. Hierin ist  $i$  ist der Index im Suchfenster, an dem das referenzierte Fragment beginnt,  $n$  die Anzahl der übereinstimmenden Symbole

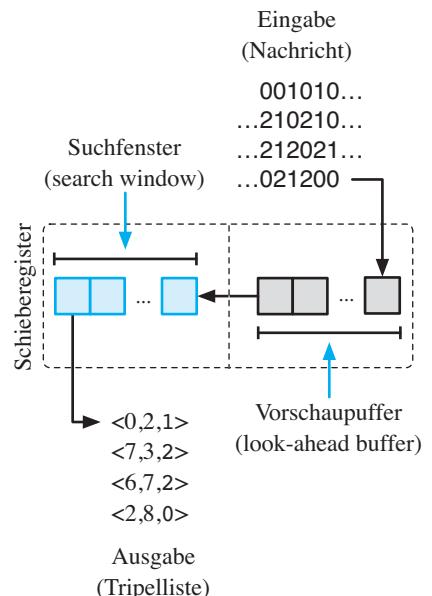


Abb. 4.41: Aufbau eines Schieberegisters für die LZ77-Kompression

## Ausgangskonfiguration



## Schieben



## Codieren



## Schieben



## Codieren



## Schieben



Handout 1



## Schieben



 Codieren



## Schieben



☞ Der Vorschaupuffer ist leer und der Algorithmus terminiert

**Abb. 4.42:** Schrittweise Durchfhrung der LZ77-Kompression am Originalbeispiel von Jacob Ziv und Abraham Lempel

und  $\sigma$  das Folgezeichen, mit dem die Nachricht im Vorschaupuffer fortgesetzt wird.

In unserem Beispiel aus Abbildung 4.42 beginnt der Vorschaupuffer mit dem Fragment 001. Die ersten beiden Symbole finden wir auch im Suchfenster wieder, sodass der Algorithmus das folgende Tripel erzeugt:

$$<0,2,1>$$

Beachten Sie, dass die Wahl der Indexpositionen an dieser Stelle nicht eindeutig ist: Der Algorithmus hätte genauso gut eines der folgenden Tripel wählen können:

$$<1,2,1>, <2,2,1>, \dots, <8,2,1>$$

Das Tripel  $<8,2,1>$  verdient unsere besondere Aufmerksamkeit. Auf den ersten Blick scheint es in der Liste nichts zu suchen zu haben, da es nicht nur das letzte Zeichen im Suchfenster, sondern auch das erste Zeichen im Vorschaupuffer kopiert. Das bedeutet, dass dieses Tripel einen Bereich referenziert, den wir noch gar nicht vollständig bearbeitet haben. Warum wir in solchen Situationen keinen Fehler machen, verrät Abbildung 4.43. Dort wird sichtbar, dass uns jedes Tripel selbst darüber Auskunft gibt, wie die Zeichenfolge im Vorschaupuffer weiter geht.

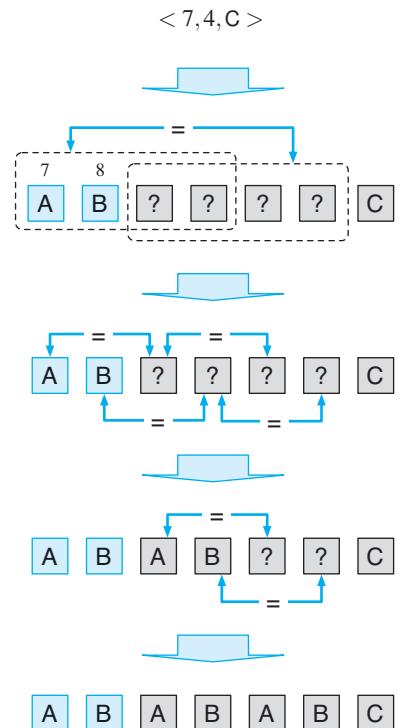
In unserem Beispiel referenziert das Tripel  $<8,2,1>$  die achte Stelle des Suchfensters. Dort befindet sich das Symbol 0, und das bedeutet, dass der Vorschaupuffer mit genau diesem Symbol anfangen muss. Folglich können wir eine Sequenz, die aus dem Suchfenster kopiert wird, ruhigen Gewissens in den Vorschaupuffer verlängern; alle darin enthaltenen Symbole sind eindeutig bestimmt.

Nachdem der Encoder das Tripel  $<0,2,1>$  ausgegeben hat, wird das bearbeitete Fragment aus dem Vorschaupuffer nach links in das Suchfenster geschoben. Danach werden die rechts frei werdenden Registerstellen mit neuen Nachrichtensymbolen aufgefüllt, und der geschilderte Vorgang beginnt von vorne. Der Algorithmus terminiert, wenn der Vorschaupuffer keine Symbole mehr enthält; die Nachricht ist dann vollständig komprimiert.

Für unsere Beispieldsequenz terminiert der Algorithmus nach vier Schritten und produziert die folgende Tripelfolge:

$$<0,2,1><7,3,2><6,7,2><2,8,0> \quad (4.28)$$

Damit das geschilderte Verfahren in der Praxis sinnvoll eingesetzt werden kann, müssen wir für die erzeugten Tripel eine möglichst kompakte



**Abb. 4.43:** Sequenzen, die aus dem Suchfenster kopiert werden, dürfen in den Vorschaupuffer hineinragen. Die Rekonstruktion ist auch diesen Fällen eindeutig möglich.

Darstellung finden. Ziv und Lempel schlugen vor, das Quellenalphabet gleichzeitig als Codealphabet zu verwenden und jedes Tripel mit  $j + k + 1$  Symbolen darzustellen. Die ersten  $j$  Symbole codieren die Indexposition, die nächsten  $k$  Symbole die Anzahl der zu kopierenden Zeichen, und das letzte Symbol ist das Folgezeichen  $\sigma$ . Ist  $b$  die Anzahl der Quellsymbole, so wird eine Sequenz aus  $j$  Symbolen ganz einfach als eine Zahl in  $b$ -adischer Darstellung interpretiert, d. h. als eine Zahl zur Basis  $b$ . In unserem Beispiel ist  $b = 3$ , da das Quellenalphabet aus den drei Elementen 0, 1 und 2 besteht. Werden für die Darstellung der Indexposition und der Fragmentlänge jeweils 2 Symbole verwendet, ist also  $j = k = 2$ , so entspricht die Folge

$$\underbrace{21}_{i} \quad \underbrace{10}_{n} \quad \underbrace{2}_{\sigma}$$

dem Tripel  $< i, n, \sigma >$  mit

$$\begin{aligned} i &= 2 \cdot 3^1 + 1 \cdot 3^0 = 7 & (\text{☞ } 21102) \\ n &= 1 \cdot 3^1 + 0 \cdot 3^0 = 3 & (\text{☞ } 21102) \end{aligned}$$

Auf die besprochene Weise lässt sich die gesamte Tripelfolge

$$<0, 2, 1> <7, 3, 2> <6, 7, 2> <2, 8, 0> \quad (4.29)$$

codieren. Übersetzen wir jeweils die ersten beiden Tripelkomponenten in ihre *triadische Darstellung*

$$\begin{aligned} &<0 \cdot 3^1 + 0 \cdot 3^0, 0 \cdot 3^1 + 2 \cdot 3^0, 1> <2 \cdot 3^1 + 1 \cdot 3^0, 1 \cdot 3^1 + 0 \cdot 3^0, 2> \\ &<2 \cdot 3^1 + 0 \cdot 3^0, 2 \cdot 3^1 + 1 \cdot 3^0, 2> <0 \cdot 3^1 + 2 \cdot 3^0, 2 \cdot 3^1 + 2 \cdot 3^0, 0>, \end{aligned}$$

so können wir die codierte Nachricht sofort ablesen. Es ist die Folge

00021211022021202220

Die Einfärbung der Symbole ist lediglich der Übersichtlichkeit geschuldet. Aus der Position eines Zeichens lässt sich eindeutig ablesen, zu welchem Tripel es gehört und welche Komponente es codiert.

Beachten Sie, dass die Werte  $j$  und  $k$ , d. h. die Anzahl der Symbole, die für die Codierung der Indexpositionen und der Fragmentlänge verwendet werden, die Größe des Suchfensters und des Vorschaupuffers limitieren. Ist  $b$ , wie oben, die Anzahl der Quellsymbole, so können nur Indexpositionen zwischen 0 und  $b^j - 1$  sowie Fragmentlängen zwischen 0 und  $b^k - 1$  codiert werden. In unserem Beispiel ist  $b^j = b^k = 9$ , und dies erklärt, warum das Suchfenster und der Vorschaupuffer genau diese Größe besitzen.

Die von Ziv und Lempel vorgeschlagene Tripelcodierung ist nicht die einzige mögliche. Anstatt die Nachricht mit den Symbolen des Quellenalphabets darzustellen, können wir genauso gut ein binäres Codealphabet verwenden. Hierzu brauchen wir lediglich den Index und die Fragmentlänge in gewöhnliche Dualzahlen und das Folgezeichen  $\sigma$  in ein binäres Codewort  $c(\sigma) \in \{0, 1\}^+$  zu übersetzen. Wählen wir für  $c$  beispielsweise die Zuordnung  $0 \mapsto 00$ ,  $1 \mapsto 01$  und  $2 \mapsto 10$ , so können wir die Tripelfolge (4.28) zunächst in

$<0000,0010,01><\textcolor{blue}{0111},\textcolor{blue}{0011},\textcolor{blue}{10}>$   
 $<0110,\textcolor{blue}{0111},\textcolor{blue}{10}><\textcolor{blue}{0010},\textcolor{blue}{1000},\textcolor{blue}{00}>$

umschreiben und danach in die folgende Bitsequenz übersetzen:

00000010010111001100111100010100000

Auch zu dieser Art der Darstellung existieren Alternativen. Eine heute weit verbreitete wurde im Jahr 1982 von James Storer und Thomas Szymanski vorgeschlagen. Sie ist die Grundlage des LZSS-Verfahrens, das zu den bekanntesten Derivaten von LZ77 zählt.

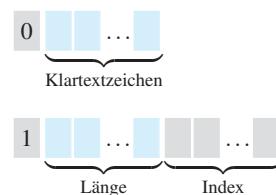
### 4.7.2 LZSS-Kompression

Bei der LZSS-Kompression werden die LZ77-Tripel in zwei Komponenten aufgeteilt. Die eine Komponente ist die Referenz, bestehend aus dem Index und der Fragmentlnge, und die andere das im Klartext gespeicherte Folgezeichen. Fr die Codierung der beiden Komponenten existieren separate Bitformate, die nach Belieben gemischt werden knnen (Abbildung 4.44). Zur Erkennung des Formattyps dient ein vorangestelltes *Indikatorbit*. Ist es gleich 0, so folgt ein Klartextzeichen; ist es gleich 1, so wird das nachfolgende Bitmuster als ein Index und eine Fragmentlnge interpretiert.

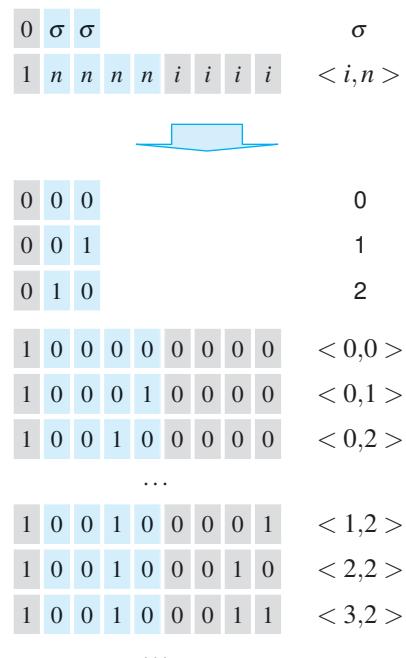
Die Umsetzung der Tripelfolge erfolgt nach einer einfachen *Break-even-Strategie*. Anstatt eine Referenz und ein Folgezeichen, wie von Ziv und Lempel vorgeschlagen, immer gemeinsam zu codieren, werden Referenzen nur noch dann verwendet, wenn hierdurch eine Verkürzung erreicht wird. Ist dies nicht der Fall, wird die Referenz aufgelöst und das Fragment Zeichen für Zeichen im Klartext codiert.

In unserem Beispiel, das wir oben für die Erklärung des LZ77-Verfahrens benutzt haben, wurde ein Zeichen mit 2 Bit und die Indexposition sowie die Fragmentlänge mit jeweils 4 Bit codiert. Zusammen

#### ■ LZSS-Formatvarianten



### ■ Beispiel ( $\Sigma = \{0, 1, 2\}$ )



**Abb. 4.44:** Im LZSS-Format können Klartextzeichen und Referenzen gemischt werden. Eine Referenz wird erst dann erzeugt, wenn hierdurch ein kürzeres Bitmuster entsteht. Andernfalls werden die Symbole im Klartext codiert

mit den Indikatorbits werden im LZSS-Format deshalb 3 Bit für die Codierung eines Zeichens und 9 Bit für die Codierung einer Referenz benötigt. Den *Break-even-Punkt* erreichen wir bei 3 Zeichen. Wird ein Fragment kopiert, das mehr als 3 Zeichen umfasst, so lohnt sich die Speicherung als Referenz. Ist ein Fragment kürzer, so benötigt die Speicherung im Klartextformat weniger Platz. Besteht ein Fragment aus 3 Zeichen, so spielt es in unserem Beispiel keine Rolle, welches Format wir verwenden. Die Speicherung von drei Klartextzeichen führt zu einer Bitsequenz der Länge 9, genauso wie die Speicherung einer Referenz.

Damit ist klar, wie die Tripelfolge (4.29) im LZSS-Format dargestellt wird. Zunächst werden alle Referenzen eliminiert, die auf Fragmente verweisen, die aus 3 oder weniger Zeichen bestehen:

00010<6,7>2<2,8>0

Das Ergebnis lässt sich jetzt direkt in das LZSS-Format übersetzen:

000	000	000	001	000	101110110	010	110000010	000
0	0	0	1	0	<6,7>	2	<2,8>	0

Ein grundlegendes Problem, das alle verlustfreien Kompressionsverfahren miteinander teilen, kann auch die LZSS-Codierung nicht lösen: Neben den vielen Nachrichten, die verkürzt werden, existieren immer auch welche, die durch die Codierung verlängert werden. Bei der LZSS-Codierung brauchen wir nach solchen Nachrichten gar nicht lange zu suchen. Wird das Verfahren auf eine Nachricht angewendet, in der sich kein einziges Fragment wiederholt, so bleibt dem Algorithmus nichts anderes übrig, als alle Zeichen im Klartextformat zu codieren. Besteht die Nachricht aus  $n$  Symbolen, so wird sie durch die zusätzliche Speicherung der Indikatorbits um  $n$  Bit verlängert.

### 4.7.3 Lempel-Ziv-78-Kompression

„For the encoder  $E$ , we employ an ILF finite-state machine that realizes a concatenated coding scheme by combining a fixed block-to-variable outer code with a state-dependent variable-to-variable inner code. The inner code is used to encode sequentially and state dependently growing segments of an input block of relatively large length  $n$ . Upon completion of a block the machine returns to its initial state, thus ‘forgetting’ all past history before starting to encode the next input block. The segments of a block that serve as input words

*of the inner code are determined according to a so-called incremental parsing procedure. This procedure is sequential, and it creates a new phrase as soon as a prefix of the still unparsed part of the string differs from all preceding phrases.“*

Jacob Ziv, Abraham Lempel [109]

Ein Jahr nach der Veröffentlichung des LZ77-Verfahrens publizierten Jacob Ziv und Abraham Lempel einen Algorithmus, der dem gleichen Substitutionsgedanken folgt, aber im Detail völlig anders funktioniert [109]. Während das LZ77-Verfahren mit dem Inhalt des Suchfenzters ein Wörterbuch konstanter Länge verwendet, benutzt das LZ78-Verfahren eine dynamisch wachsende Tabelle, die initial leer ist und während der Kompression kontinuierlich erweitert wird. Der LZ78-Encoder arbeitet, genau wie der LZ77-Encoder, iterativ. Er durchsucht die Tabelle in jedem Schritt nach übereinstimmenden Fragmenten und produziert als Ausgabe eine Referenz. Anders als bei LZ77 wird eine Referenz aber nicht als Tripel, sondern als ein Paar

$$\langle i, \sigma \rangle$$

dargestellt. Der Index  $i$  ist die Tabellenposition des referenzierten Fragments und  $\sigma$  das Folgezeichen, mit dem die Nachricht weitergeht. Die Fragmentlänge muss nicht codiert werden, da diese Information implizit im Wörterbuch gespeichert ist. Dem Index 0 kommt im LZ78-Verfahren eine besondere Bedeutung zu: Er wird immer dann eingesetzt, wenn in der Tabelle keine übereinstimmende Sequenz gefunden werden konnte.

Wie die LZ78-Kompression im Detail abläuft, wollen wir anhand der folgenden Nachricht herausarbeiten:

ABABC BABABAAAAAA

Das Beispiel geht auf den amerikanischen Computerwissenschaftler Terry Welch zurück, der es im Jahr 1984 für die Erklärung des LZW-Verfahrens verwendete [101]. Hierbei handelt es sich um eine trickreiche Weiterentwicklung des LZ78-Verfahrens, das wir detailliert in Abschnitt 4.7.4 besprechen.

Das erste Zeichen unserer Nachricht ist ein A. Da wir mit einem leeren Wörterbuch beginnen, taucht es noch nicht in der Tabelle auf und muss notgedrungen mit dem speziellen Index 0 codiert werden. Der Encoder produziert also das Paar  $\langle 0, A \rangle$  und fügt das Symbol A gleichzeitig an Position 1 in die Tabelle ein (Abbildung 4.45 oben). Der nächste

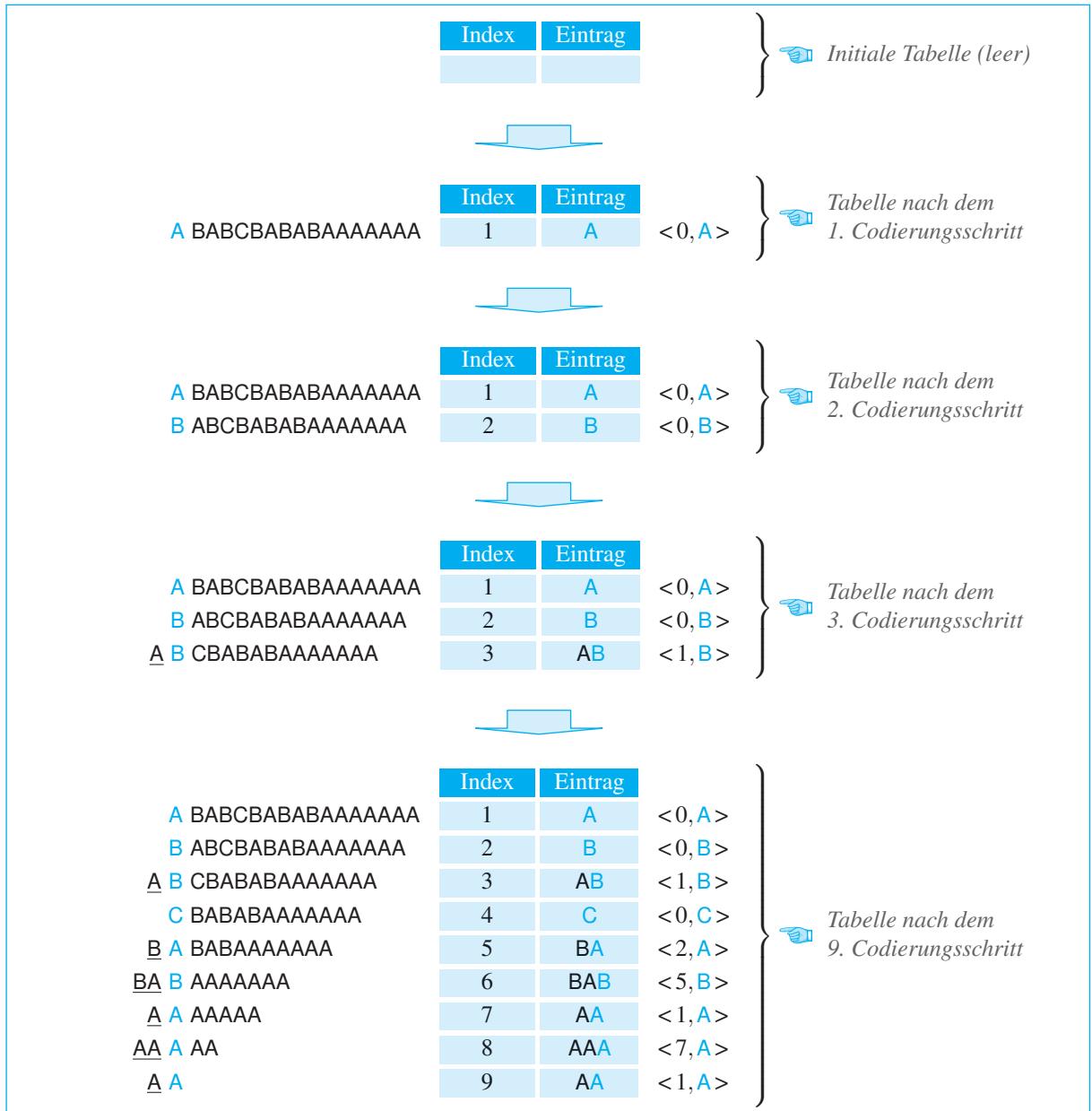


Abb. 4.45: Schrittweise Durchführung der LZ78-Kompression am Beispiel der Nachricht ABABCBABABAAAAAAA

Schritt verläuft analog. Da noch kein Tabelleneintrag existiert, der mit dem Zeichen B beginnt, produziert der Encoder das Paar  $<0, B>$  und erweitert die Tabelle um das Symbol B. Das dritte Symbol in unserer Beispielnachricht ist erneut ein A. Jetzt produziert der Encoder eine Referenz und gibt das Paar  $<1, B>$  aus. Im Klartext codiert dieses Paar die Sequenz AB und genau um diese wird auch die Tabelle erweitert. Auf die beschriebene Weise wird die Codierung nun so lange fortgesetzt, bis das letzte Zeichen der Nachricht bearbeitet wurde.

Für unser Beispiel terminiert das Verfahren nach 9 Schritten mit der folgenden Ausgabe:

$<0, A><0, B><1, B><0, C><2, A><5, B><1, A><7, A><1, A>$

Im nächsten Schritt geht es darum, diese Folge kompakt darzustellen. Die Folgezeichen können wir, genau wie bei der LZ77-Kompression, mit einem Binärkode fester Länge codieren, z. B. mithilfe der folgenden Zuordnung: A  $\mapsto$  00, B  $\mapsto$  01, C  $\mapsto$  10.

Bei der Codierung der Tabellenpositionen müssen wir beachten, dass die Tabelle dynamisch anwächst und die Indizes hierdurch immer größer werden können. Um eine geeignete Darstellung zu finden, hilft die folgende Beobachtung: Der Index des ersten Paars ist immer 0, der Index des zweiten Paars ist 0 oder 1, der Index des dritten Paars ist 0, 1 oder 2 und so fort. Verallgemeinert gilt:



Bei der LZ78-Kompression liegt der Index des n-ten Paars zwischen 0 und  $n - 1$ .

Folgerichtig ist es ausreichend, den Index des  $n$ -ten Paars mit  $\lceil \log_2 n \rceil$  Bits zu codieren, und genau diesen Weg schlägt das LZ78-Verfahren ein (Abbildung 4.46).

Die oben erzeugte Paarfolge erhält dann die folgende Darstellung:

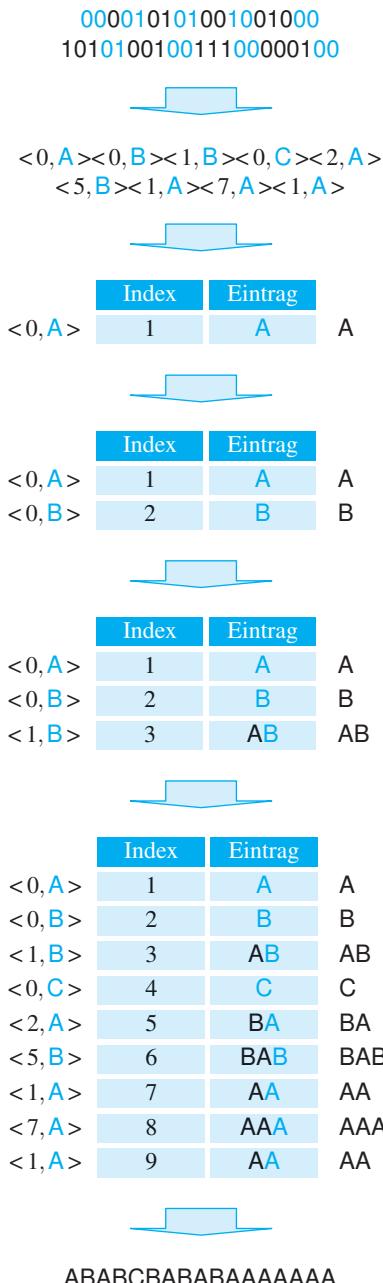
$\underbrace{00}_{<0,A>} \underbrace{001}_{<0,B>} \underbrace{0101}_{<1,B>} \underbrace{0010}_{<0,C>} \underbrace{01000}_{<2,A>} \underbrace{10101}_{<5,B>} \underbrace{00100}_{<1,A>} \underbrace{11100}_{<7,A>} \underbrace{0000100}_{<1,A>}$

Es ist eine Stärke der LZ78-Kompression, dass das Wörterbuch, obwohl es dynamisch erzeugt wurde und deshalb von Nachricht zu Nachricht variiert, nicht übertragen werden muss. Alle Tabelleneinträge sind durch die produzierte Paarfolge eindeutig bestimmt und lassen sich

$< \underbrace{i}, \underbrace{\sigma} >$   
 $\underbrace{\text{...}}_{\text{Index}} \underbrace{\text{...}}_{\text{Format}}$

	Index- bereich	Format
1. Paar:	0	$\text{... } \text{...}$
2. Paar:	0 ... 1	$\text{... } \text{... } \text{...}$
3. Paar:	0 ... 2	$\text{... } \text{... } \text{... } \text{...}$
4. Paar:	0 ... 3	$\text{... } \text{... } \text{... } \text{... }$
5. Paar:	0 ... 4	$\text{... } \text{... } \text{... } \text{... } \text{...}$
6. Paar:	0 ... 5	$\text{... } \text{... } \text{... } \text{... } \text{... }$
7. Paar:	0 ... 6	$\text{... } \text{... } \text{... } \text{... } \text{... } \text{...}$
8. Paar:	0 ... 7	$\text{... } \text{... } \text{... } \text{... } \text{... } \text{... }$
9. Paar:	0 ... 8	$\text{... } \text{... } \text{... } \text{... } \text{... } \text{... } \text{...}$
10. Paar:	0 ... 9	$\text{... } \text{... } \text{... } \text{... } \text{... } \text{... } \text{... }$
11. Paar:	0 ... 10	$\text{... } \text{... } \text{... } \text{... } \text{... } \text{... } \text{... }$
12. Paar:	0 ... 11	$\text{... } \text{... } \text{... } \text{... } \text{... } \text{... } \text{... }$
13. Paar:	0 ... 12	$\text{... } \text{... } \text{... } \text{... } \text{... } \text{... } \text{... }$
14. Paar:	0 ... 13	$\text{... } \text{... } \text{... } \text{... } \text{... } \text{... } \text{... }$
15. Paar:	0 ... 14	$\text{... } \text{... } \text{... } \text{... } \text{... } \text{... } \text{... }$
16. Paar:	0 ... 15	$\text{... } \text{... } \text{... } \text{... } \text{... } \text{... } \text{... }$
17. Paar:	0 ... 16	$\text{... } \text{... } \text{... } \text{... } \text{... } \text{... } \text{... }$
...		

**Abb. 4.46:** Im LZ78-Format wächst die Anzahl der Bits, die für die Codierung der Indizes verwendet wird, dynamisch an.



**Abb. 4.47:** Das Wörterbuch muss bei der LZ78-Kompression nicht übertragen werden. Es wird bei der Decodierung automatisch rekonstruiert.

während der Decodierung sehr einfach rekonstruieren. Um dies einzusehen, wollen wir versuchen, die eben erzeugte Paarfolge

$<0, A>$   $<0, B>$   $<1, B>$   $<0, C>$   $<2, A>$

$<5, B>$   $<1, A>$   $<7, A>$   $<1, A>$

schrittweise zurückzuübersetzen (Abbildung 4.47). Im ersten Schritt bearbeitet der Decoder das Paar  $<0, A>$ : Er gibt das Zeichen A aus und fügt es gleichzeitig an der ersten Position in die noch leere Tabelle ein. Das Gleiche passiert bei der Decodierung von  $<0, B>$ : Das Zeichen B wird ausgegeben und die Tabelle um dieses Zeichen erweitert. Die Tabelle enthält jetzt zwei Einträge und sieht genauso aus wie die Tabelle des Encoders nach der zweiten Iteration. Im dritten Schritt wird das Paar  $<1, B>$  decodiert. Hierzu liest der Decoder die Tabelle an der Position 1 aus und ergänzt das dort gefundene Fragment um das Symbol B. Als Ergebnis entsteht die Zeichenkette AB. Der Encoder gibt die Zeichenkette aus und erzeugt für sie anschließend einen neuen Tabelleneintrag. Spätestens jetzt ist klar, dass bei der Decodierung die gleiche Tabelle entsteht wie bei der Codierung.

Führen wir die Rückübersetzung weiter aus, so erhalten wir nach 9 Schritten die ursprüngliche Nachricht zurück.

#### 4.7.4 Lempel-Ziv-Welch-Kompression

*„The compression procedure discussed here is called the LZW method. A variation on the Lempel-Ziv procedure, it retains the adaptive properties of Lempel-Ziv and achieves about the same compression ratios in normal commercial computer applications. LZW is distinguished by its very simple logic, which yields relatively inexpensive implementations and the potential for very fast execution.“*

Terry Welch [101]

Die LZW-Kompression ist eine Variante der LZ78-Kompression, mit der Terry Welch im Jahr 1984 ein ganz praktischen Problem anging. Welch versuchte, den LZ78-Algorithmus so zu modifizieren, dass er sich effizient in Hardware oder Software implementieren ließ. Während seiner Suche nach einem geeigneten Algorithmus machte er eine interessante Entdeckung: Ihm fiel auf, dass auf die Speicherung der Klartextzeichen verzichtet werden kann, wenn die Kompression mit einer

initialisierten Tabelle beginnt, und nicht, wie bei LZ78, mit einer leeren. Die Ausgabe des LZW-Algorithmus besteht daher nur noch aus Zahlen, die allesamt Tabellenindizes sind.

Als Beispiel betrachten wir die Nachricht aus der Originalarbeit von Welch [101]. Es ist dieselbe Nachricht, mit der wir weiter oben die Durchführung der LZ78-Kompression erklärt haben:

ABABCBABABAAAAAAA

Der LZW-Encoder beginnt mit einer Tabelle, die für jedes der drei Quellsymbole einen Eintrag enthält (Abbildung 4.48 oben). Beachten Sie, dass die Kompression in der Praxis zumeist auf der Byteebene durchgeführt wird und die Eingaben dann als Sequenzen über einem Quellenalphabet mit 256 Elementen interpretiert werden. In diesem Fall würde dann auch die initiale LZW-Tabelle 256 Einträge enthalten – einen für jeden der 256 möglichen 8-Bit-Werte.

Nach der Initialisierung der Tabelle verläuft die Kompression ähnlich wie bei LZ78. In jedem Schritt sucht der Encoder in der Tabelle nach deckungsgleichen Fragmenten und gibt die Indexposition des Eintrags mit der größten Übereinstimmung aus. Durch die Initialisierung der Tabelle ist sichergestellt, dass der Encoder immer ein passendes Fragment findet und wir auf eine Fallunterscheidung, wie es das LZ78-Verfahren erfordert, verzichten können. Damit können wir auch den Index 0, der im LZ78-Verfahren eine Sonderrolle einnimmt, als ganz normalen Index behandeln. Dies erklärt, warum die erste Zeile einer LZW-Tabelle über den Index 0 adressiert wird, und nicht, wie bei LZ78, über den Index 1.

Unsere Beispieldaten beginnen mit dem Symbol A. Der Encoder findet das Symbol an der Indexposition 0 in der Tabelle wieder und produziert daraufhin die Ausgabe 0. Gleichzeitig wird ein neuer Tabelleneintrag erzeugt, der aus dem referenzierten Fragment (dem Symbol A) und dem nächsten Nachrichtensymbol (dem Symbol B) besteht. Damit ist AB der neue Tabelleneintrag. Im nächsten Schritt wird das B bearbeitet. Der Encoder durchsucht erneut die Tabelle und produziert die Ausgabe 1. Zusätzlich generiert er einen neuen Tabelleneintrag, der wieder aus dem referenzierten Fragment (dem Symbol B) und dem nächsten Nachrichtensymbol (dem Symbol A) besteht. Damit ist BA der neue Tabelleneintrag. Auf diese Weise wird so lange fortgefahrene, bis die Nachricht vollständig bearbeitet ist (Abbildung 4.48 unten). In unserem Beispiel terminiert der Algorithmus nach 10 Schritten mit der Ausgabe

$$0, 1, 3, 2, 4, 7, 0, 9, 10, 0 \quad (4.30)$$

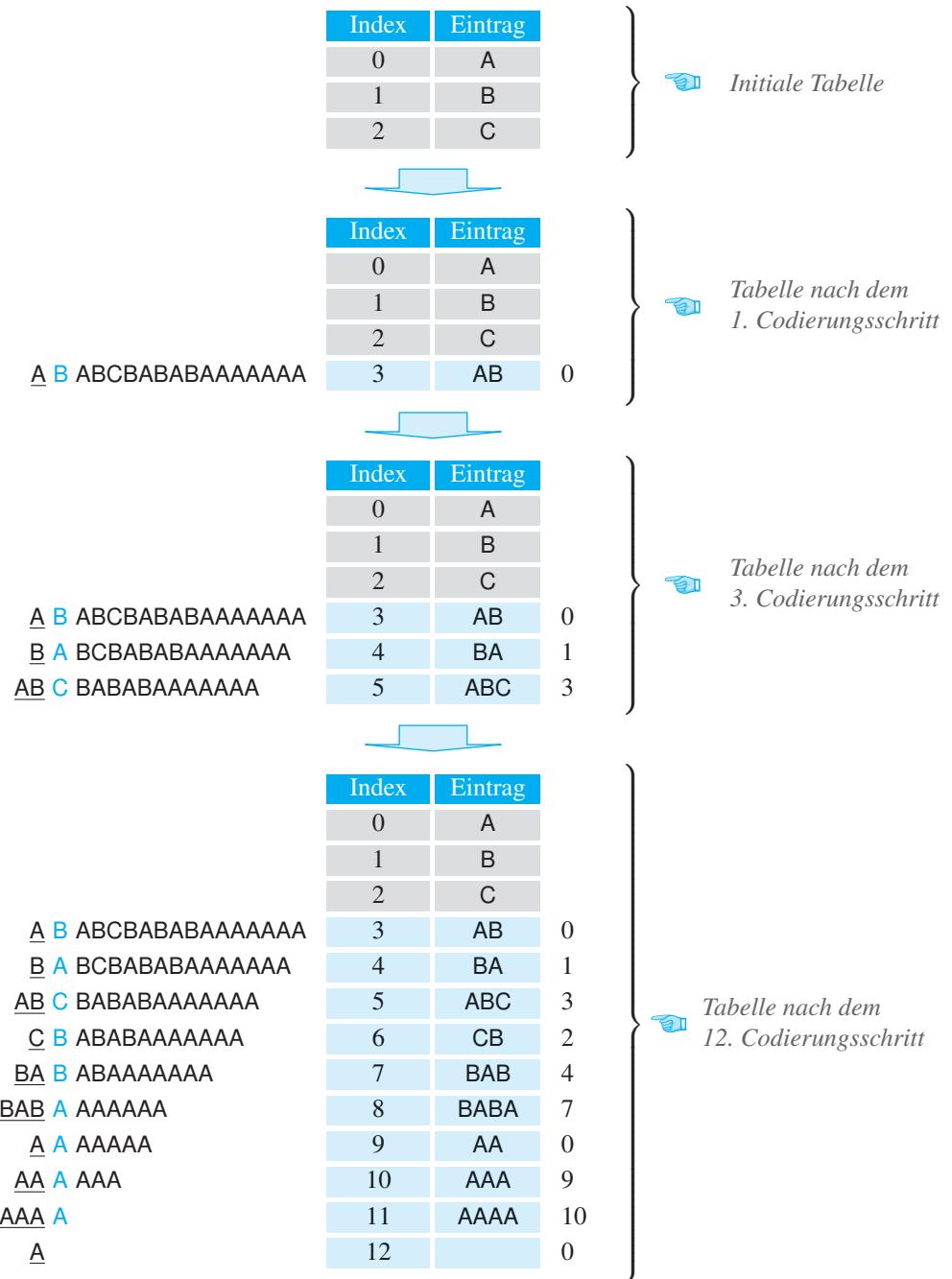


Abb. 4.48: Schrittweise Durchführung der LZW-Kompression am Beispiel der Nachricht ABABCBABABAAAAAAA

Im nächsten Schritt wird die Zahlenfolge in eine Binärsequenz übersetzt, in der die einzelnen Folgenglieder, genau wie bei der LZ78-Kompression, mit einer unterschiedlichen Anzahl an Bits codiert werden. Die Anzahl der Bits, die für die einzelnen Indizes aufgewendet werden, basiert auf der folgenden Beobachtung:



*Bei der LZW-Kompression einer Nachricht  $\mathbf{u} \in \Sigma^+$  liegt die  $n$ -te Referenz zwischen  $0$  und  $|\Sigma| + n - 2$ .*

Dass wir für die Berechnung der Bitbreite, anders als bei der LZ78-Kompression, die Anzahl der Quellsymbole einbeziehen müssen, liegt an der Eigenschaft der LZW-Kompression, mit einer initialisierten Tabelle zu starten. Abbildung 4.49 macht klar, wie wir die Zahlenfolge (4.30) codieren müssen. Wir erhalten die folgende Darstellung:

00 01 011 010 100 111 0000 1001 1010 0000  
0 1 3 2 4 7 0 9 10 0

Als Nächstes wollen wir untersuchen, wie eine LZW-codierte Nachricht auf der Empfängerseite rekonstruiert werden kann. Genau wie bei der LZ78-Kompression verläuft die Decodierung in zwei Schritten. Zunächst wird die Bitsequenz in eine Zahlenfolge zurückübersetzt und diese anschließend in einem iterativen Prozess bearbeitet.

Unsere Aufmerksamkeit müssen wir vor allem auf den zweiten Schritt richten. Der Decoder beginnt, genau wie der Encoder, mit einer Tabelle, die mit den Quellsymbolen initialisiert ist, und bearbeitet dann nacheinander die empfangenen Zahlen. In jedem Schritt wird der referenzierte Wörterbuchinhalt ausgegeben und anschließend ein neuer Eintrag erstellt.

In unserem Beispiel ist die erste Zahl eine 0. Der Decoder gibt daraufhin das Zeichen A aus und erweitert die Tabelle um den Eintrag AB (Abbildung 4.50 oben). An dieser Stelle müssen wir innehalten: Um den neuen Tabelleneintrag zu erstellen, musste der Decoder wissen, dass auf das Zeichen A ein B folgt, doch wie kam er zu dieser Information? Die Antwort ist einfach: Der Decoder hat diese Information durch eine Vorausschau auf die nächste Referenz erhalten. An zweiter Stelle unserer Zahlenfolge befindet sich eine 1, und damit ist klar, dass die Nachricht mit einem B weitergehen muss.

Tatsächlich ist diese Vorausschau ein Schlüsselement des LZW-Verfahrens. Sie ist der Grund dafür, dass wir auf die Speicherung von

Index	Eintrag
0	A
1	B
2	C

Indexbereich	Format
1. Referenz:	0 ... 2
2. Referenz:	0 ... 3
3. Referenz:	0 ... 4
4. Referenz:	0 ... 5
5. Referenz:	0 ... 6
6. Referenz:	0 ... 7
7. Referenz:	0 ... 8
8. Referenz:	0 ... 9
9. Referenz:	0 ... 10
10. Referenz:	0 ... 11

**Abb. 4.49:** LZW-Bitformat für unsere Beispieldaten

Index	Eintrag
0	A
1	B
2	C
0	AB
1	AB
2	ABC
3	CB
4	BAB
5	BABA
6	AA
7	AAA
8	AAAA
9	AA

Klartextzeichen verzichten können. Wie trickreich Welch bei der Konstruktion des Kompressionsverfahrens vorgegangen ist, wird sichtbar, wenn wir mit der Decodierung fortfahren, bis der Tabelleneintrag mit dem Index 7 erzeugt wurde (Abbildung 4.50 Mitte). Um das letzte Zeichen des Wörterbucheintrags zu erstellen, schaut der Decoder auch hier wieder auf die nachfolgende Referenz. In unserem Beispiel verweist diese Referenz auf den Eintrag mit dem Index 7, also genau auf jenen Eintrag, den wir gerade erstellen. Tatsächlich ist dies für den Algorithmus kein Problem. Um das Folgezeichen abzulesen, müssen wir nämlich gar nicht den vollständigen Eintrag kennen, sondern lediglich das erste Zeichen, und dieses ist uns ja bereits bekannt. Wir wissen, dass der Wörterbucheintrag an Position 7 mit BA beginnt und damit ist B das gesuchte Folgezeichen; hängen wir es an BA an, so erhalten wir mit BAB den vollständigen Wörterbucheintrag.

Fahren wir auf die besprochene Weise fort, so ist die ursprüngliche Nachricht nach 10 Schritten vollständig rekonstruiert. Nebenbei hat der Algorithmus das komplette Wörterbuch erzeugt, ohne ein einziges Zeichen im Klartext erhalten zu haben. Diese Eigenschaft macht die LZW-Kompression unter den betrachteten Substitutionscodierungen einzigartig.

## 4.8 Burrows-Wheeler-Transformation

„The idea is to apply a reversible transformation to a block of text to form a new block that contains the same characters, but is easier to compress by simple compression algorithms. The transformation tends to group characters together so that the probability of finding a character close to another instance of the same character is increased substantially.“

M. Burrows, D. J. Wheeler [14]

In diesem Abschnitt beschäftigen wir uns mit einem Verfahren, das keine Kompression im eigentlichen Sinne ist. Durch die *Burrows-Wheeler-Transformation* werden die Symbole einer Nachricht nicht verkürzt, sondern lediglich so umgeordnet, dass sie von gängigen Kompressionsverfahren besser verarbeitet werden können (Abbildung 4.51). Die Eigenschaft, die Symbole einer Nachricht lediglich zu permutieren, ist der Grund, warum das Verfahren als Transformation und nicht als Kompression bezeichnet wird.

**Abb. 4.50:** LZW-Dekompression unserer Beispieldaten

Zurück geht das Verfahren auf eine Idee von David Wheeler aus dem Jahr 1978 [13] (Abbildung 4.52). Der britische Computerwissenschaftler sah damals davon ab, sie zu publizieren; er war dafür bekannt, viele seiner Neuentdeckungen für offensichtlich zu halten, und das Gleiche dachte er offenbar auch von seiner Transformation. Drei Jahre nach Wheelers Tod äußerte sich Michael Burrows mit den folgenden Worten über diesen Wesenszug seines ehemaligen Lehrers:

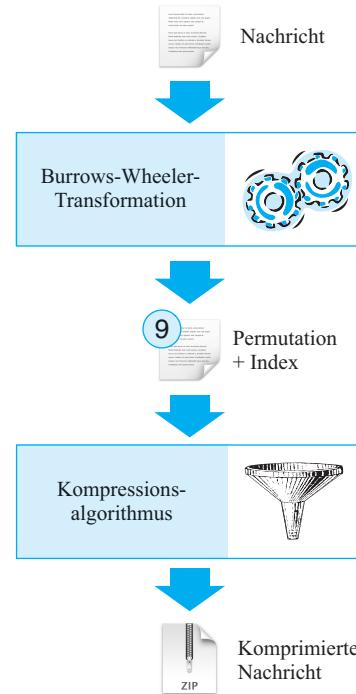
*„David would sometimes invent a cunning algorithm but would file it away rather than publish it because he felt it was obvious. Many Cambridge graduate students found these ideas less obvious, and had the experience of visiting his office with a problem, only to be given a sheet of paper on which David had written the solution some years previously. This was my experience in 1984, when I became another of David’s students. I hate to think of how many of his techniques lay undiscovered in those filing cabinets simply because no student had asked the right question.“*

Michael Burrows [13]

So kam es, dass der Öffentlichkeit die von Wheeler entdeckte Transformation über 10 Jahre verborgen blieb. Erst Michael Burrows gelang es, Wheeler zu einer Veröffentlichung zu überreden. In [13] erinnert er sich mit den folgenden Worten:

*„Years passed, and it became clear that David had no thought of publishing the algorithm – he was too busy thinking of new things. Eventually, I decided to force his hand: I could not make him write a paper, but I could write a paper with him, given the right excuse. So I enlisted his help in finding ways to execute the algorithm’s sorting step efficiently, which involved considering constant factors as much as asymptotic behavior. We tried many things, only some of which made it into the paper, but we met my goals: we showed that the algorithm could be made fast enough to see practical use on modern machines, and I did enough to assuage my guilt in putting my name next to David’s when describing his algorithm.“*

Michael Burrows [13]



**Abb. 4.51:** Die Burrows-Wheeler-Transformation ist kein Kompressionsverfahren im klassischen Sinne. Sie wird als Vorverarbeitungsstufe für andere Kompressionsverfahren eingesetzt.

Tatsächlich gestaltete es sich schwieriger als gedacht, die Arbeit zu veröffentlichen. Das Programmkomitee der *Data Compression Conference*

Die Mitglieder des *Computer Laboratory* der Universität Cambridge im Mai 1949 [20]



David Wheeler  
(1927 – 2004)

**Abb. 4.52:** Der britische Computerwissenschaftler David Wheeler gehört zu den Pionieren des Computerzeitalters. Als ein Mann der ersten Stunde war er an der Konzeption der elementaren Datenverarbeitungsmechanismen beteiligt, auf denen auch heute noch sämtliche Computersysteme beruhen. Das berühmteste Beispiel ist die JSR-Instruktion (*jump to subroutine*), die früher gerne als *Wheeler jump* bezeichnet wurde. Später hat sich Wheeler der Codierungstheorie und der Kryptografie zugewandt und auf diesen Gebieten wichtige Arbeiten publiziert. Viele Programmierer kennen David Wheeler aus einem anderen Grund. Er war der Doktorvater von Bjarne Stroustrup, dem Schöpfer der Programmiersprache C++.

lehnte die Arbeit ab, und so entschlossen sich Burrows und Wheeler dazu, ihre Ergebnisse in einem Forschungsbericht niederzuschreiben. Der Bericht erschien im Jahr 1994 unter dem Titel *A Block-sorting Lossless Data Compression Algorithm* und trägt das Kürzel SRC-124. SRC ist die Abkürzung für das *Systems Research Center*, das die Digital Equipment Corporation (DEC) im Jahr 1984 im Kalifornischen Palo Alto gegründet hatte und 1994 die Arbeitsstätte von Burrows und Wheeler war.

Bekannt wurde die Arbeit von Burrows und Wheeler durch einen Artikel von Mark Nelson aus dem Jahr 1996 [67]. In diesem Artikel wurde erstmals der Name *Burrows-Wheeler-Transformation* verwendet, den das Verfahren bis heute trägt.

*„The Data Compression Conference did not like the paper, perhaps put off by my rather pragmatic presentation. However, Mark Nelson drew attention to it in a Dr. Dobbs article, and that was enough to ensure its survival. I was embarrassed to see that Mark Nelson called it the ‚Burrows–Wheeler Transform‘, even though I tried to make it clear that the algorithm was David’s. Of course, David was far too polite to mention this.“*

Michael Burrows [13]

## Codierung

So trickreich die Burrows-Wheeler-Transformation auch ist: Sie lässt sich in wenigen Worten beschreiben:

*„Briefly, our algorithm transforms a string  $S$  of  $N$  characters by forming the  $N$  rotations (cyclic shifts) of  $S$ , sorting them lexicographically, and extracting the last character of each of the rotations. A string  $L$  is formed from these characters, where the  $i$ th character of  $L$  is the last character of the  $i$ th sorted rotation. In addition to  $L$ , the algorithm computes the index  $I$  of the original string  $S$  in the sorted list of rotations.“*

M. Burrows and D. J. Wheeler [14]

Wir wollen das beschriebene Verfahren am Beispiel der Nachricht

$$S := \text{EIERBREI}$$

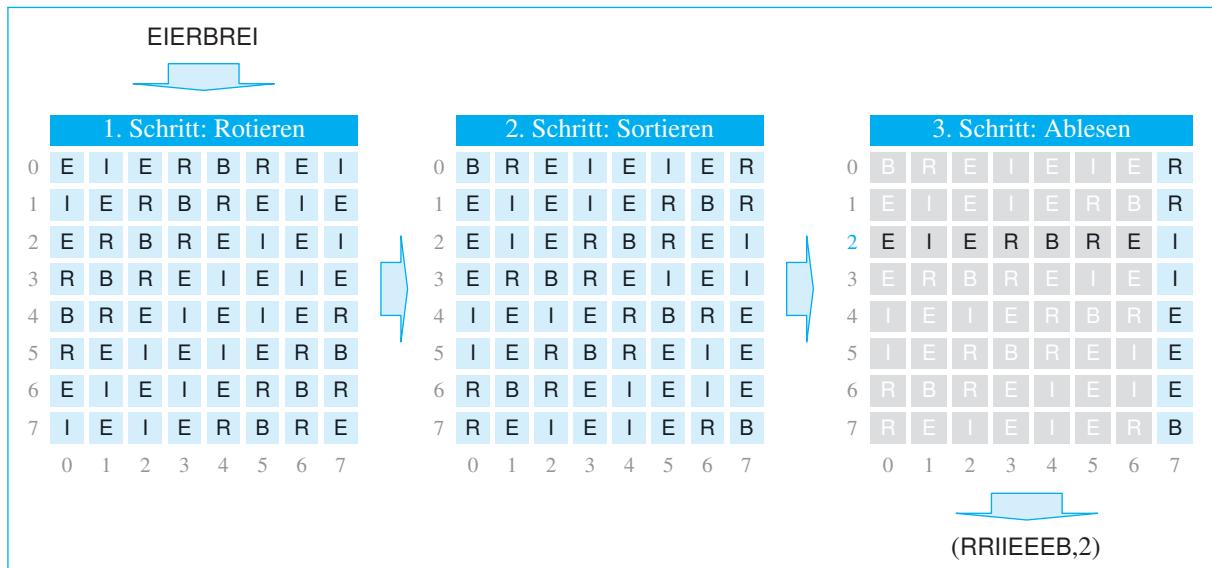


Abb. 4.53: Burrows-Wheeler-Transformation der Nachricht EIERBREI

erproben und führen hierzu die drei in Abbildung 4.53 gezeigten Schritte aus:

- Im ersten Schritt wird die *Rotationsmatrix* erzeugt. Diese entsteht, indem die Nachricht mehrfach um ein Zeichen nach links rotiert wird und die entstehenden Zeichenketten untereinander aufgeschrieben werden. Ist  $N$  die Anzahl der Symbole, so besteht die Rotationsmatrix aus  $N$  Zeilen und  $N$  Spalten. In unserem Beispiel ist  $N = 8$  und das Ergebnis eine Matrix mit 64 Feldern.
- Im zweiten Schritt werden die Zeilen der Rotationsmatrix lexikografisch sortiert. Beachten Sie, dass in der neuen Matrix lediglich die Reihenfolge der Zeilen eine andere ist und die Spaltenpositionen der einzelnen Symbole nicht angetastet werden.
- Im dritten Schritt werden die Symbole der letzten Spalte von oben nach unten ausgelesen und zusammen mit der Nummer der Zeile ausgegeben, in der sich die Originalnachricht befindet. In der verbalen Beschreibung von Burrows und Wheeler ist der Spalteninhalt der String  $L$ , die Zeilennummer der Index  $I$  und das Paar  $(L, I)$  das Ergebnis der Burrows-Wheeler-Transformation.

EIERBREI	
	
Symbolpaare	
EIERBREI	👉 EI
EIERBREI	👉 IE
EIERBREI	👉 ER
EIERBREI	👉 RB
EIERBREI	👉 BR
EIERBREI	👉 RE
EIERBREI	👉 EI
EIERBREI	👉 IE
	
2. Schritt: Sortieren	
0	B R
1	E I
2	E I
3	E R
4	I E
5	I E
6	R B
7	R E
0	1 2 3 4 5 6 7

**Abb. 4.54:** Werden die Symbolpaare lexikografisch angeordnet, so entstehen die ersten beiden Spalten der sortierten Rotationsmatrix.

Für die Beispielnachricht EIERBREI erhalten wir das Ergebnis

$$(L, I) = (\text{RRIIIEEEB}, 2)$$

Die produzierte Ausgabe macht deutlich, welches Ziel die Burrows-Wheeler-Transformation verfolgt: Sie versucht, die Symbole der Originalnachricht in eine Abfolge zu bringen, die regelmäßiger ist als die ursprüngliche. Für unser Beispiel gelingt dies besonders gut, und wir wollen uns überlegen, warum dies so ist. In der Nachricht EIERBREI kommt der Buchstabe E dreimal vor und wird dabei zweimal, wie es für deutsche Texte typisch ist, mit dem Buchstaben I fortgesetzt. Die lexikografische Sortierung der Rotationstabelle bewirkt, dass die beiden I in der ersten Spalte direkt untereinander stehen, und dies wiederum führt dazu, dass die zugehörigen E in der letzten Spalte ebenfalls untereinander stehen. Dass in der letzten Spalte auch das dritte E in unmittelbarer Nähe der beiden anderen E steht, ist dagegen Zufall. Sein Nachfolgesymbol ist ein R und dieses Symbol taucht nur deshalb in der ersten Spalte unterhalb der beiden I auf, da unsere sehr kurz gewählte Beispielnachricht keine Symbole enthält, die lexikografisch zwischen I und R eingeordnet sind.

Unsere Überlegung zeigt, dass die Burrows-Wheeler-Transformation auf vorhandene Kontextabhängigkeiten angewiesen ist. Nur dann, wenn bestimmte Symbolkombinationen, wie die Kombination EI in unserem Beispiel, gehäuft vorkommen, entsteht in der letzten Spalte der sortierten Rotationsmatrix eine gewisse Ordnung. Dies bedeutet im Umkehrschluss, dass die Burrows-Wheeler-Transformation nutzlos ist, wenn die zu transformierende Nachricht aus einer gedächtnislosen Quelle stammt. Sind keine Kontextabhängigkeiten zwischen den emittierten Symbolen vorhanden, so muss die transformierte Nachricht genauso ungeordnet sein wie die Originalnachricht.

## Decodierung

Damit sind wir an dem interessantesten Aspekt der Burrows-Wheeler-Transformation angekommen: der Decodierung. Damit die Transformation überhaupt sinnvoll eingesetzt werden kann, muss es möglich sein, die Originalnachricht aus dem Paar  $(L, I)$  wiederherzustellen. Auf den ersten Blick scheint dies ein aussichtsloses Unterfangen zu sein, und selbst auf den zweiten Blick ist es schwierig zu sehen, dass das Paar  $(L, I)$  noch genug Information enthält, um die sortierte Rotationsmatrix vollständig zu rekonstruieren. Der Schlüssel für die Rekonstruktion ist die lexikografische Sortierung; sie ist die notwendige Zusatzinformation, die uns erlaubt, aus dem vergleichsweise regelmäßigen String  $L$  und

dem Index  $I$  die Originalnachricht  $S$  zurückzugewinnen. Wie dies im Detail geschehen kann, wollen wir uns jetzt genauer ansehen.

Die einfachste Möglichkeit, die sortierte Rotationsmatrix zu rekonstruieren, besteht darin, die Spalten schrittweise von links nach rechts wiederherzustellen. Die Rekonstruktion der ersten Spalte basiert auf der Eigenschaft, dass die Symbole der Originalnachricht in jeder Spalte der Rotationsmatrix mit der gleichen Häufigkeitsverteilung vorkommen. Mit anderen Worten: Jede Spalte der Rotationsmatrix ist eine Permutation der Originalnachricht. Ferner wissen wir, dass die Symbole in der ersten Spalte lexikografisch sortiert sind. Das bedeutet, dass wir die erste Spalte direkt ausfüllen können, indem wir die Symbole der Originalnachricht sortiert auflisten.

Um die zweite Spalte auszufüllen, benötigen wir eine Eigenschaft der Rotationsmatrix, die in Abbildung 4.54 (oben) veranschaulicht wird. Dort sind sämtliche Symbolpaare aufgelistet, die in der Nachricht EI-ERBREI vorkommen. Die Nachricht wird dabei zyklisch betrachtet, sodass das letzte und das erste Symbol ebenfalls ein Symbolpaar bilden. Ein vergleichender Blick auf die sortierte Rotationsmatrix zeigt, dass wir sämtliche Symbolpaare auch in der ersten und der zweiten Spalte wiederfinden – in sortierter Reihenfolge.

Damit ist der Weg für die Rekonstruktion geebnet. Da uns die erste Spalte der Rotationsmatrix bereits bekannt ist, können wir auch die Symbolpaare bestimmen. Dies gelingt, weil wir auch die letzte Spalte der Rotationsmatrix kennen; diese ist mit der transformierten Nachricht identisch. Um sie zu erzeugen, brauchen wir also lediglich die transformierte Nachricht vor die erste Spalte zu schreiben und die Zeilen lexikografisch zu sortieren. Als Ergebnis entsteht die erste und die zweite Spalte der sortierten Rotationsmatrix (Abbildung 4.54 unten).

Die Rekonstruktion der anderen Spalten verläuft analog. Betrachten wir die ersten drei Spalten der Rotationsmatrix, so finden wir dort alle Symboltripel der Nachricht EI-ERBREI wieder (Abbildung 4.55). Das bedeutet, dass wir die dritte Spalte rekonstruieren können, indem wir erneut die Nachricht RRIIEEEB vor die Rotationsmatrix schreiben und die Zeilen lexikografisch sortieren. Fahren wir, wie in Abbildung 4.56 gezeigt, auf die gleiche Weise fort, so gewinnen wir in jedem Schritt eine weitere Spalte hinzu, bis die sortierte Rotationsmatrix vollständig rekonstruiert ist.

The diagram illustrates the construction of the first three columns of the sorted rotation matrix. It starts with the transformed message "EI-ERBREI" at the top. Below it, a blue box labeled "Symboltripel" contains all possible symbol pairs from the message: EIE, IER, ERB, RBR, BRE, REI, EIE, and IEI. A blue bracket groups the first four pairs (EIE, IER, ERB, RBR) under the heading "EIERBREI". This group is then mapped to the first column of a 8x8 grid, where each row is indexed from 0 to 7. The first column is filled with the symbols B, R, E, I, E, I, R, and B respectively. The second column is partially filled with E, I, E, R, E, R, B, and I. The third column is partially filled with R, B, R, E, and I. The bottom of the grid shows indices 0 through 7.

EIERBREI							
Symboltripel							
EIERBREI	☞ EIE						
EIERBREI	☞ IER						
EIERBREI	☞ ERB						
EIERBREI	☞ RBR						
EIERBREI	☞ BRE						
EIERBREI	☞ REI						
EIERBREI	☞ EIE						
EIERBREI	☞ IEI						

2. Schritt: Sortieren							
0	B	R	E				
1	E	I	E				
2	E	I	E				
3	E	R	B				
4	I	E	I				
5	I	E	R				
6	R	B	R				
7	R	E	I				
	0	1	2	3	4	5	6

Abb. 4.55: Werden die Symboltripel der Originalnachricht lexikografisch angeordnet, so entstehen die ersten drei Spalten der sortierten Rotationsmatrix.

1. Voranstellen								2. Sortieren								3. Voranstellen								4. Sortieren							
R	B							B	R						R	B	R					B	R	E							
R	E							E	I						R	E	I					E	I	E							
I	E							E	I						I	E	I					E	I	E							
I	E							E	R						I	E	R					E	R	B							
E	I							I	E						E	I	E					I	E	I							
E	I							I	E						E	I	E					I	E	R							
E	R							R	B						E	R	B					R	B	R							
B	R							R	E						B	R	E					R	E	I							
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	
5. Voranstellen								6. Sortieren								7. Voranstellen								8. Sortieren							
R	B	R	E					B	R	E	I				R	B	R	E	I			B	R	E	I						
R	E	I	E					E	I	E	I				R	E	I	E	I			E	I	E	I						
I	E	I	E					E	I	E	R				I	E	I	E	R			E	I	E	R	B					
I	E	R	B					E	R	B	R				I	E	R	B	R			E	R	B	R						
E	I	E	I					I	E	I	E				E	I	E	I	E			I	E	I	E						
E	I	E	R					I	E	R	B				E	I	E	R	B			I	E	R	B						
E	R	B	R					R	B	R	E				E	R	B	R	E			R	B	R	E	I					
B	R	E	I					R	E	I	E				B	R	E	I	E			R	E	I	E	I					
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	
...																															
11. Voranstellen								12. Sortieren								13. Voranstellen								14. Sortieren							
R	B	R	E	I	E	I		B	R	E	I	E	I		R	B	R	E	I	E	I		B	R	E	I	E	I	E	R	
R	E	I	E	I	E	R		E	I	E	I	E	R	B	R	E	I	E	I	E	R	B	E	I	E	I	R	B	R		
I	E	I	E	R	B	R		E	I	E	R	B	R	E	I	E	I	E	R	B	R	E	E	I	E	R	B	R	E	I	
I	E	R	B	R	E	I		E	R	B	R	E	I	E	I	E	R	B	R	E	I	E	E	R	B	R	E	I	E	I	
E	I	E	I	E	R	B		I	E	I	E	R	B	R	E	I	E	I	E	R	B	R	I	E	I	E	R	B	R	E	
E	I	E	R	B	R	E		I	E	R	B	R	E	I	E	I	E	R	B	R	E	I	I	E	R	B	R	E	I	E	
E	R	B	R	E	I	E		R	B	R	E	I	E	I	E	R	B	R	E	I	E	I	R	B	R	E	I	E	I	E	
B	R	E	I	E	I	E		R	E	I	E	I	E	R	B	E	R	I	E	I	E	R	B	E	I	E	I	R	B	E	I
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	

Abb. 4.56: Inverse Burrows-Wheeler-Transformation der Nachricht (RRIIEEEB,2)

#### 4.8.1 Move-to-front-Codierung

In Abbildung 4.51 haben wir skizziert, dass die Burrows-Wheeler-Transformation als Vorverarbeitungsstufe eingesetzt wird, aber noch kein Wort über den nachgeschalteten Kompressor verloren. In ihrer Originalarbeit haben Burrows und Wheeler die in Abbildung 4.57 dargestellte Verarbeitungskette vorgeschlagen. Die transformierten Daten werden zunächst mit einem *Move-to-front-Codierer* weiter verarbeitet und anschließend Huffman-codiert.

Hinter der Move-to-front-Codierung verbirgt sich ein einfaches Verfahren, das eine Folge von Symbolen in eine gleich lange Folge von Indizes übersetzt. Die Indizes werden dabei aus einer Liste ausgelesen, die sich dynamisch ändert. Abbildung 4.58 (links) demonstriert das Vorgehen am Beispiel der Nachricht RRIIEEEB. Initial enthält die Liste die Symbole des Quellenalphabets in einer vorab festgelegten Reihenfolge; in unserem Beispiel sind die Symbole lexikografisch sortiert. Jetzt wird die Nachricht Symbol für Symbol verarbeitet. In jedem Schritt wird der Index des aktuell betrachteten Symbols ausgegeben und dieses anschließend an den Anfang der Liste gestellt (*move to front*). Die relative Position der anderen Symbole bleibt dabei unverändert. Dieses Verfahren wird so lange wiederholt, bis die Nachricht vollständig codiert ist.

Für unser Beispiel erhalten wir die Folge

30303003.

die anschließend mit einem Huffman-Encoder in eine Binärsequenz übersetzt wird. Wie in Abbildung 4.58 (Mitte) zu sehen ist, liefert der Huffman-Encoder die Ausgabe

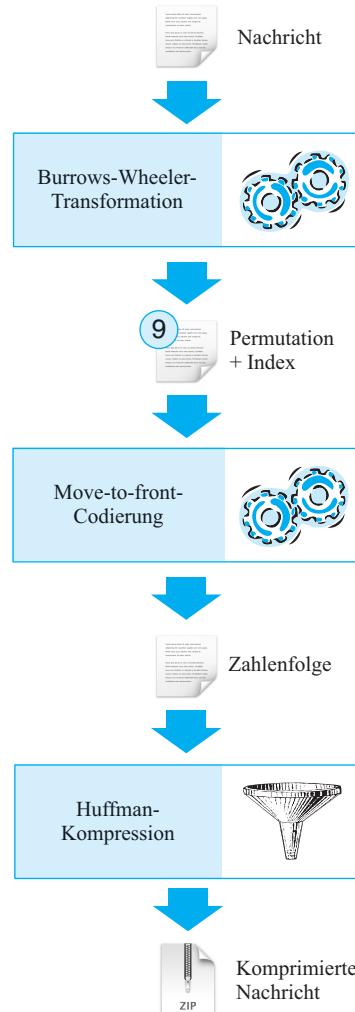
10101001

(4.31)

Abbildung 4.58 (rechts) zeigt, welches Ergebnis der Huffman-Encoder erzeugt, wenn auf die Move-to-front-Codierung verzichtet wird. Mit

1010010100000011 bzw. 1101101010000111

liefert er eine doppelt so lange Binärsequenz. Für unser Beispiel hat die Move-to-front-Codierung die Wahrscheinlichkeitsverteilung der Symbole so verändert, dass der Huffman-Algorithmus einen Code mit einer geringeren mittleren Codewortlänge erzeugen kann.



**Abb. 4.57:** Die Burrows-Wheeler-Transformation, kombiniert mit einem Move-to-front- und einem Huffmann-Codierer

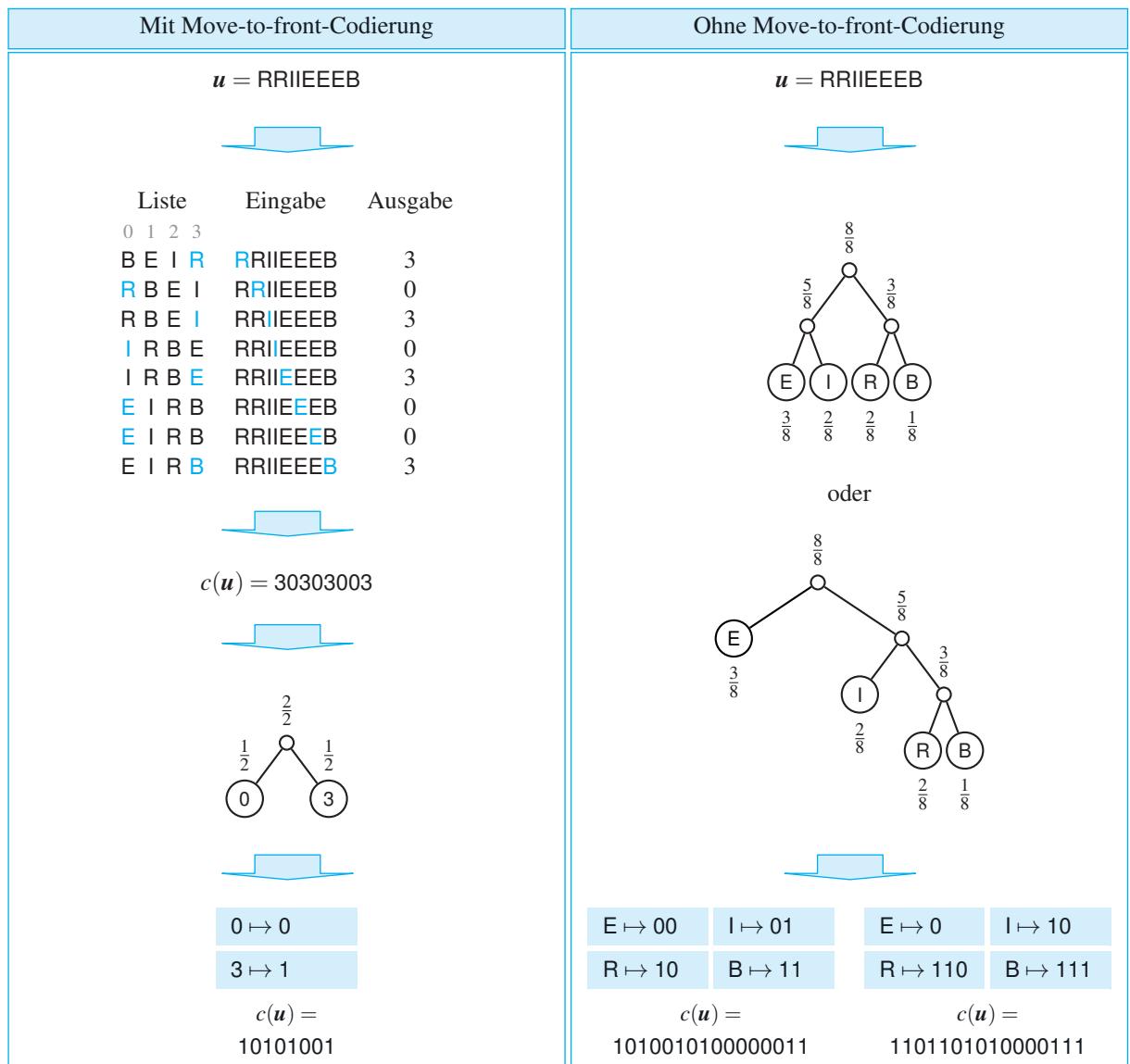


Abb. 4.58: Move-to-front-Codierung am Beispiel der Nachricht RRIIIEEEB

## 4.9 Übungsaufgaben

Auf Seite 240 haben wir begründet, warum der Grenzwert

$$L = \lim_{|\mathbf{u}| \rightarrow \infty} \frac{|c(\mathbf{u})|}{|\mathbf{u}|} \quad (4.32)$$

als Maßzahl für die Kompressionsgüte geeignet ist. Die Definition ist intuitiv einleuchtend, und wir können für nahezu alle in der Praxis vorkommenden Datenquellen und Codierungen erwarten, dass dieser mutmaßliche Grenzwert auch tatsächlich existiert.

Nichtsdestotrotz lassen sich Fälle konstruieren, in denen dies nicht zutrifft. Als Beispiel betrachten wir eine Datenquelle mit dem Quellenalphabet  $\Sigma = \{A, B\}$  und die Codierung  $c$  mit  $c(A) = 0$  und  $c(B) = 11$ .

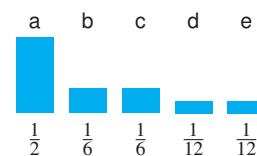
Die Datenquelle sei so konzipiert, dass sie zunächst das Symbol A emittiert. Danach folgen 2 B, dann 4 A, dann 8 B und so fort.



ABBAAAABBBBBBBBAAAAAAAAAAAAAAA...

Zeigen Sie, dass der Grenzwert (4.32) in unserem fiktiven Beispiel nicht existiert.

In dieser Aufgabe sind drei Datenquellen und drei Codierungen vorgegeben:



Codierung $c_1$	
$a \mapsto 0$	$b \mapsto 100$
$c \mapsto 101$	$d \mapsto 110$
$e \mapsto 111$	

Codierung $c_2$	
$a \mapsto 00$	$b \mapsto 010$
$c \mapsto 011$	$d \mapsto 10$
$e \mapsto 11$	

Codierung $c_3$	
$a \mapsto 0$	$b \mapsto 10$
$c \mapsto 110$	$d \mapsto 1110$
$e \mapsto 1111$	

Bestimmen Sie für jede Datenquelle, welche der drei Codierungen die höchste Kompressionsrate liefert.

### Aufgabe 4.1

Webcode  
4012

### Aufgabe 4.2

Webcode  
4059

**Aufgabe 4.3**
**Webcode  
4149**

Die folgenden Bilder zeigen verschiedene Wahrscheinlichkeitsverteilungen, die durch die Analyse zweier Texte gewonnen wurden. Als Grundlage für die Datenerhebung diente Lewis Carrolls Klassiker *Alice's Adventures in Wonderland*, einmal im englischen Original und ein zweites Mal in einer deutschen Übersetzung. In der Analyse wurde nicht zwischen Groß- und Kleinbuchstaben unterschieden, und es wurden alle Sonderzeichen außer dem Leerzeichen („ „) entfernt. Die Wahrscheinlichkeiten sind durch Quadrate proportionaler Größe visualisiert.

Alice's Adventures in Wonderland (Englisch)

Bild 1

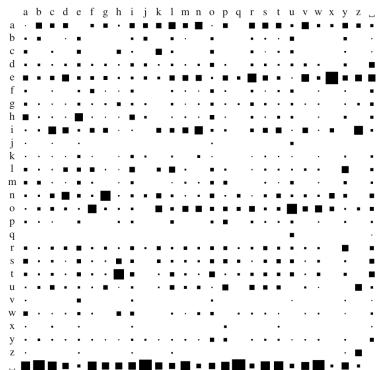


Bild 2

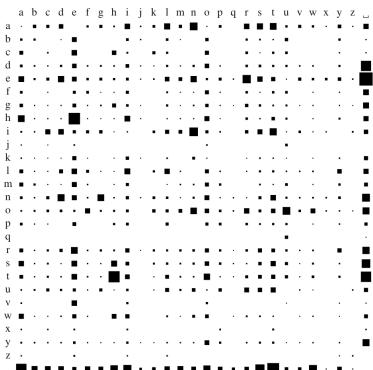
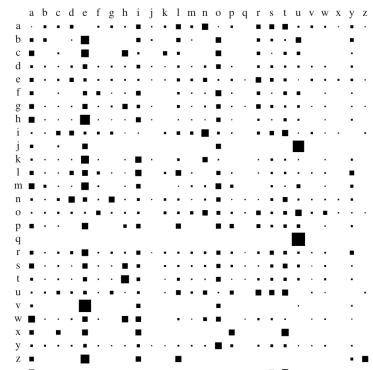


Bild 3



Alice im Wunderland (Deutsch)

Bild 1

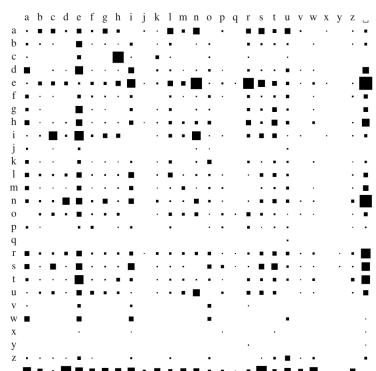


Bild 2

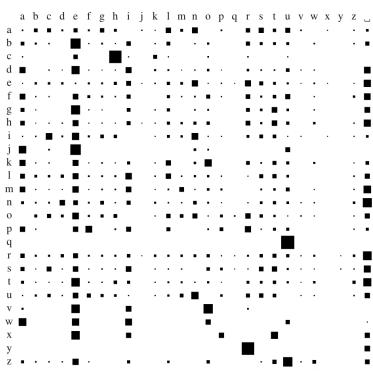
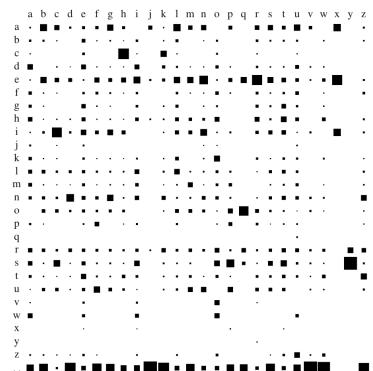


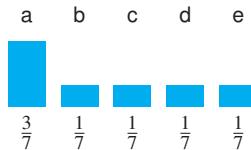
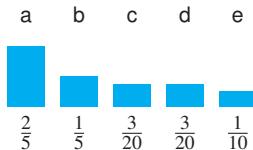
Bild 3



Finden Sie für jede Sprache heraus,

- welches Bild die Bigrammwahrscheinlichkeiten  $p(\sigma, \tau)$  visualisiert,
- welches Bild die Transitionswahrscheinlichkeiten  $p_\sigma(\tau)$  visualisiert und
- welche Bedeutung den verbleibenden Bildern zukommt.

Gegeben seien die folgenden Datenquellen über dem Alphabet  $\Sigma = \{a, b, c, d, e\}$ :



### Aufgabe 4.4



Webcode  
4163

- Wenden Sie auf jede Quelle die Verfahren von Shannon, Fano und Huffman an.
- Welche Codierung produziert das jeweils beste Ergebnis?

Mit dem folgenden Beispiel hat David A. Huffman in seiner Arbeit aus dem Jahr 1952 das nach ihm benannte Konstruktionsverfahren erklärt [44]:

### Aufgabe 4.5



Webcode  
4240

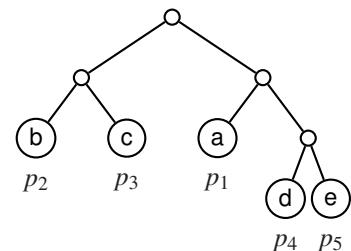
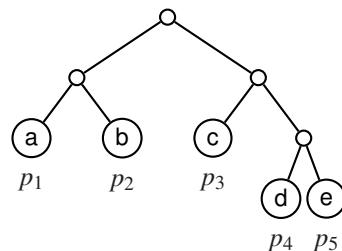
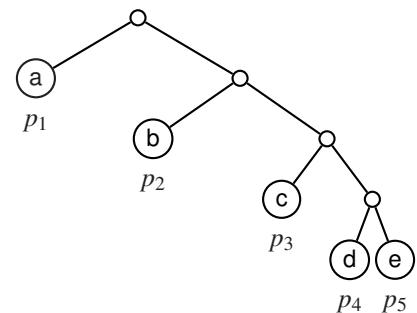
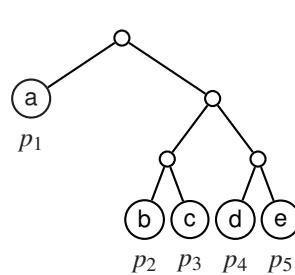
TABLE I  
OPTIMUM BINARY CODING PROCEDURE

Original Message Ensemble	Message Probabilities											
	1	2	3	4	5	6	7	8	9	10	11	12
0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.24	0.36	0.40	0.60	1.00
0.18	0.18	0.18	0.18	0.18	0.18	0.18	0.18	0.20	0.24	0.36	0.40	
0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.14	0.14	0.20	0.36	
0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.20	0.36	
0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.20	0.36	
0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.10	0.14	0.20	0.36	
0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.10	0.20	
0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.10	0.20	
0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.10	
*0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.10	
0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.10	
0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.10	
0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	
0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	

- Übersetzen Sie Huffmans tabellarische Darstellung in einen Codebaum.
- Wie groß ist die mittlere Codewortlänge dieses Codes?
- Produziert der Fano-Algorithmus für Huffmans Beispiel ein schlechteres Ergebnis?

**Aufgabe 4.6****Webcode  
4264**

Finden Sie Werte für  $p_1, \dots, p_5$ , sodass alle vier der nachstehend abgebildeten Codebäume gleichzeitig Huffman-Bäume sind:

**Aufgabe 4.7****Webcode  
4343**

In der untenstehenden Tabelle wird die Shannon-Codierung ausgeführt, die Sie in Abschnitt 4.4.1 kennengelernt haben. Versuchen Sie, die Tabelle, in der einige Einträge fehlen, vollständig zu rekonstruieren:

	$i$	$p_i$	$\log_2 \frac{1}{p_i}$	$\lceil \log_2 \frac{1}{p_i} \rceil$	$P_i := \sum_{j=1}^{i-1} p_j$	Codewort
a	1				= 0,000000...	00
b	2				= 0,010000...	010
c	3				= 0,011100...	011
d	4				= 0,100110...	100
e	5				= 0,110000...	1100
f	6				= 0,110110...	1101
g	7				= 0,111100...	11110

Beweisen oder widerlegen Sie die folgenden Behauptungen:

- a) Emittiert eine gedächtnislose Quelle ihre Symbole mit den Wahrscheinlichkeiten

$$\frac{1}{2^{n_1}}, \frac{1}{2^{n_2}}, \dots, \quad (n_i \in \mathbb{N})$$

so produziert der Huffman-Algorithmus eine redundanzfreie Codierung.

- b) Emittiert eine gedächtnislose Quelle ihre Symbole mit einer anderen Wahrscheinlichkeitsverteilung, so produziert der Huffman-Algorithmus eine Codierung mit einem positiven Redundanzanteil.

**Aufgabe 4.8**

**Webcode  
4346**

In Kapitel 3 haben Sie auf Seite 187 den BCD-Code kennengelernt, der für die Darstellung von Dezimalzahlen geschaffen wurde. Bei der BCD-Codierung wird jede Dezimalziffer anhand der folgenden Tabelle in ein 4-Bit-Paket übersetzt:

**Aufgabe 4.9**

**Webcode  
4419**

0 $\mapsto$ 0000	1 $\mapsto$ 0001	2 $\mapsto$ 0010	3 $\mapsto$ 0011	4 $\mapsto$ 0100
5 $\mapsto$ 0101	6 $\mapsto$ 0110	7 $\mapsto$ 0111	8 $\mapsto$ 1000	9 $\mapsto$ 1001

In den folgenden Teilaufgaben betrachten wir eine gedächtnislose Datenquelle, die Dezimalziffern emittiert, und eine Codierung  $c$ , die den Ziffernstrom in BCD-Pakete übersetzt. Wir nehmen an, dass alle Ziffern mit der gleichen Wahrscheinlichkeit emittiert werden.

- a) Berechnen Sie die Entropie der Datenquelle.  
 b) Berechnen Sie die Redundanz der BCD-Codierung.

Wir wollen versuchen, die Redundanz der Codierung zu verringern, indem die Ziffern nicht mehr einzeln, sondern in Blöcken codiert werden. Hierzu werden jeweils  $n$  Ziffern zusammengefasst und gemeinsam auf ein Bitpaket der Länge  $m$  abgebildet.

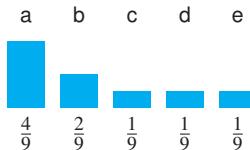
- c) Wie groß muss  $m$  für die Fälle  $n = 2$  und  $n = 3$  gewählt werden?  
 d) Wie hat sich die Entropie und die Redundanz des modifizierten Codes jeweils verändert? Welchem Wert strebt die Redundanz für den Fall  $n \rightarrow \infty$  zu?

*Ternäre Huffman-Codes* unterscheiden sich von den gewöhnlichen Huffman-Codes dadurch, dass sie anstelle des zweielementigen Codealphabets  $\{0, 1\}$  das dreielementige Codealphabet  $\{0, 1, 2\}$  verwenden. Auf den ersten Blick sieht es so aus, als könnten wir den Huffman-Algorithmus in fast unveränderter Form für die Konstruktion eines optimalen ternären Codes einsetzen, indem wir in den einzelnen Konstruktionsschritten nicht zwei, sondern jeweils drei Knoten zusammenfassen.

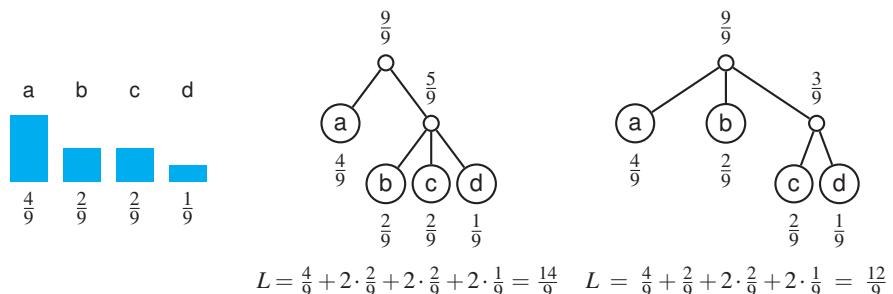
**Aufgabe 4.10**

**Webcode  
4420**

a) Erzeugen Sie auf die geschilderte Weise eine ternäre Codierung für die folgende Quelle:



b) Das nächste Beispiel zeigt, dass wir auf diese Weise nicht immer einen optimalen ternären Code erhalten. Führen wir das Verfahren wie geschildert aus, so entsteht der linke der beiden untenstehenden Codebäume. Der Baum rechts daneben zeigt, dass ein ternärer Code mit einer geringeren mittleren Codewortlänge existiert.



Modifizieren Sie den vorgeschlagenen Algorithmus so, dass immer ein optimaler ternärer Code entsteht.

### Aufgabe 4.11



Webcode  
4500

Nachstehend sehen Sie den Anfang der *Fibonacci-Folge*, die der italienische Mathematiker Leonardo da Pisa, genannt Fibonacci, in der ersten Hälfte des dreizehnten Jahrhunderts entdeckte:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55,  ,  ,  , ...



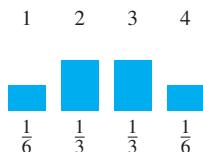
- Führen Sie die Folge durch das Ausfüllen der leergelassenen Felder fort.
- Welche rekursive Berechnungsvorschrift liegt der Fibonacci-Folge zugrunde?
- Nehmen Sie an, dass die Wahrscheinlichkeiten, mit der eine Datenquelle ihre  $n$  Symbole emittiert, im gleichen Größenverhältnis zueinander stehen wie die ersten  $n$  Fibonacci-Zahlen. Wie sieht ein optimaler präfixfreier Code für diese Datenquelle aus?

Leonardo da Pisa  
(ca. 1170 – ca. 1250)

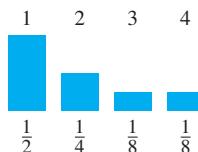
In Kapitel 3 haben Sie gelernt, wie Faxgeräte die Lauflängen von weißen und schwarzen Pixelketten mithilfe eines längenvariablen Codes in Bitsequenzen übersetzen. In dieser Aufgabe gehen wir von zwei fiktiven Wahrscheinlichkeitsverteilungen der Lauflängen aus, die durch die folgenden beiden Histogramme gegeben sind:

**Aufgabe 4.12****Webcode****4560**

Weiße Pixelketten



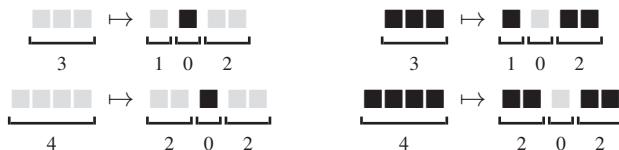
Schwarze Pixelketten



Um die Betrachtung zu vereinfachen, nehmen wir an, dass keine anderen als die genannten Lauflängen 1, 2, 3 und 4 vorkommen.

- a) Erzeugen Sie für jedes Histogramm einen optimalen präfixfreien Code.

Sie wissen bereits, dass der Faxcode nicht für alle Lauflängen Codewörter bereithält. Die Konsequenzen, die sich daraus ergeben, wollen wir an unserem Beispiel nachbilden. Wir nehmen hierzu an, dass nur für die Lauflängen 0, 1 und 2 Codewörter zur Verfügung stehen. Mithilfe der Lauflänge 0 ist es aber weiterhin möglich, auch längere Pixelketten zu codieren. Dies geschieht über die folgenden Zerlegungen:



- b) Berechnen Sie für jedes Histogramm, um wie viel Prozent sich eine Folge von Lauflängen verlängert, wenn die Längen 3 und 4, wie oben gezeigt, zerlegt werden.
- c) Berechnen Sie für jedes Histogramm, wie häufig die Lauflängen 0, 1 und 2 in den übersetzten Folgen vorkommen.
- d) Berechnen Sie für die beiden neuen Wahrscheinlichkeitsverteilungen einen optimalen präfixfreien Code.
- e) Ist eine derartige Zerlegung der Lauflängen sinnvoll, wenn eine möglichst hohe Kompressionsrate angestrebt wird? Welche Vorteile bietet sie in der Praxis?

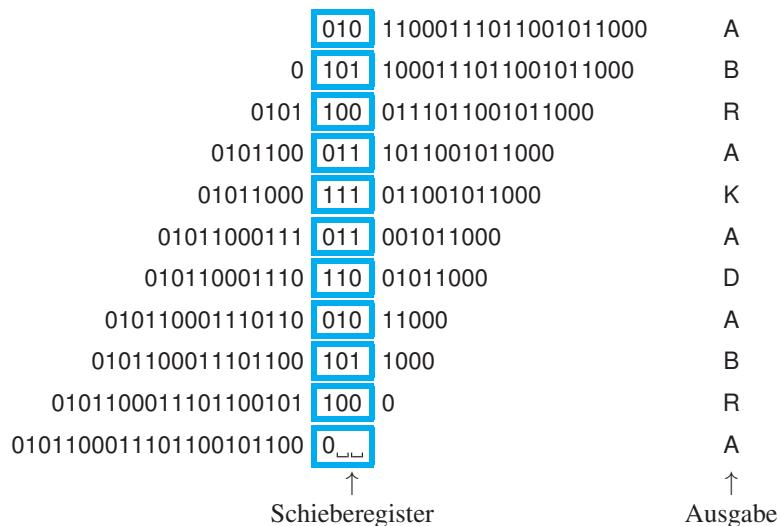
**Aufgabe 4.13**

Nachstehend sehen Sie, wie die Nachricht

 **Webcode**  
4635

01011000111011001011000

mithilfe eines Schieberegisters decodiert wird:



- a) Wie sieht die Decodiertabelle aus, die für die Rekonstruktion der Originalnachricht verwendet wurde? Übersetzen Sie die Tabelle anschließend in die optimierte Form.

Index	Symbol	Offset
000		
001		
010		
011		
100		
101		
110		
111		

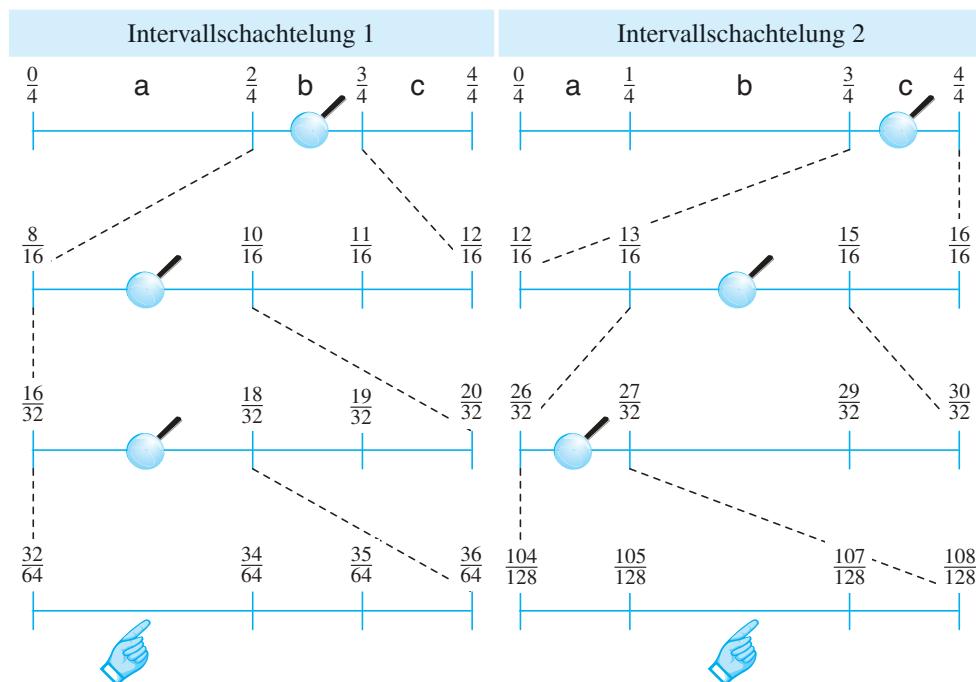
Decodiertabelle (nicht optimiert)

Index	Symbol	Offset
000		
001		
010		
011		
100		
101		
110		
111		

Decodiertabelle (optimiert)

- b) Wie viele Schritte werden eingespart, wenn für die Decodierung die optimierte Tabelle verwendet wird?

Die folgenden beiden Intervallschachtelungen sind im Rahmen der arithmetischen Codierung entstanden:



- Bestimmen Sie für beide Beispiele, mit welchen Wahrscheinlichkeiten die Datenquelle ihre Symbole emittiert.
- Welche Nachrichten repräsentieren die markierten Intervalle?
- Bestimmen Sie für beide Intervalle diejenige Zahl mit dem kürzesten Nachkommaanteil.
- Wie lauten die Codewörter, die für die beiden Beispielnachrichten erzeugt werden?

Die arithmetische Codierung folgt der Idee, der zu codierenden Nachricht ein reelles Intervall der Form  $[a; b)$  zuzuordnen und jede Zahl  $c \in [a; b)$  als einen Repräsentanten dieser Nachricht anzusehen.

- Die wenigsten reellen Zahlen weisen eine endliche Nachkommadarstellung auf. Ist es möglich, dass das Intervall  $[a; b)$  so zusammenschrumpft, dass darin keine Zahlen mit einer endlichen Nachkommadarstellung mehr enthalten sind?
- Für die Codierung wird die Zahl  $c$  mit der kürzesten Binärdarstellung verwendet. Ist die Zahl  $c$  in jedem Fall eindeutig bestimmt?

### Aufgabe 4.14



**Webcode  
4660**

### Aufgabe 4.15

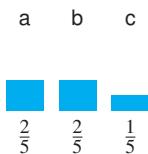
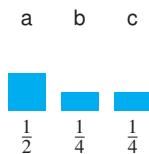
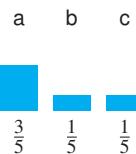


**Webcode  
4700**

**Aufgabe 4.16**

Gegeben seien die folgenden drei Datenquellen:

   
**Webcode**  
**4704**



- a) Ordnen Sie die nachstehenden rANS-Codierungstabellen jeweils der korrekten Datenquelle zu:

Tabelle 1

$x_i$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...
$u_i = a$	0	1			2	3			4	5			6	7			8	9			...
$u_i = b$			0			1				2				3				4			...
$u_i = c$				0			1				2				3				4		...

Tabelle 2

$x_i$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...
$u_i = a$	0	1	2			3	4	5			6	7	8			9	10	11			...
$u_i = b$				0				1				2				3			3		...
$u_i = c$					0					1					2				3		...

Tabelle 3

$x_i$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...
$u_i = a$	0	1				2	3				4	5				6	7				...
$u_i = b$			0	1				2	3				4	5			6	7		3	...
$u_i = c$					0					1					2				3		...

- b) Codieren Sie die folgenden Nachrichten, unter Verwendung der jeweils korrekten Codierungstabelle aus der vorherigen Teilaufgabe.

$$\mathbf{u}_1 = abac$$

$$\mathbf{u}_2 = aabc$$

$$\mathbf{u}_3 = aacb$$

$$\mathbf{u}_4 = aaabc$$

Codieren Sie die Nachrichten  $u_1 = \text{bacab}$  und  $u_2 = \text{cabaa}$ , indem Sie in den nachstehenden Rechnung die leergelassenen Felder ergänzen:

**Aufgabe 4.17**
**Webcode  
4714**

$$\begin{aligned}x_1 &= C(\quad 0\quad, b) = \quad \quad \quad \\x_2 &= C(\quad \quad , a) = \quad \quad \quad \\x_3 &= C(\quad \quad , c) = \quad \quad \quad \\x_4 &= C(\quad \quad , a) = \quad \quad \quad \\x_5 &= C(\quad \quad , b) = \quad \quad 178\end{aligned}$$

$$\begin{aligned}x_1 &= C(\quad 0\quad, c) = \quad \quad \quad \\x_2 &= C(\quad \quad , a) = \quad \quad \quad \\x_3 &= C(\quad \quad , b) = \quad \quad \quad \\x_4 &= C(\quad \quad , a) = \quad \quad \quad \\x_5 &= C(\quad \quad , a) = \quad \quad 91\end{aligned}$$

Um eine rANS- bzw. uABS-codierte Nachricht zu dekodieren, müssen wir die Iterationsvorschriften (4.26) und (4.27) umkehren. Zwar wurde im Fließtext erwähnt, dass eine solche Umkehrung möglich ist, die konkreten Formeln wurden aber nicht gezeigt.

**Aufgabe 4.18**
**Webcode  
4737**

- Konstruieren Sie die inverse Iterationsvorschrift zu Formel (4.26).
- Konstruieren Sie die inverse Iterationsvorschrift zu Formel (4.27).

Auf Seite 271 haben Sie mit Formel (4.27) die Iterationsvorschrift der uABS-Codierung kennen gelernt. Eine alternative Codierungsfunktion ist diese hier [26]:

$$C(u_i, x_{i-1}) := \begin{cases} \left\lfloor \frac{x_{i-1}}{1-p} \right\rfloor & \text{falls } u_i = 0 \\ \left\lceil \frac{x_{i-1}+1}{p} \right\rceil - 1 & \text{falls } u_i = 1 \end{cases} \quad (4.33)$$

Wie müssen die Codierungstabellen in Abbildung 4.38 abgeändert werden, wenn sie aus dieser Formel anstatt der ursprünglichen abgeleitet werden?

**Aufgabe 4.19**
**Webcode  
4759**

Gegeben sei eine Bernoulli-Quelle, die ihre beiden Symbole 0 und 1 mit der gleichen Wahrscheinlichkeit emittiert.

**Aufgabe 4.20**
**Webcode  
4764**

- Bestimmen Sie die rANS-Codewörter der Nachrichten  $1, 11, 111, \dots, 0, 00, 000, \dots$ .
- Welches Problem ist in der vorherigen Teilaufgabe aufgetreten? Welche Möglichkeiten fallen Ihnen ein, um es zu beheben?

**Aufgabe 4.21**

 **Webcode**  
**4769**

Wenden Sie die LZ77-Kompression auf die Nachricht FRISCHE FISCHE FISCHEN an.  
Ergänzen Sie hierzu die leer gelassenen Felder:

 Schieben

0 1 2 3 4 5 6 7



 Codieren

0 1 2 3 4 5 6 7



 Schieben

0 1 2 3 4 5 6 7



 Codieren

0 1 2 3 4 5 6 7



 Schieben

0 1 2 3 4 5 6 7



 Codieren

0 1 2 3 4 5 6 7



 Schieben

0 1 2 3 4 5 6 7



 Codieren

0 1 2 3 4 5 6 7



 Schieben

0 1 2 3 4 5 6 7



 Codieren

0 1 2 3 4 5 6 7



 Schieben

0 1 2 3 4 5 6 7



- 👉 Codieren 0 1 2 3 4 5 6 7  

- 👉 Schieben 0 1 2 3 4 5 6 7  

- 👉 Codieren 0 1 2 3 4 5 6 7  

- 👉 Schieben 0 1 2 3 4 5 6 7  

- 👉 Codieren 0 1 2 3 4 5 6 7  

- 👉 Schieben 0 1 2 3 4 5 6 7  

- 👉 Codieren 0 1 2 3 4 5 6 7  

- 👉 Schieben 0 1 2 3 4 5 6 7  

- 👉 Codieren 0 1 2 3 4 5 6 7  

- 👉 Schieben 0 1 2 3 4 5 6 7  

- 👉 Codieren 0 1 2 3 4 5 6 7  

- 👉 Schieben 0 1 2 3 4 5 6 7  


**Aufgabe 4.22**

 **Webcode**  
**4785**

In diesem Kapitel haben wir mit der Nachricht 001010210210212021021200 das LZ77-Verfahren demonstriert. Komprimieren Sie die gleiche Nachricht mit dem LZ78- und dem LZW-Verfahren:

a) LZ78

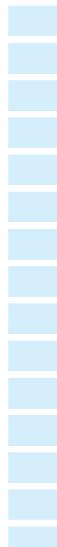
0 01010210210212021021200  
0 1 010210210212021021200  
01 0 210210212021021200  
2 10210212021021200  
1 0210212021021200  
0 2 1021021021200  
1 0 21021021021200  
2 1 2021021200  
2 0 21021200  
21 0 21200  
21 2 00  
0 0

Index	Eintrag
1	< , >
2	< , >
3	< , >
4	< , >
5	< , >
6	< , >
7	< , >
8	< , >
9	< , >
10	< , >
11	< , >
12	< , >

b) LZW

0 0 1010210210212021021200  
0 1 010210210212021021200  
1 0 10210210212021021200  
01 0 210210212021021200  
0 2 10210212021021200  
2 1 0210212021021200  
10 2 10210212021021200  
21 0 21021021200  
02 1 2021021200  
1 2 021021200  
2 0 21021200  
021 0 21200  
021 2 00  
20 0  
0

Index	Eintrag
0	0
1	1
2	2
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	



Die folgende Zahlenfolge wurde von einem LZW-Encoder erzeugt:

0, 0, 1, 4, 0, 2, 5, 8, 7, 1, 2, 11, 11, 13, 0

**Aufgabe 4.23**


**Webcode  
4810**

Decodieren Sie diese Sequenz, indem Sie die nachstehende Tabelle ergänzen:

Index	Eintrag	Index	Eintrag	
0	A	5	9	
1	B	8	10	
2	C	7	11	
0	3	1	12	
0	4	2	13	
1	5	11	14	
4	6	11	15	
0	7	13	16	
2	8	0	17	

Kann die folgende Zahlenfolge von einem LZW-Encoder erzeugt worden sein?

0, 0, 1, 4, 0, 2, 5, 8, 7, 1, 1, 11, 11, 13, 0

**Aufgabe 4.24**


**Webcode  
4817**

Finden Sie die Antwort, indem Sie die nachstehende Tabelle ergänzen:

Index	Eintrag	Index	Eintrag	
0	A	5	9	
1	B	8	10	
2	C	7	11	
0	3	1	12	
0	4	1	13	
1	5	11	14	
4	6	11	15	
0	7	13	16	
2	8	0	17	

**Aufgabe 4.25**

**Webcode  
4851**

In dieser Aufgabe geht es um die folgenden Bitsequenzen, die beide die gleiche Nachricht codieren:

1. Sequenz : 001000011110000011100001010000

2. Sequenz : 00000010000000000001100000110000

Eine davon wurde von einem LZ78-Encoder erzeugt und die andere von einem LZW-Encoder. Wir wissen, dass die Originalnachricht aus den Symbolen A, B, E, N und U besteht und dass die Encoder für die Codierung der Symbole die folgende Zuordnung verwendet haben:

A $\mapsto$ 000	B $\mapsto$ 001	E $\mapsto$ 010	N $\mapsto$ 011	U $\mapsto$ 100
-----------------	-----------------	-----------------	-----------------	-----------------

- Finden Sie anhand der Längen heraus, welche Sequenz mit dem LZ78-Encoder und welche mit dem LZW-Encoder erstellt wurde.
- Leider enthalten die Bitsequenzen einige unleserliche Stellen. Versuchen Sie, die Originalnachricht trotzdem vollständig zu rekonstruieren.

**Aufgabe 4.26**

**Webcode  
4876**

Das Unix-Werkzeug compress ist der Vorgänger des bekannten Kompressionsprogramms gzip. Im Kern basiert es auf der LZW-Kompression, die wir in Abschnitt 4.7.4 besprochen haben.

*„The compress utility reduces the size of files using adaptive Lempel-Ziv coding. Each file is renamed to the same name plus the extension .Z. A file argument with a .Z extension will be ignored except it will cause an error exit after other arguments are processed. If compression would not reduce the size of a file, the file is ignored.“*

BSD General Commands Manual

Wenden wir compress auf die Nachricht ABABCBABABAAAAAAA an, so erhalten wir die folgende Ausgabe:

```
hexdump ababcbababaaaaaaa.txt.Z
```



```
1F 9D 90 41 84 04 1C 22 b0 60 90 83      ...A...".`...
08 83 28 00      ..(.
```

- a) Übersetzen Sie die Hexadezimalausgabe in eine Bitsequenz.
- b) Die von compress produzierte Sequenz ist von jener, die wir auf Seite 287 ausgerechnet haben, völlig verschieden. Welche Gründe könnten hierfür verantwortlich sein?

Wenden Sie die Burrows-Wheeler-Transformation auf die Nachricht BANANENANBAU an:

**Aufgabe 4.27**



**Webcode  
4945**

Rotationsmatrix												U	Sortierte Rotationsmatrix											
0	B	A	N	A	N	E	N	A	N	B	A	U	0	1	2	3	4	5	6	7	8	9	10	11
1													0	1	2	3	4	5	6	7	8	9	10	11
2													0	1	2	3	4	5	6	7	8	9	10	11
3													0	1	2	3	4	5	6	7	8	9	10	11
4													0	1	2	3	4	5	6	7	8	9	10	11
5													0	1	2	3	4	5	6	7	8	9	10	11
6													0	1	2	3	4	5	6	7	8	9	10	11
7													0	1	2	3	4	5	6	7	8	9	10	11
8													0	1	2	3	4	5	6	7	8	9	10	11
9													0	1	2	3	4	5	6	7	8	9	10	11
10													0	1	2	3	4	5	6	7	8	9	10	11
11													0	1	2	3	4	5	6	7	8	9	10	11

Ergebnis: ( , )

In ihrer Originalarbeit erklären Michael Burrows und David Wheeler die nach ihnen benannte Transformation anhand einer Beispieldnachricht über dem Quellenalphabet  $\Sigma = \{a, b, c, r\}$ .

Über die gewählte Nachricht wissen wir das Folgende:

**Aufgabe 4.28**



**Webcode  
4970**

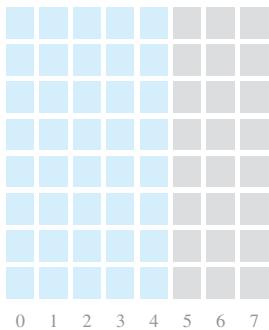
- Der Burrows-Wheeler-Transformator liefert den Index  $I = 1$ .
- Der Move-to-front-Codierer liefert das Ergebnis 2, 1, 3, 1, 0, 3.

Rekonstruieren Sie aus diesen Informationen die ursprüngliche Nachricht. Gehen Sie davon aus, dass der Move-to-front-Codierer mit einer Symboliste startet, in der die Elemente des Quellenalphabets lexikografisch sortiert sind.

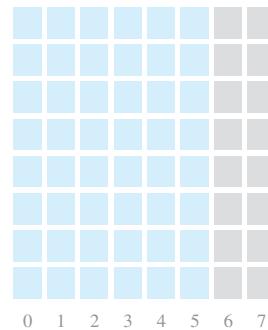
**Aufgabe 4.29****Webcode  
4983**

In Abbildung 4.56 haben wir die inverse Burrows-Wheeler-Transformation am Beispiel des Paars (RRIIEEEB,2) demonstriert. Führen Sie die Decodierungsschritte aus, die in der Abbildung aus Platzgründen ausgespart wurden.

9. Voranstellen



10. Sortieren



# 5 Grenzen der Quellencodierung

In diesem Kapitel werden Sie ...

- verstehen, was sich hinter dem Informationsgehalt eines Zeichens und
- der Entropie einer Datenquelle verbirgt,
- den erarbeiteten Begriffsapparat auf gedächtnisbehaftete Quellen übertragen,
- blockweise arbeitende Codierungen kennenlernen und
- das Quellencodierungstheorem von Shannon beweisen.



## 5.1 Motivation



**Abb. 5.1:** Jeder Ring steht für die Menge der Bitsequenzen einer bestimmten Länge. Eine Nachricht wird komprimiert, wenn sie von einem weiter außen liegenden Ring auf einen weiter innen liegenden Ring abgebildet wird.

In Kapitel 4 haben wir mehrere Kompressionsverfahren, zusammen mit deren Vor- und Nachteilen, vorgestellt. Alle Verfahren arbeiteten verlustfrei, d. h., es war immer möglich, die ursprüngliche Nachricht aus der komprimierten Symbolfolge originalgetreu zu rekonstruieren.

Tatsächlich hat die verlustfreie Rekonstruierbarkeit einschneidende Konsequenzen. Um diese zu verstehen, betrachten wir ein beliebiges Kompressionsverfahren, das Bitsequenzen der Länge  $\leq n$  verarbeitet und als Ausgabe ebenfalls wieder Bitsequenzen produziert. Die in Frage kommenden Sequenzen können wir uns in Ringen angeordnet vorstellen, wie sie in Abbildung 5.1 zu sehen sind. Der innere Ring umfasst die Sequenzen der Länge 1, und in den angrenzenden Ringen befinden sich die Sequenzen der Länge 2, 3 und so fort. Intuitiv würden wir von einem guten Kompressionsverfahren erwarten, dass es die vorgelegten Bitsequenzen verkürzt. Manche Sequenzen würde das Verfahren stärker und andere weniger stark komprimieren, und vielleicht würden wir für die eine oder die andere Sequenz auch akzeptieren, dass die produzierte Ausgabe genauso lang ist wie die Eingabe. Eine Verlängerung würde dem Sinn der Kompression jedoch gehörig widersprechen.

Doch ist die Konstruktion eines solchen Verfahrens überhaupt möglich? Die Antwort ist Nein und anhand unseres Ringmodells einfach zu verstehen (Abbildung 5.2). Die Verkürzung einer Nachricht bedeutet, dass die Binärsequenz eines weiter außen liegenden Ringes auf eine Binärsequenz eines weiter innen liegenden Ringes abgebildet wird. Da sich die Anzahl der Binärsequenzen von Ring zu Ring verdoppelt, bieten die inneren Regionen zu wenig Platz, damit alle Nachrichten verkürzt werden können. Schlimmer noch: Jede Nachricht, die durch ein Kompressionsverfahren verkürzt wird, nimmt in den inneren Ringen ein Codewort weg, das für die Codierung der Nachrichten innerhalb dieser Ringe dann nicht mehr zur Verfügung steht. Dieser Mangel an Codewörtern lässt sich nur dadurch ausgleichen, dass manche Sequenzen der inneren Ringe auf Sequenzen der äußeren Ringe abgebildet, und damit verlängert werden.

Damit haben wir ein Ergebnis erzielt, das auf den ersten Blick ernüchternd wirkt:

Für jedes verlustfreie Kompressionsverfahren  $K$  gilt:

*Gibt es Nachrichten, die durch  $K$  verkürzt werden, so gibt es auch welche, die durch  $K$  verlängert werden.*

Beachten Sie, dass sich die gewählte Formulierung auf Kompressionscodierungen bezieht, in denen das Quellenalphabet und das Codealphabet identisch sind. Auf andere Codierungen ließe sich das Ergebnis sinngemäß übertragen, allerdings müssten wir dann genauer festlegen, was eine Verkürzung bzw. eine Verlängerung in diesen Fällen bedeutet.

Damit sind wir in der Lage, präziser zu formulieren, was ein gutes Kompressionsverfahren auszeichnet: Es muss gewährleisten, dass möglichst alle der häufig vorkommenden Nachrichten verkürzt und lediglich selten vorkommenden Nachrichten verlängert werden. Die bisher diskutierten Codierungen haben genau dies geleistet: Substitutionscodierungen werden für die Kompression von Nachrichten verwendet, die viele, sich wiederholende Textfragmente enthalten, und funktionieren in diesen Anwendungsszenarien exzellent. Entropiedcodierungen verhalten sich ganz ähnlich. Sie funktionieren umso besser, je *typischer* eine Nachricht ist, d. h., je eher die einzelnen Symbole mit den Wahrscheinlichkeiten vorkommen, wie sie von der Datenquelle im Durchschnitt emittiert werden.

Am Ende dieses Kapitels wird die Erkenntnis stehen, dass es exakt die *typischen Nachrichten* sind, die wir bei der Kompression beachten müssen, und wir alle anderen in zunehmendem Maße ignorieren dürfen. Der Fehler, den wir dabei machen, wird mit steigender Nachrichtenlänge beliebig klein. Dies wird uns in die Lage versetzen, die maximal mögliche Kompressionsrate durch eine klar definierbare Schranke nach unten abzuschätzen und die Güte von Kompressionsalgorithmen quantitativ zu erfassen. An dieses genauso erstaunliche wie spannende Ergebnis werden wir uns nun schrittweise herantasten.

## 5.2 Entropie, Information, Redundanz

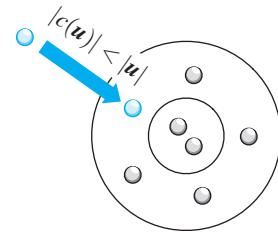
Wir beginnen mit einem Satz, der uns eine wichtige untere Schranke für die mittlere Codewortlänge  $L(c)$  einer präfixfreien Codierung  $c$  liefert:

### Satz 5.1

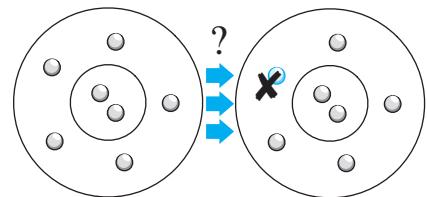
Es seien  $X$  eine Datenquelle und  $p_1, \dots, p_n$  die Auftrittswahrscheinlichkeiten der emittierten Quellsymbole. Dann erfüllt jede präfixfreie Binärcodierung  $c$  die Beziehung

$$\sum_i p_i \log_2 \frac{1}{p_i} \leq L(c)$$

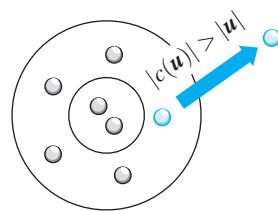
*Wird eine Nachricht durch ein verlustfreies Kompressionsverfahren verkürzt, ...*



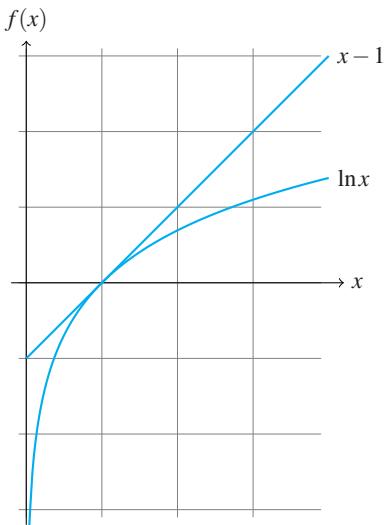
*... so enthalten die inneren Ringe nicht mehr genügend Bitsequenzen, um eine Selbstabbildung zu ermöglichen.*



*Die Knappheit der Codewörter sorgt dafür, dass es Nachrichten geben muss, die verlängert werden.*



**Abb. 5.2:** Eine injektive Abbildung, die jede Nachricht verkürzt, kann es nicht geben.



**Abb. 5.3:** Der natürliche Logarithmus wird durch die Funktion  $x - 1$  nach oben begrenzt. Lediglich an der Stelle  $x = 1$  berühren sich die beiden Graphen.

*Beweis:* Es seien  $l_1, \dots, l_n$  die Längen der Codewörter. Dann können wir die folgende Umformung vornehmen:

$$\begin{aligned} \sum_i p_i \log_2 \frac{1}{p_i} - L &= \sum_i p_i \log_2 \frac{1}{p_i} - \sum_i p_i l_i = \sum_i \left( p_i \log_2 \frac{1}{p_i} - p_i l_i \right) \\ &= \sum_i p_i \left( \log_2 \frac{1}{p_i} - l_i \right) = \sum_i p_i \left( \log_2 \frac{1}{p_i} + \log_2 2^{-l_i} \right) \\ &= \sum_i p_i \log_2 \frac{2^{-l_i}}{p_i} \end{aligned} \quad (5.1)$$

Aus der bekannten Ungleichung  $\ln x \leq x - 1$  (Abbildung 5.3) folgt

$$\log_2 x = \log_2 e^{\ln x} = (\log_2 e) \ln x \leq (\log_2 e)(x - 1),$$

womit wir (5.1) folgendermaßen abschätzen können:

$$\begin{aligned} \sum_i p_i \log_2 \frac{2^{-l_i}}{p_i} &\leq \sum_i p_i (\log_2 e) \left( \frac{2^{-l_i}}{p_i} - 1 \right) \\ &= (\log_2 e) \sum_i \left( 2^{-l_i} - p_i \right) = (\log_2 e) \left( \sum_i 2^{-l_i} - \sum_i p_i \right) \\ &= (\log_2 e) \left( \sum_i 2^{-l_i} - 1 \right) \end{aligned} \quad (5.2)$$

Da  $c$  präfixfrei ist, gilt nach Satz 3.1 die Kraft'sche Ungleichung

$$\sum_i 2^{-l_i} \leq 1$$

Angewendet auf (5.2) erhalten wir

$$\sum_i p_i \log_2 \frac{1}{p_i} - L \leq (\log_2 e)(1 - 1) = 0$$

Hieraus folgt unmittelbar

$$\sum_i p_i \log_2 \frac{1}{p_i} \leq L,$$

was zu beweisen war.  $\square$

Ein vergleichender Blick auf die Abbildungen 4.13 und 5.4 zeigt, dass die mittleren Codewortlängen der auf Seite 242 aufgeführten Beispieldcodierungen von der unteren Schranke

$$\sum_i p_i \log_2 \frac{1}{p_i} \quad (5.3)$$

nicht allzu weit entfernt sind. Der folgende Satz untermauert, dass dies nichts Außergewöhnliches ist: Für jede Datenquelle, ob gedächtnislos oder gedächtnisbehaftet, können wir eine präfixfreie Codierung finden, deren mittlere Codewortlänge um höchsten 1 größer ist als die angegebene Schranke:

### Satz 5.2

Es seien  $X$  eine Datenquelle und  $p_1, \dots, p_n$  die Auftrittswahrscheinlichkeiten der emittierten Quellsymbole. Dann gibt es eine präfixfreie Codierung  $c$  mit der Eigenschaft

$$L(c) < \sum_i p_i \log_2 \frac{1}{p_i} + 1$$

*Beweis:* Wir zeigen zunächst, dass eine präfixfreie Codierung existiert, deren Codewörter die Längenbeziehungen

$$l_i = \left\lceil \log_2 \frac{1}{p_i} \right\rceil \quad (5.4)$$

erfüllen. (5.4) ist äquivalent zu den folgenden beiden Forderungen:

$$l_i \geq \log_2 \frac{1}{p_i} \quad (5.5)$$

$$l_i < \log_2 \frac{1}{p_i} + 1 \quad (5.6)$$

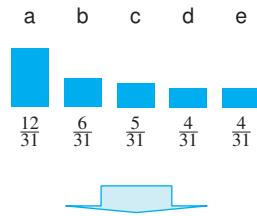
Aus (5.5) folgt  $-l_i \leq -\log_2 \frac{1}{p_i} = \log_2 p_i$  und hieraus wiederum

$$\sum_i 2^{-l_i} \leq \sum_i 2^{\log_2 p_i} = \sum_i p_i = 1$$

Das bedeutet, dass die gewählten Werte  $l_1, \dots, l_n$  die Kraft'sche Ungleichung erfüllen. Somit existiert nach Satz 3.1 eine präfixfreie Codierung  $c$  mit den vorgegebenen Codewortlängen. Mithilfe von (5.6) können wir die mittlere Codewortlänge  $L(c)$  dann folgendermaßen abschätzen:

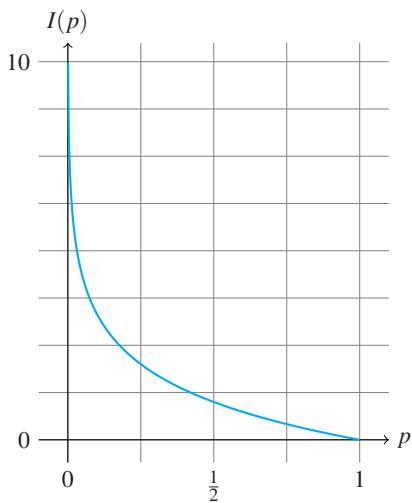
$$\begin{aligned} L(c) &= \sum_i p_i l_i < \sum_i p_i \left( \log_2 \frac{1}{p_i} + 1 \right) \\ &= \sum_i p_i \log_2 \frac{1}{p_i} + \sum_i p_i = \sum_i p_i \log_2 \frac{1}{p_i} + 1 \end{aligned}$$

Damit ist die Behauptung bewiesen.  $\square$



$$\begin{aligned} \sum_i p_i \log_2 \frac{1}{p_i} &= \frac{12}{31} \cdot \log_2 \frac{31}{12} \\ &\quad + \frac{6}{31} \cdot \log_2 \frac{31}{6} \\ &\quad + \frac{5}{31} \cdot \log_2 \frac{31}{5} \\ &\quad + \frac{4}{31} \cdot \log_2 \frac{31}{4} \\ &\quad + \frac{4}{31} \cdot \log_2 \frac{31}{4} \approx 2,17552 \end{aligned}$$

**Abb. 5.4:** Untere Schranke für die mittlere Codewortlänge präfixfreier Codierung



**Abb. 5.5:** Der Informationsgehalt eines Zeichens nimmt mit steigender Auftrittswahrscheinlichkeit kontinuierlich ab.

Offensichtlich ist (5.3) eine wichtige Größe. Sie definiert eine untere Schranke für die mittlere Codewortlänge, die wir mit präfixfreien Codierungen nicht durchbrechen können. Ist  $X$  eine gedächtnislose Quelle, so wird sie als die *Entropie* von  $X$  bezeichnet und mit dem Symbol  $H(X)$  abgekürzt. Die Abkürzung geht auf das griechische Alphabet zurück. Dort ist  $H$  das Symbol für das große Eta.

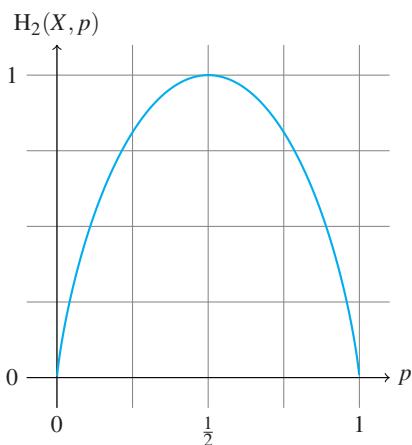


### Definition 5.1 (Entropie gedächtnisloser Quellen)

Für eine *gedächtnislose* Quelle  $X$ , die ihre Symbole  $\sigma_1, \dots, \sigma_n$  mit den Wahrscheinlichkeiten  $p_1, \dots, p_n$  emittiert, nennen wir

$$H(X) := \sum_i p_i \log_2 \frac{1}{p_i}$$

die *Entropie* von  $X$ .



**Abb. 5.6:** Entropie einer Bernoulli-Quelle, die ihre beiden Symbole mit den Wahrscheinlichkeiten  $p$  und  $1 - p$  emittiert.

Wir können dem Entropiebegriff eine anschauliche Bedeutung verleihen, wenn wir uns an die Diskussion des Informationsbegriffs aus Abschnitt 3.6 erinnern. Dort haben wir einem Ereignis, das mit der Wahrscheinlichkeit  $p$  eintritt, den Informationsgehalt  $\log_2 \frac{1}{p}$  zugewiesen. Diesen Begriff können wir eins zu eins auf die Symbole einer gedächtnislosen Datenquelle übertragen (Abbildung 5.5):



### Definition 5.2 (Informationsgehalt eines Zeichens)

Für eine *gedächtnislose* Quelle  $X$ , die ihre Symbole  $\sigma_1, \dots, \sigma_n$  mit den Wahrscheinlichkeiten  $p_1, \dots, p_n$  emittiert, nennen wir

$$I(\sigma_i) := \log_2 \frac{1}{p_i}$$

den *Informationsgehalt* von  $\sigma_i$ .

Damit können wir die Definition von  $H(X)$  folgendermaßen umformulieren:

$$H(X) = \sum_i p_i I(\sigma_i)$$

Jetzt haben wir die Bedeutung schwarz auf weiß vor uns. Die Entropie einer gedächtnislosen Datenquelle  $X$  ist die Information, gemessen in Bit, die  $X$  pro Symbol im Durchschnitt emittiert.

Damit sind wir in der Lage, den in Abbildung 5.4 ausgerechneten Wert inhaltlich zu interpretieren. Er bedeutet, dass die dort gezeigte Datenquelle, die wir in diesem Kapitel schon mehrfach als Beispiel herangezogen haben, pro Symbol ca. 2,17552 Bit an Information emittiert.

Als Nächstes Beispiel betrachten wir eine Bernoulli-Quelle  $X$ , d. h. eine gedächtnislose Datenquelle mit dem zweielementigen Quellenalphabet  $\{0, 1\}$ . Ist  $p$  die Auftrittswahrscheinlichkeit des Symbols 0, so emittiert die Quelle das Symbol 1 mit der Wahrscheinlichkeit  $1 - p$ . Damit können wir die Entropie von  $X$  als eine Funktion von  $p$  betrachten und folgendermaßen ausrechnen:

$$\begin{aligned} H_2(X, p) &= p \log_2 \frac{1}{p} + (1 - p) \log_2 \frac{1}{1 - p} \\ &= -p \log_2 p - (1 - p) \log_2 (1 - p) \end{aligned}$$

Ein Blick auf Abbildung 5.6 macht zwei Besonderheiten dieser Funktion deutlich:

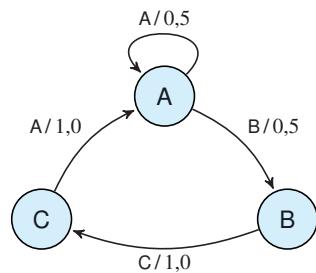
- Die Funktion hat bei  $p = 0$  und  $p = 1$  jeweils eine Nullstelle, und das bedeutet, dass die Datenquelle  $X$  in diesen beiden Fällen überhaupt keine Information erzeugt. In der Tat steht dies im Einklang mit unserem Informationsbegriff, den wir als ein quantitatives Maß für die Reduktion der Unsicherheit definiert haben. Ist  $p = 0$  oder  $p = 1$ , so emittiert die Quelle stets das gleiche Symbol. Das bedeutet, dass wir die emittierte Sequenz direkt vorhersagen können, ohne die Quelle zu beobachten. Durch die Emission eines Symbols gewinnen wir also tatsächlich keine Information.
- Die Entropie erreicht ihr Maximum, wenn die Symbole 0 und 1 mit der gleichen Wahrscheinlichkeit emittiert werden. Es ist

$$H_2(X, \frac{1}{2}) = 1$$

Das heißt, dass die Quelle pro Symbol exakt 1 Bit an Information emittiert. In Abschnitt 5.3 werden wir daraus ein interessantes Ergebnis über die Komprimierbarkeit verschlüsselter Nachrichten ableiten.

## Erweiterung auf gedächtnisbehaftete Quellen

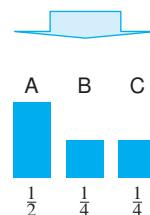
Wir rekapitulieren: Im Falle von gedächtnislosen Quellen entspricht die Entropie  $H(X)$  der Information, gemessen in Bit, die eine gedächtnislose Datenquelle pro Symbol im Durchschnitt emittiert. Die Markov-Quelle in Abbildung 5.7 macht deutlich, dass diese Bedeutung verloren



Typische Sequenzen:

AABC...  
BCABC...  
AABC...  
BCAA...  
BCAAAB...

BCAABC...  
ABCAA...  
BCAAABC...  
BCAAABCA...

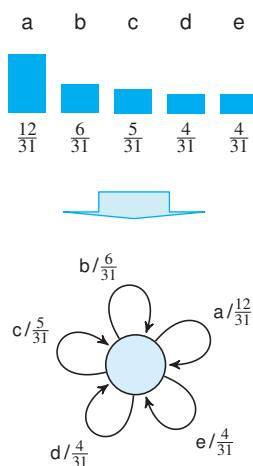


**Abb. 5.7:** Diese Markov-Quelle der Ordnung 1 ist so gestaltet, dass die Beobachtung des Zeichens C keinerlei Information liefert. Dies zeigt, dass wir den Entropiebegriff nicht eins zu eins von gedächtnislosen auf gedächtnisbehaftete Quellen übertragen dürfen.

Das Wort *Entropie* geht nicht auf Claude Shannon zurück. In die Wissenschaft hielt es bereits im Jahr 1864 Einzug, in der Arbeit *Abhandlungen über die mechanische Wärmetheorie* von Rudolf Clausius [17]. In der Thermodynamik ist die Entropie heute ein allgegenwärtiger Begriff, und in der Vergangenheit wurde gezeigt, dass sich tatsächlich Parallelen zwischen dem informationstheoretischen Entropiebegriff und seinem sprachlichen Pendant aus der Physik finden lassen [53, 95]. Die Autoren von [95] legen dar, dass Shannon den Namen eher zufällig wählte und dabei einem Vorschlag des Mathematikers und Computerwissenschaftlers John von Neumann folgte:



*„In 1961 one of us (Tribus) asked Shannon what he had thought about when he finally confirmed his famous measure. Shannon replied: ‘My greatest concern was what to call it. I thought of calling it ‘information’, but the word was overly used, so I decided to call it ‘uncertainty’. When I discussed it with John von Neumann, he had a better idea. Von Neumann told me, ‘You should call it entropy, for two reasons. In the first place your uncertainty function has been used in statistical mechanics under the name, so it already has a name. In the second place, and more important, no one knows what entropy really is, so in a debate you will always have the advantage.’“*



**Abb. 5.8:** Gedächtnislose Quellen lassen sich als Zustandsdiagramme mit genau einem Zustand beschreiben.

geht, wenn wir von gedächtnislosen zu gedächtnisbehafteten Quellen übergehen. Die Quelle emittiert die Symbole A, B und C und besitzt drei Zustände. Beobachten wir die Markov-Quelle über einen langen Zeitraum, so werden die Quellsymbole mit den folgenden Wahrscheinlichkeiten auftreten:

$$p(A) = \frac{1}{2}, \quad p(B) = \frac{1}{4}, \quad p(C) = \frac{1}{4}$$

Würden wir den Quellsymbolen einen Informationsgehalt zuordnen, wie wir es in Definition 5.2 für gedächtnislose Quellen vereinbart haben, entspräche die Emission des Zeichens C beispielsweise einem Informationsgewinn von 2 Bit. Tatsächlich gewinnen wir bei der Beobachtung dieses Zeichens überhaupt keine Information hinzu; die Markov-Quelle emittiert ein C nämlich genau dann, wenn vorher ein B emittiert wurde. Das bedeutet, dass wir für das Zeichen C zu jedem Zeitpunkt vorhersagen können, ob es erscheinen wird oder nicht.

Aus diesem Grund werden wir nun eine allgemeinere Berechnungsformel entwickeln, die für gedächtnislose wie gedächtnisbehaftete Quellen gleichermaßen gilt. Die Bedeutung der Entropie wird sich dabei nicht ändern. Ihr Wert wird auch weiterhin der Information entsprechen, die eine Datenquelle pro Symbol im Durchschnitt emittiert.

Bevor wir den Entropiebegriff auf gedächtnisbehaftete Quellen übertragen, erinnern wir uns daran, dass jede gedächtnislose Quelle, wie in Abbildung 5.8 gezeigt, mithilfe eines Zustandsdiagramms modelliert werden kann. Ein solches Diagramm verfügt über einen einzigen Zustand, und für jedes Zeichen  $\sigma_i$  existiert eine ausgehende Kante, die mit der Wahrscheinlichkeit  $p_i$  beschriftet ist. Geben wir diesem Zustand die Nummer 1 und bezeichnen die Wahrscheinlichkeit, dass die Datenquel-

le diesen Zustand entlang der  $j$ -ten Kante verlässt, mit  $p_{1j}$ , so können wir die uns vertraute Entropieformel so aufschreiben:

$$H(X) := \sum_j p_{1j} \log_2 \frac{1}{p_{1j}}$$

Gedächtnisbehaftete Quellen besitzen mehrere Zustände, denen wir jeweils eine individuelle Entropie  $H_i(X)$  zuordnen können. Ist  $p_{ij}$  die Wahrscheinlichkeit, dass die Datenquelle den Zustand  $i$  entlang der  $j$ -ten Kante verlässt, so können wir  $H_i(X)$  über die Formel

$$H_i(X) := \sum_j p_{ij} \log_2 \frac{1}{p_{ij}}$$

berechnen. In Worten ausgedrückt ist  $H_i(X)$  die Information, gemessen in Bit, die  $X$  im Zustand  $i$  pro Symbol im Durchschnitt emittiert. Bezeichnen wir die Wahrscheinlichkeit, dass sich die Quelle  $X$  im  $i$ -ten Zustand aufhält, mit  $P_i$ , so können wir die Information, die  $X$  pro Symbol im Durchschnitt emittiert, ganz einfach als gewichtete Summe berechnen. Wir erhalten dann

$$\begin{aligned} H(X) &= \sum_i P_i H_i(X) \\ &= \sum_i P_i \sum_j p_{ij} \log_2 \frac{1}{p_{ij}} = \sum_{i,j} P_i p_{ij} \log_2 \frac{1}{p_{ij}} \end{aligned}$$

Mit dieser Formel ist es uns gelungen, den Entropiebegriff erfolgreich auf gedächtnisbehaftete Quellen zu übertragen:



### Definition 5.3 (Entropie)

Für eine Datenquelle, die sich mit der Wahrscheinlichkeit  $P_i$  im  $i$ -ten Zustand aufhält, nennen wir

$$H(X) := \sum_i P_i H_i(X) = \sum_{i,j} P_i p_{ij} \log_2 \frac{1}{p_{ij}}$$

die *Entropie* von  $X$ .

Diese Definition gilt für gedächtnislose und gedächtnisbehaftete Datenquellen gleichermaßen. Für gedächtnislose Quellen degeneriert die angegebene Formel zu jener vereinfachten Form, die wir in Definition 5.1 benutzt haben.

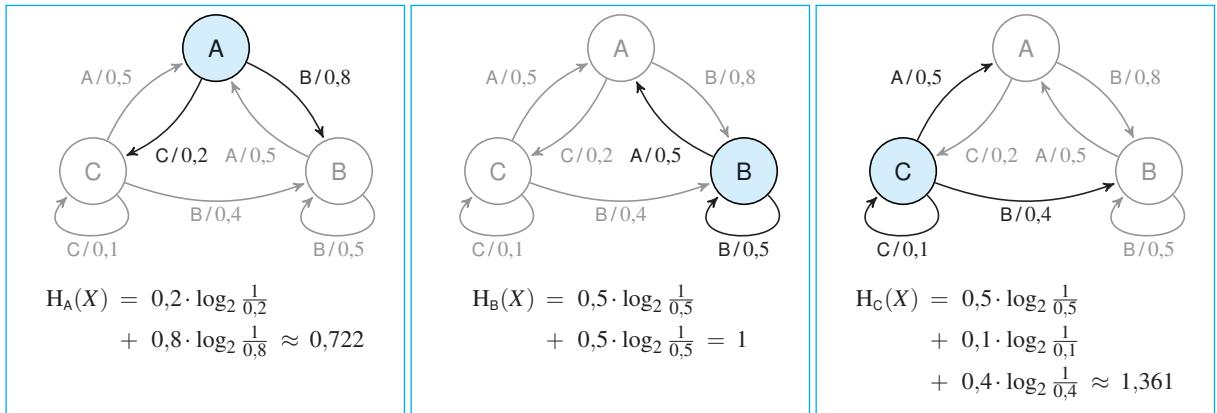
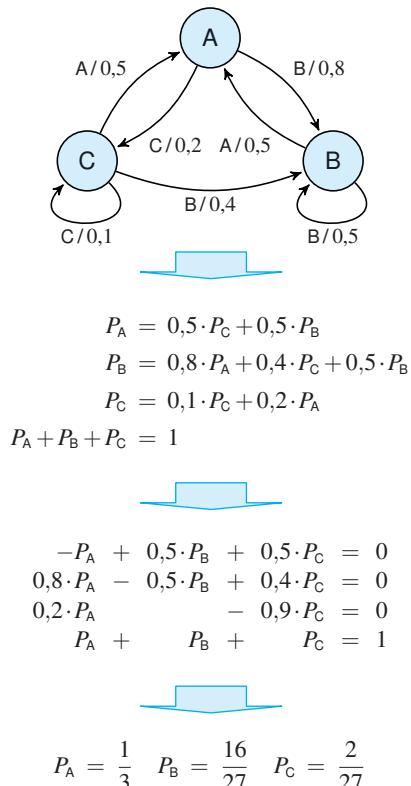


Abb. 5.9: Berechnung der individuellen Entropien

Abb. 5.10: Berechnung der Aufenthaltswahrscheinlichkeiten  $P_A$ ,  $P_B$  und  $P_C$ 

Als Beispiel wollen wir die Entropie der Markov-Quelle aus Abbildung 4.8 bestimmen. Das Zustandsdiagramm dieser Quelle besteht aus drei Zuständen, für die wir, im ersten Schritt, die individuellen Entropien berechnen müssen. Das Ergebnis ist in Abbildung 5.9 zu sehen.

Die Aufenthaltswahrscheinlichkeiten  $P_A$ ,  $P_B$  und  $P_C$  gehen ebenfalls in die Berechnung ein; sie geben an, mit welcher Wahrscheinlichkeit sich die Markov-Quelle in den Zuständen A, B bzw. C aufhält. Die Wahrscheinlichkeiten lassen sich zwar nicht direkt aus dem Diagramm ableSEN, dafür aber indirekt über die Kantenmarkierungen charakterisieren. Abbildung 5.10 zeigt, welche Gleichungen dabei entstehen, und wie diese in ein lineares Gleichungssystem übersetzt werden können. Das Gleichungssystem lässt sich eindeutig lösen und führt zu dem folgenden Ergebnis:

$$P_A = \frac{1}{3} \quad P_B = \frac{16}{27} \quad P_C = \frac{2}{27}$$

Jetzt können wir die Entropie der Datenquelle direkt ausrechnen. Es ist

$$\begin{aligned} H(X) &= P_A \cdot H_A(X) + P_B \cdot H_B(X) + P_C \cdot H_C(X) \\ &\approx \frac{1}{3} \cdot 0,722 + \frac{16}{27} \cdot 1 + \frac{2}{27} \cdot 1,361 \approx 0,934 \end{aligned}$$

Das bedeutet, dass die Datenquelle pro Symbol im Durchschnitt etwas weniger als 1 Bit an Information emittiert.

Wir beenden diesen Abschnitt mit der Einführung eines letzten Begriffs:



### Definition 5.4 (Redundanz)

Sei  $X$  eine Datenquelle und  $c$  eine Codierung. Die Differenz

$$R(X, c) := L(c) - H(X)$$

heißt *Redundanz*. Eine Codierung mit  $R = 0$  ist *redundanzfrei*.

Ist  $X$  eine gedächtnislose Quelle, so ist die Redundanz nach Satz 5.1 für jeden präfixfreien Code  $\geq 0$ . Sie drückt aus, wie weit die mittlere Codewortlänge von der Entropieschranke entfernt ist, und eignet sich als ein quantitatives Maß, um die Güte von Codierungen zu vergleichen (Abbildung 5.11).

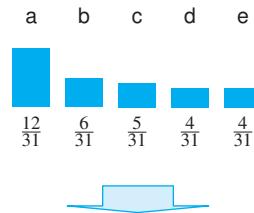
Dass die Größe  $R$  ihren Namen verdient, wird im nächsten Abschnitt in voller Pracht ersichtlich werden. Dort werden wir herleiten, dass die Beziehung  $R \geq 0$  eine universelle ist: Sie gilt für ausnahmslos jede Datenquelle  $X$  und jede Codierung  $c$ .

## 5.3 Das Quellencodierungstheorem

In diesem Abschnitt werden wir in mehreren Schritten Shannons berühmtes Quellencodierungstheorem herleiten. Wir beginnen im nächsten Abschnitt mit dem Beweis, dass die Eigenschaft der Entropie, eine untere Schranke für die Kompression einer Nachricht zu bilden, nicht nur für präfixfreie, sondern für sämtliche Codierungen gilt, und danach werden wir zeigen, dass sich die Schranke mit dem Prinzip der blockweisen Codierung beliebig annähern lässt. Abschließend fassen wir die erarbeiteten Ergebnisse zusammen und erhalten auf diese Weise das Shannon'sche Quellencodierungstheorem in dessen historischer Formulierung.

Im Folgenden betrachten wir eine gedächtnislose Datenquelle  $X$  und eine typische, von dieser Quelle emittierte Nachricht der Länge  $N$ . Wählen wir für  $N$  immer größere Werte, so steigt die Wahrscheinlichkeit kontinuierlich an, dass die Nachricht annähernd  $p_1N$ -mal das Symbol  $\sigma_1$ , enthält,  $p_2N$ -mal das Symbol  $\sigma_2$  und so fort. Die Auftrittswahr-

### Datenquelle



$$H(X) = 2,17552$$

### Codierung $c_1$

$a \mapsto 00$	$b \mapsto 011$	$c \mapsto 100$
$d \mapsto 101$	$e \mapsto 110$	

$$\begin{aligned} L(c_1) &= 2,61290 \\ - H(X) &= 2,17552 \\ \hline R(X, c_1) &= 0,43738 \end{aligned}$$

### Codierung $c_2$

$a \mapsto 00$	$b \mapsto 01$	$c \mapsto 10$
$d \mapsto 110$	$e \mapsto 111$	

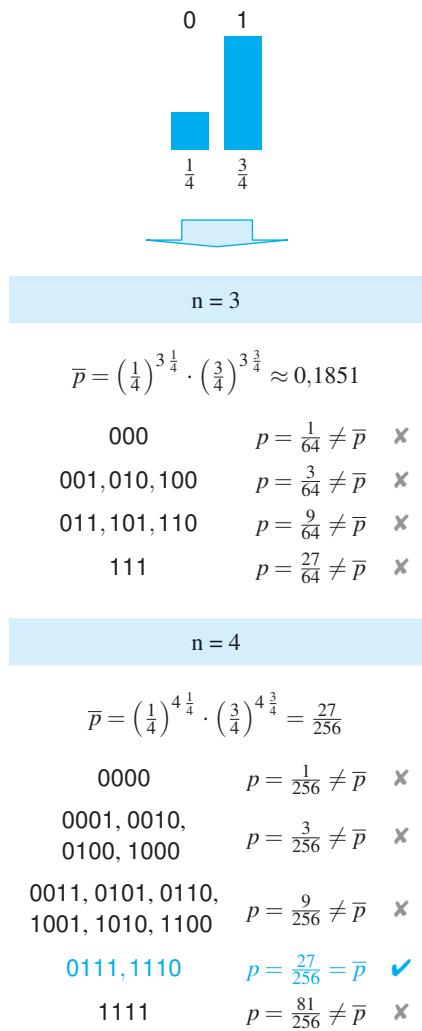
$$\begin{aligned} L(c_2) &= 2,25806 \\ - H(X) &= 2,17552 \\ \hline R(X, c_2) &= 0,08254 \end{aligned}$$

### Codierung $c_3$

$a \mapsto 0$	$b \mapsto 100$	$c \mapsto 101$
$d \mapsto 110$	$e \mapsto 111$	

$$\begin{aligned} L(c_3) &= 2,22581 \\ - H(X) &= 2,17552 \\ \hline R(X, c_3) &= 0,05029 \end{aligned}$$

Abb. 5.11: Redundanzberechnung



**Abb. 5.12:** Weist eine Nachricht eine Länge auf, die nicht durch 4 teilbar ist, so kann sie keinesfalls die in (5.7) formulierte Auftrittswahrscheinlichkeit  $\bar{p}$  besitzen.

scheinlichkeit einer typischen Sequenz ist damit ungefähr

$$\begin{aligned} \bar{p} &\approx \underbrace{p_1 \cdot \dots \cdot p_1}_{p_1 N\text{-mal}} \cdot \underbrace{p_2 \cdot \dots \cdot p_2}_{p_2 N\text{-mal}} \cdot \dots \cdot \underbrace{p_n \cdot \dots \cdot p_n}_{p_n N\text{-mal}} \\ &= p_1^{p_1 N} \cdot p_2^{p_2 N} \cdot \dots \cdot p_n^{p_n N} = \prod_i p_i^{p_i N} \end{aligned} \quad (5.7)$$

Ohne das Wörtchen „ungefähr“ wäre das Gesagte falsch. Beispielsweise existiert für die Bernoulli-Quelle aus Abbildung 5.12 nur dann eine Sequenz mit der exakten Wahrscheinlichkeit von  $\bar{p}$ , wenn  $N$  eine durch 4 teilbare Zahl ist. Das bedeutet, dass wir bei der Betrachtung endlicher Nachrichten miteinbeziehen müssen, dass sich die relative Häufigkeit, mit der das Symbol  $\sigma_i$  vorkommt, geringfügig von  $p_i$  unterscheidet. Wir lösen das Problem, indem wir eine Nachricht  $\mathbf{u}$  der Länge  $N$  als typisch erachten, wenn sie für eine vorgegebene Konstante  $\delta > 0$  in der folgenden Menge liegt:

$$T_\delta(N) := \left\{ \mathbf{u} \mid |\mathbf{u}| = N, \prod_i p_i^{(p_i - \delta)N} < p(\mathbf{u}) < \prod_i p_i^{(p_i + \delta)N} \right\} \quad (5.8)$$

Egal, wie klein wir die Konstante  $\delta$  auch wählen: Nach dem Gesetz der großen Zahlen strebt die Wahrscheinlichkeit, dass die Datenquelle eine Sequenz aus der Menge  $T_\delta(N)$  emittiert, für  $N \rightarrow \infty$  gegen 1. Das bedeutet, dass für jedes  $\varepsilon > 0$  ein  $N_0$  existiert, sodass die Wahrscheinlichkeit, eine Sequenz innerhalb der Menge  $T_\delta(N)$  zu beobachten, für alle  $N > N_0$  größer als  $1 - \varepsilon$  ist:

$$1 > \sum_{\mathbf{u} \in T_\delta(N)} p(\mathbf{u}) > 1 - \varepsilon \quad (5.9)$$

Aus diesem Zwischenergebnis ergeben sich tiefgreifende Konsequenzen für die Konstruktion von Kompressionsalgorithmen. Um diese einzusehen, nehmen wir an, uns läge ein Algorithmus vor, der ausschließlich für die Sequenzen aus der Menge  $T_\delta(N)$  ein sinnvolles Ergebnis liefert und für alle anderen abstürzt, unendlich lange weiterläuft oder eine unsinnige Ausgabe produziert. Nach dem oben Gesagten können wir das Fehlverhalten für immer größere Werte von  $N$  zunehmend nachlässigen: Die Wahrscheinlichkeit, dass unser Kompressionsverfahren versagt, strebt gegen 0.

Konkreter können wir diesen Umstand so formulieren: Für jede noch so kleine Fehlerwahrscheinlichkeit  $\varepsilon$  gibt es ein  $N_0$ , sodass die Wahrscheinlichkeit eines Fehlers für alle Eingaben mit einer Länge  $N > N_0$  kleiner als  $\varepsilon$  ist. Diese Überlegung zeigt, dass wir uns bei der asymptotischen Betrachtung des Kompressionsproblems ruhigen Gewissens

auf die Sequenzen aus der Menge  $T_\delta(N)$  beschränken dürfen (Abbildung 5.13).

Damit stehen wir vor der Frage, wie viele typische Sequenzen in der Menge  $T_\delta(N)$  zusammengefasst sind. Hierfür kombinieren wir (5.8) und (5.9) zunächst zu den folgenden beiden Abschätzungen:

$$1 > \sum_{\mathbf{u} \in T_\delta(N)} p(\mathbf{u}) > \sum_{\mathbf{u} \in T_\delta(N)} \prod_i p_i^{(p_i - \delta)N} = |T_\delta(N)| \cdot \prod_i p_i^{(p_i - \delta)N},$$

$$1 - \varepsilon < \sum_{\mathbf{u} \in T_\delta(N)} p(\mathbf{u}) < \sum_{\mathbf{u} \in T_\delta(N)} \prod_i p_i^{(p_i + \delta)N} = |T_\delta(N)| \cdot \prod_i p_i^{(p_i + \delta)N},$$

aus denen wir den folgenden Schluss ziehen können:

$$\frac{1 - \varepsilon}{\prod_i p_i^{(p_i + \delta)N}} < |T_\delta(N)| < \frac{1}{\prod_i p_i^{(p_i - \delta)N}}$$

Durch Logarithmieren erhalten wir

$$\log_2(1 - \varepsilon) - \sum_i (p_i + \delta)N \log_2 p_i < \log_2 |T_\delta(N)| < -\sum_i (p_i - \delta)N \log_2 p_i,$$

was wir weiter umschreiben können, in:

$$\frac{\log_2(1 - \varepsilon)}{N} + \sum_i (p_i + \delta) \log_2 \frac{1}{p_i} < \frac{\log_2 |T_\delta(N)|}{N} < \sum_i (p_i - \delta) \log_2 \frac{1}{p_i}$$

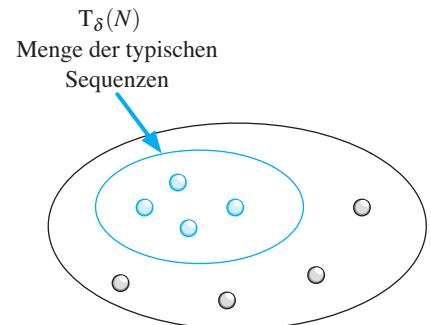
Hieraus folgt

$$\lim_{\substack{\delta \rightarrow 0 \\ N \rightarrow \infty}} \frac{\log_2 |T_\delta(N)|}{N} = \sum_i p_i \log_2 \frac{1}{p_i} = H(X). \quad (5.10)$$

Wählen wir also für  $\delta$  immer kleinere Werte und für  $N$  immer größere, so machen wir einen immer kleineren Fehler, wenn wir die Anzahl der Elemente in der Menge  $T_\delta(N)$  folgendermaßen abschätzen:

$$|T_\delta(N)| \approx 2^{N H(X)} \quad (5.11)$$

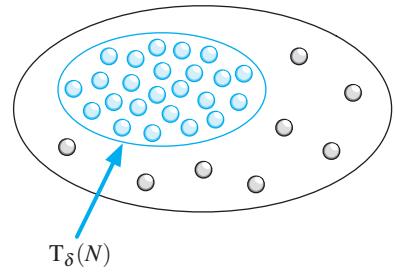
Für sehr große Werte von  $N$  muss ein Kompressionsalgorithmus somit  $2^{N H(X)}$  verschiedene Nachrichten unterscheiden, die alle in etwa gleich wahrscheinlich sind. Um  $2^{N H(X)}$  Nachrichten zu unterscheiden, sind  $N H(X)$  Bits notwendig. Wenn wir also im Durchschnitt  $H(X)$  Bits



*Nach dem Gesetz der großen Zahlen geht für immer größere Werte von  $N$  ...*

$$N \rightarrow \infty$$

*... die Wahrscheinlichkeit gegen 1, eine Sequenz aus der Menge  $T_\delta(N)$  zu erhalten.*



*Menge der typischen Sequenzen*

**Abb. 5.13:** Die Menge  $T_\delta(N)$  enthält alle Sequenzen, die von der Datenquelle „typischerweise“ emittiert werden. Die präzise Definition dieser Menge ist das Schlüssellement im Beweis von Shannons berühmtem Quellencodierungstheorem.

Aus Satz 5.3 können wir eine Strategie für den Umgang mit verschlüsselten Daten ableiten. Die Bitfolge einer perfekt verschlüsselten Nachricht, wie sie beispielsweise durch die Anwendung eines One-Time-Pads entsteht, hat die Struktur einer echten Zufallszahl. In ihr kommen die Symbole 0 und 1 gleichhäufig vor, und es bestehen keinerlei Kontextabhängigkeiten zwischen den Ziffern. Das heißt, dass wir eine perfekt verschlüsselte Nachricht als eine typische Nachricht einer Bernoulli-Quelle  $X$  mit der Entropie 1 ansehen dürfen (vgl. Abbildung 5.6). Aus Satz 5.3 folgt dann für jede Codierung die Beziehung  $L \geq 1$ . Das bedeutet, dass für die Darstellung einer perfekt verschlüsselten Nachricht der Länge  $N$  mindestens  $N$  Bits benötigt werden, wenn wir für  $N$  sehr große Zahlen wählen. Mit anderen Worten: Perfekt verschlüsselte Nachrichten sind unkomprimierbar, und deshalb muss die Kompression einer Nachricht stets vor der Verschlüsselung erfolgen.



für die Darstellung eines Quellsymbols verwenden, so sind in ausreichender Anzahl Codewörter vorhanden, um alle typischen Nachrichten eindeutig zu codieren.

Wird die Schranke auch nur minimal unterschritten, wird ein Symbol also mit durchschnittlich  $H(X) - \varepsilon$  Bits dargestellt, so stehen für die Codierung der  $2^{N(H(X))}$  typischen Sequenzen  $2^{N(H(X)-\varepsilon)}$  verschiedene Codewörter zur Verfügung. Wählen wir  $\varepsilon$  sehr klein, so scheinen immer noch genug Codewörter für die Sequenzen aus  $T_\delta(N)$  vorhanden zu sein, doch dieser Anschein trügt. Deutlich wird dies, wenn wir beobachten, wie sich das Verhältnis zwischen der Anzahl der Codewörter zu der Anzahl der typischen Nachrichten für wachsende Werte von  $N$  entwickelt:

$$\frac{2^{N(H(X)-\varepsilon)}}{2^{N(H(X))}} = \frac{2^{N(H(X))} \cdot 2^{-N\varepsilon}}{2^{N(H(X))}} = \frac{1}{2^{N\varepsilon}}$$

Egal, wie klein wir  $\varepsilon$  auch wählen: Der Quotient strebt gegen 0, und das bedeutet, dass wir einen immer kleiner werdenden Anteil der typischen Sequenzen mit Codewörtern belegen können. Damit wird das Auftreten eines Fehlers für wachsende Werte von  $N$  immer wahrscheinlicher und für  $N \rightarrow \infty$  schließlich zur Gewissheit. Wir fassen zusammen:



### Satz 5.3

Es sei  $X$  eine *gedächtnislose* Datenquelle und  $\varepsilon > 0$ . Dann gilt:

- Für jedes Kompressionsverfahren mit  $L \leq H(X) - \varepsilon$  konvergiert die Fehlerwahrscheinlichkeit gegen 1.

## Erweiterung auf gedächtnisbehaftete Quellen

Als Nächstes wollen wir erarbeiten, wie die Ergebnisse von Satz 5.3 auf gedächtnisbehaftete Quellen übertragen werden können. In Abschnitt 4.2.2 haben wir gefordert, dass alle gedächtnisbehafteten Quellen, die hier betrachtet werden, ergodisch sind. Hieraus folgt unter anderem, dass alle typischen Nachrichten die gleichen statistischen Eigenschaften aufweisen. Über diese Eigenschaften wollen wir nun Genaues herausfinden.

Wie bisher, bezeichnen wir mit  $P_i$  die Wahrscheinlichkeit, dass sich die Datenquelle im Zustand  $i$  befindet, und mit  $p_{ij}$  die Wahrscheinlichkeit,

dass ein Wechsel von Zustand  $i$  in den Zustand  $j$  vollzogen wird. Emittiert die Datenquelle eine Nachricht der Länge  $N$ , so können wir über die Formel

$$P_i p_{ij} N$$

abschätzen, wie oft die Kante von Zustand  $i$  nach  $j$  ungefähr durchlaufen wurde. Je größer wir den Wert von  $N$  wählen, desto genauer nähert diese Formel die tatsächliche Anzahl an. Das bedeutet, dass eine typische Sequenz mit der Wahrscheinlichkeit

$$\bar{p} \approx \prod_{i,j} p_{ij}^{P_i p_{ij} N} \quad (5.12)$$

emittiert wird. Diese Formel ist das Pendant zu Formel (5.7) für gedächtnisbehaftete Quellen.

Genau wie bei den gedächtnislosen Quellen dürfen wir auch hier nicht einfach darüber hinweggehen, dass  $\bar{p}$  nur die ungefähre Wahrscheinlichkeit ist, mit der die typischen Sequenzen auftreten. Erneut lösen wir das Problem, indem wir eine Nachricht als typisch erachten, wenn sie für eine vorgegebene Konstante  $\delta$  in der nachstehenden Menge liegt:

$$T_\delta(N) := \left\{ \mathbf{u} \mid |\mathbf{u}| = N, \prod_{i,j} p_{ij}^{(P_i p_{ij} - \delta)N} < p(\mathbf{u}) < \prod_{i,j} p_{ij}^{(P_i p_{ij} + \delta)N} \right\} \quad (5.13)$$

Die nächsten Argumentationsschritte klingen vertraut. Egal, wie klein wir die Konstante  $\delta$  auch wählen: Nach dem Gesetz der großen Zahlen strebt die Wahrscheinlichkeit, dass die Datenquelle eine Sequenz aus der Menge  $T_\delta(N)$  emittiert, für  $N \rightarrow \infty$  gegen 1. Das bedeutet, dass für jedes  $\varepsilon > 0$  ein  $N_0$  existiert, sodass die Wahrscheinlichkeit, eine Sequenz innerhalb der Menge  $T_\delta(N)$  zu beobachten, für alle  $N > N_0$  größer als  $1 - \varepsilon$  ist:

$$1 > \sum_{\mathbf{u} \in T_\delta(N)} p(\mathbf{u}) > 1 - \varepsilon \quad (5.14)$$

Wie viele Elemente sind in der Menge  $T_\delta(N)$  enthalten? Genau wie bei den gedächtnislosen Quellen können wir diese Frage beantworten, indem wir (5.13) und (5.14) zu der folgenden Abschätzung miteinander kombinieren:

$$\frac{1 - \varepsilon}{\prod_{i,j} p_{ij}^{(P_i p_{ij} + \delta)N}} < |T_\delta(N)| < \frac{1}{\prod_{i,j} p_{ij}^{(P_i p_{ij} - \delta)N}}$$

Nebenrechnung zu (5.16)

$$\begin{aligned} N \cdot H(X) &= N \sum_{i,j} P_i p_{ij} \log_2 \frac{1}{p_{ij}} \\ &= \sum_{i,j} \log_2 p_{ij}^{-P_i p_{ij} N} \end{aligned}$$

$$\begin{aligned} |T_\delta(N)| &\approx 2^{N \cdot H(X)} \\ &= \prod_{i,j} 2^{\log_2 p_{ij}^{-P_i p_{ij} N}} \\ &= \prod_{i,j} p_{ij}^{-P_i p_{ij} N} \\ &= \frac{1}{\bar{p}} \end{aligned}$$

**Abb. 5.14:** Für  $N \rightarrow \infty$  nähert sich die Anzahl der Elemente von  $T_\delta(N)$  dem Kehrwert von  $\bar{p}$  immer weiter an.

Hieraus folgt

$$\begin{aligned} \frac{\log_2(1-\varepsilon)}{N} + \sum_{i,j} (P_i p_{ij} + \delta) \log_2 \frac{1}{p_{ij}} &< \frac{\log_2 |T_\delta(N)|}{N} \\ &< \sum_{i,j} (P_i p_{ij} - \delta) \log_2 \frac{1}{p_{ij}}, \end{aligned}$$

woraus wir analog zu (5.10) schließen können:

$$\lim_{\substack{\delta \rightarrow 0 \\ N \rightarrow \infty}} \frac{\log_2 |T_\delta(N)|}{N} = \sum_{i,j} P_i p_{ij} \log_2 \frac{1}{p_{ij}} = H(X)$$

Wählen wir also für  $\delta$  immer kleinere Werte und für  $N$  immer größere, so machen wir einen immer kleineren Fehler, wenn wir die Anzahl der Elemente in der Menge  $T_\delta(N)$  folgendermaßen abschätzen:

$$|T_\delta(N)| \approx 2^{NH(X)} \quad (5.15)$$

Damit haben wir es geschafft, die Beziehung (5.11) von gedächtnislosen auf beliebige Datenquellen zu verallgemeinern. Die weitere Argumentation können wir eins zu eins aus dem vorhergehenden Abschnitt übernehmen und das erzielte Ergebnis dann folgendermaßen zusammenfassen:

### Satz 5.4

Es sei  $X$  eine Datenquelle und  $\varepsilon > 0$ . Dann gilt:

- Für jedes Kompressionsverfahren mit  $L \leq H(X) - \varepsilon$  konvergiert die Fehlerwahrscheinlichkeit gegen 1.

Bevor wir fortfahren, halten wir fest, dass sich die Beziehung 5.15 noch kompakter niederschreiben lässt. Wie die Nebenrechnung in Abbildung 5.14 zeigt, besteht zwischen der Anzahl der Elemente von  $T_\delta(N)$  und der Wahrscheinlichkeit  $\bar{p}$ , mit der eine einzelne typische Sequenz der Länge  $N$  emittiert wird, der folgende Zusammenhang:

$$|T_\delta(N)| \approx \frac{1}{\bar{p}} \quad (5.16)$$

## 5.4 Blockweise Codierung

Satz 5.2 hat gezeigt, dass wir uns der Schranke

$$\sum_i p_i \log_2 \frac{1}{p_i}$$

mit präfixfreien Codierungen so weit annähern können, dass die mittlere Codewortlänge um höchstens 1 größer ist. Dies führt unmittelbar zu der folgenden Frage: Lässt sich das Ergebnis verbessern, wenn wir den zeichenweise arbeitenden Codierungen den Rücken kehren? Die Antwort lautet Ja. Die Redundanz lässt sich auf einfache Weise verringern, indem die Datenquelle  $X$  gedanklich in eine Datenquelle  $X^n$  umgewandelt wird, die ihre Symbole nicht mehr zeichenweise emittiert, sondern in Blöcken, die aus jeweils  $N$  Quellensymbolen bestehen.

Abbildung 5.15 zeigt den Übergang von  $X$  zu  $X^2$  für die gedächtnislose Datenquelle, die wir auch weiter oben als Beispiel herangezogen haben. Der Quelle  $X$  liegt das Quellenalphabet

$$\Sigma = \{a, b, c, d, e\}$$

zugrunde, und das bedeutet, dass  $X^2$  in jedem Schritt ein Symbol aus der folgenden Menge emittiert:

$$\begin{aligned} & \{ aa, ab, ac, ad, ae, \\ & ba, bb, bc, bd, be, \\ & ca, cb, cc, cd, ce, \\ & da, db, dc, dd, de, \\ & ea, eb, ec, ed, ee \} \end{aligned}$$

Mit den Sätzen 5.1 und 5.2 sind wir imstande, ein wichtiges Ergebnis über die Quelle  $X^n$  herzuleiten, und zwar unabhängig davon, ob  $X$  gedächtnisfrei ist oder nicht. Aus den beiden Sätzen folgt, dass für die Quelle  $X^n$  eine präfixfreie Codierung  $c$  existiert, mit

$$\sum_{|\mathbf{u}|=N} p(\mathbf{u}) \log_2 \frac{1}{p(\mathbf{u})} \leq L(X^N, c) < \sum_{|\mathbf{u}|=N} p(\mathbf{u}) \log_2 \frac{1}{p(\mathbf{u})} + 1 \quad (5.17)$$

Da  $c$  die Quellensymbole nicht mehr einzeln, sondern blockweise codiert, entspricht die mittlere Codewortlänge  $L(X^N, c)$  der Anzahl der Symbole des Codealphabets, die für die Codierung von  $N$  Symbolen des Quellenalphabets im Mittel aufgewendet werden. Das bedeutet, dass jedes Quellensymbol mit durchschnittlich

$$L = \frac{L(X^N, c)}{N}$$

Symbolen des Codealphabets dargestellt wird. Damit können wir (5.17) folgendermaßen umformen:

$$\frac{1}{N} \sum_{|\mathbf{u}|=N} p(\mathbf{u}) \log_2 \frac{1}{p(\mathbf{u})} \leq L < \frac{1}{N} \sum_{|\mathbf{u}|=N} p(\mathbf{u}) \log_2 \frac{1}{p(\mathbf{u})} + \frac{1}{N}$$

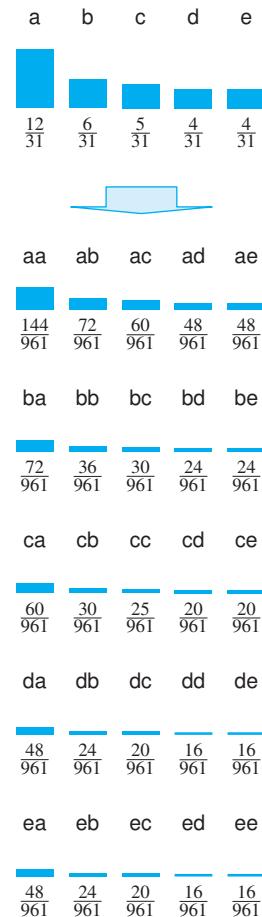


Abb. 5.15: Übergang von  $X$  zu  $X^2$

Nebenrechnung zu Satz 5.6

$$\begin{aligned}
 & \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{|\mathbf{u}|=N} p(\mathbf{u}) \log_2 \frac{1}{p(\mathbf{u})} \\
 &= \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{\mathbf{u} \in T_\delta(N)} p(\mathbf{u}) \log_2 \frac{1}{p(\mathbf{u})} \\
 &= \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{\mathbf{u} \in T_\delta(N)} \bar{p} \log_2 \frac{1}{\bar{p}} \\
 &= \lim_{N \rightarrow \infty} \frac{1}{N} |T_\delta(N)| \bar{p} \log_2 \frac{1}{\bar{p}} \\
 &\stackrel{(5.16)}{=} \lim_{N \rightarrow \infty} \frac{1}{N} \log_2 |T_\delta(N)| \\
 &\stackrel{(5.15)}{=} \lim_{N \rightarrow \infty} \frac{1}{N} N H(X) \\
 &= H(X)
 \end{aligned}$$

**Abb. 5.16:** Bei der Berechnung des Grenzwerts nutzen wir aus, dass für große Werte von  $N$  der Fehler beliebig klein wird, wenn wir uns bei der Auswahl der Nachrichten  $\mathbf{u}$  auf die Menge  $T_\delta(N)$  der *typischen Nachrichten* beschränken.

Für wachsende  $N$  wird der Korridor, der durch die linke und die rechte Seite dieser Ungleichung gebildet wird, immer enger, sodass  $L$  ab einer gewissen Blockgröße  $N$  den Wert

$$\sum_{|\mathbf{u}|=N} p(\mathbf{u}) \log_2 \frac{1}{p(\mathbf{u})} + \varepsilon$$

für jedes beliebige  $\varepsilon > 0$  unterschreitet. In Satzform lautet dieses Ergebnis so:

### Satz 5.5

Für jede Datenquelle  $X$  und jedes  $\varepsilon > 0$  gibt es eine blockweise arbeitende Codierung  $c$  mit der Eigenschaft

$$\frac{1}{N} \sum_{|\mathbf{u}|=N} p(\mathbf{u}) \log_2 \frac{1}{p(\mathbf{u})} \leq L < \frac{1}{N} \sum_{|\mathbf{u}|=N} p(\mathbf{u}) \log_2 \frac{1}{p(\mathbf{u})} + \varepsilon$$

Tatsächlich ist es von diesem Satz nur noch ein kleiner Schritt zu jenem Ergebnis, das wir suchen. Der Wert auf der linken Seite der Abschätzung ist für  $N \rightarrow \infty$  nämlich nichts anderes als die Entropie der Datenquelle  $X$ , wie die Nebenrechnung in Abbildung 5.16 beweist. Damit können wir aus Satz 5.5 den folgenden Schluss ziehen:

### Satz 5.6

Für jede Datenquelle  $X$  und jedes  $\varepsilon > 0$  gibt es eine blockweise arbeitende Codierung  $c$  mit der Eigenschaft

$$L < H(X) + \varepsilon$$

Kombinieren wir diese Aussage mit dem Inhalt von Satz 5.4, so erhalten wir das folgende Ergebnis, mit dem wir die Ziellinie schon fast überquert haben:

### 9. THE FUNDAMENTAL THEOREM FOR A NOISELESS CHANNEL

We will now justify our interpretation of  $H$  as the rate of generating information by proving that  $H$  determines the channel capacity required with most efficient coding.

 **Theorem 9:** Let a source have entropy  $H$  (bits per symbol) and a channel have a capacity  $C$  (bits per second). Then it is possible to encode the output of the source in such a way as to transmit at the average rate  $\frac{C}{H} - \epsilon$  symbols per second over the channel where  $\epsilon$  is arbitrarily small. It is not possible to transmit at an average rate greater than  $\frac{C}{H}$ .

**Abb. 5.17:** Das Quellencodierungstheorem ist das neunte Theorem in Shannons historischer Arbeit aus dem Jahr 1948.



### Satz 5.7

Es sei  $X$  eine Datenquelle und  $\epsilon > 0$ . Dann gilt:

- Es gibt ein Kompressionsverfahren mit  $L \leq H(X) + \epsilon$  und einer Fehlerwahrscheinlichkeit, die gegen 0 konvergiert.
- Für jedes Kompressionsverfahren mit  $L \leq H(X) - \epsilon$  konvergiert die Fehlerwahrscheinlichkeit gegen 1.

Als Nächstes wollen wir dieses Ergebnis mit dem Wissen vereinen, das wir uns in Abschnitt 3.5.1 angeeignet haben. Dort haben wir herausgearbeitet, dass sich jedem Übertragungskanal eine Kapazität  $C$  zuordnen lässt, die angibt, wie viele Bits pro Sekunde maximal übertragen werden können. Ist  $X$  eine Datenquelle, so garantiert uns Satz 5.7, dass wir die Nachrichten so komprimieren können, dass in etwa  $H(X)$  Bits pro Quellsymbol verwendet werden. Das bedeutet, dass wir durch eine geeignete Kompression  $\frac{C}{H}$  Symbole pro Sekunde übertragen können, aber keinesfalls mehr. Übersetzen wir dieses Ergebnis in Satzform, so erreichen wir das berühmte Quellencodierungstheorem von Claude Shannon (Abbildung 5.17):



### Satz 5.8 (Quellencodierungstheorem, Shannon, 1948)

Es sei  $X$  eine Datenquelle und  $\epsilon > 0$ . Dann gilt:

- Es ist möglich, Nachrichten mit einer Transferrate von  $\frac{C}{H} - \epsilon$  Symbolen pro Sekunde zu übertragen.
- Es ist nicht möglich, Nachrichten mit einer Transferrate zu übertragen, die  $\frac{C}{H}$  übersteigt.

## 5.5 Übungsaufgaben

### Aufgabe 5.1



**Webcode  
5045**

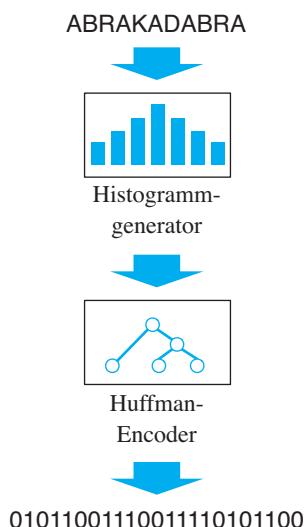
Auf Seite 316 haben wir begründet, warum jedes verlustfreie Kompressionsverfahren  $K$  die folgende Eigenschaft besitzen muss:

„Gibt es Nachrichten, die durch  $K$  verkürzt werden, so gibt es auch welche, die durch  $K$  verlängert werden.“

- a) Wäre die folgende Formulierung ebenfalls richtig?

„Jedes verlustfreie Kompressionsverfahren  $K$ , das  $n$  Nachrichten verkürzt, muss mindestens  $n$  Nachrichten verlängern.“

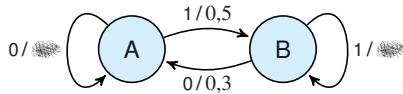
- b) In dieser Teilaufgabe betrachten wir ein Verfahren, das Bytefolgen codiert, d. h., das Quellenalphabet besteht aus 256 verschiedenen Symbolen. Der Encoder arbeitet in zwei Schritten. Zunächst bestimmt er die Häufigkeiten, mit denen die Quellsymbole in der zu übertragenden Nachricht vorkommen. Danach berechnet er mit dem Huffman-Algorithmus einen optimalen präfixfreien Code und codiert damit die Nachricht.



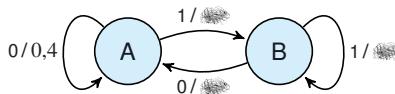
Wir wissen, dass Huffman-Codes das schlechteste Ergebnis liefern, wenn alle Quellsymbole mit der gleichen Wahrscheinlichkeit vorkommen. In diesem Fall würde jedes der 256 Quellsymbole mit 8 Bit codiert werden. Im schlechtesten Fall würde die Länge der Nachricht somit unverändert bleiben, und dies steht im Widerspruch zu der oben getätigten Aussage. Lösen Sie den Widerspruch auf!

Gegeben seien die folgenden beiden ergodischen Markov-Quellen der Ordnung 1:

**Aufgabe 5.2**

**Webcode**  
**5194**


$$P_A = \text{[redacted]} \quad P_B = \text{[redacted]}$$



$$P_A = \frac{1}{4} \quad P_B = \frac{3}{4}$$

Rekonstruieren Sie die Diagramme, in denen leider einige Werte verloren gegangen sind.  
Hinweis:  $P_A$  und  $P_B$  sind die Wahrscheinlichkeiten, dass sich die Markov-Quellen in den Zuständen A bzw. B aufhalten.

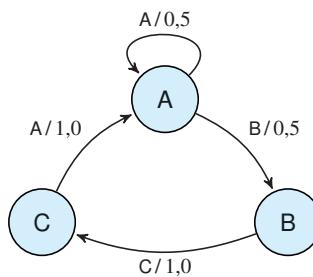
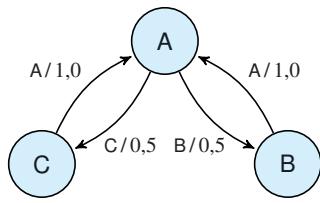
Es seien die folgenden beiden Markov-Quellen gegeben:

**Aufgabe 5.3**

**Webcode**  
**5243**

Quelle 1

Quelle 2



- a) Berechnen Sie, mit welchen Wahrscheinlichkeiten sich die beiden Markov-Quellen in den Zuständen A, B und C aufhalten.

$$P_A = \text{[redacted]}$$

$$P_A = \text{[redacted]}$$

$$P_B = \text{[redacted]}$$

$$P_B = \text{[redacted]}$$

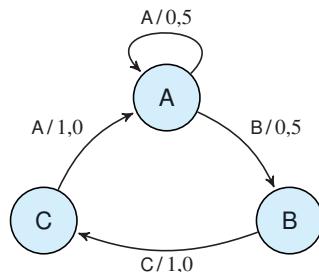
$$P_C = \text{[redacted]}$$

$$P_C = \text{[redacted]}$$

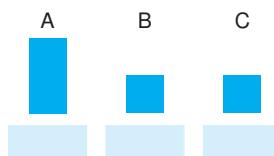
- b) Wie lauten die beiden linearen Gleichungssysteme, mit denen sich die Ergebnisse aus der vorherigen Teilaufgabe formal berechnen lassen?

**Aufgabe 5.4****Webcode****5355**

Auf Seite 321 haben wir anhand der folgenden Markov-Quelle motiviert, warum der Entropiebegriff nicht eins zu eins von gedächtnislosen auf gedächtnisbehaftete Datenquellen übertragen werden darf:



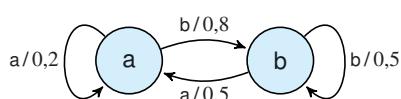
- a) Rekapitulieren Sie, wie oft die Symbole A, B und C im Durchschnitt emittiert werden. Tragen Sie die Werte in das nachstehende Histogramm ein:



- b) Berechnen Sie die Entropie der Datenquelle.
- c) Berechnen Sie die Entropie einer gedächtnislosen Datenquelle mit  $\Sigma = \{A, B, C\}$ , die ihre Symbole mit den in Teilaufgabe a) ermittelten Wahrscheinlichkeiten emittiert. Um wie viel unterscheidet sich die Entropie von dem in Teilaufgabe b) berechneten Wert?
- d) Begründen Sie anschaulich, weshalb die Entropie einer gedächtnisbehafteten Quelle niemals größer sein kann als die Entropie einer gedächtnislosen Quelle, die ihre Symbole mit den gleichen Wahrscheinlichkeiten emittiert.

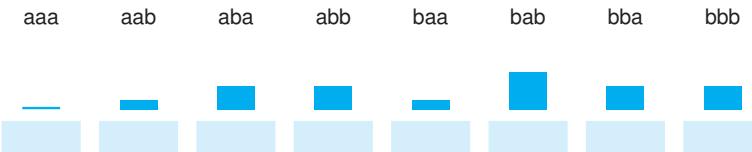
**Aufgabe 5.5****Webcode****5522**

Gegeben sei die folgende Markov-Quelle X der Ordnung 1:



- a) Berechnen Sie für jeden Zustand die individuelle Entropie.

- b) Berechnen Sie für jeden Zustand die Aufenthaltswahrscheinlichkeit.
- c) Berechnen Sie aus den erzielten Teilergebnissen die Entropie der Datenquelle.
- d) Vervollständigen Sie das nachstehende Histogramm für die Datenquelle  $X^3$ :



- e) Konstruieren Sie mit dem Huffman-Algorithmus einen Code für die Datenquelle  $X^3$ .
- f) Wie groß ist die mittlere Codewortlänge und die Redundanz der ermittelten Codierung?

Bestimmen Sie den Grenzwert

$$\lim_{x \rightarrow 0} \left( x \cdot \log_2 \frac{1}{x} \right)$$

und leiten Sie daraus ab, wie ein Zeichen mit sehr geringer Auftrittswahrscheinlichkeit die Entropie einer Datenquelle beeinflusst.

**Aufgabe 5.6**



**Webcode**

**5803**

In Abschnitt 5.4 haben wir besprochen, wie sich aus einer Datenquelle  $X$  eine neue Datenquelle  $X^2$  ableiten lässt. In dieser Aufgabe nehmen wir an,  $X$  sei eine gedächtnislose Quelle. Das bedeutet, dass die Quelle  $X^2$  ein Symbolpaar  $\sigma_i \sigma_j$  mit einer Wahrscheinlichkeit emittiert, die dem Produkt der Einzelwahrscheinlichkeiten der Blocksymbole entspricht:

$$p(\sigma_i \sigma_j) = p(\sigma_i) \cdot p(\sigma_j) = p_i p_j$$

In dieser Aufgabe wollen wir der Frage nachgehen, wie groß die Entropie von  $X^2$  ist. Legen wir die anschauliche Sichtweise zugrunde, dass sie die Information beziffert, die  $X$  pro Symbol im Durchschnitt emittiert, so lässt sich die Antwort bereits erahnen. Da  $X^2$  in jedem Schritt doppelt so viele Symbole emittiert wie  $X$ , aber ansonsten mit  $X$  identisch ist, muss die Entropie  $H(X^2)$  genau doppelt so groß sein wie  $H(X)$ :

$$H(X^2) = 2 \cdot H(X)$$

Beweisen Sie diese Vermutung.

**Aufgabe 5.7**



**Webcode**

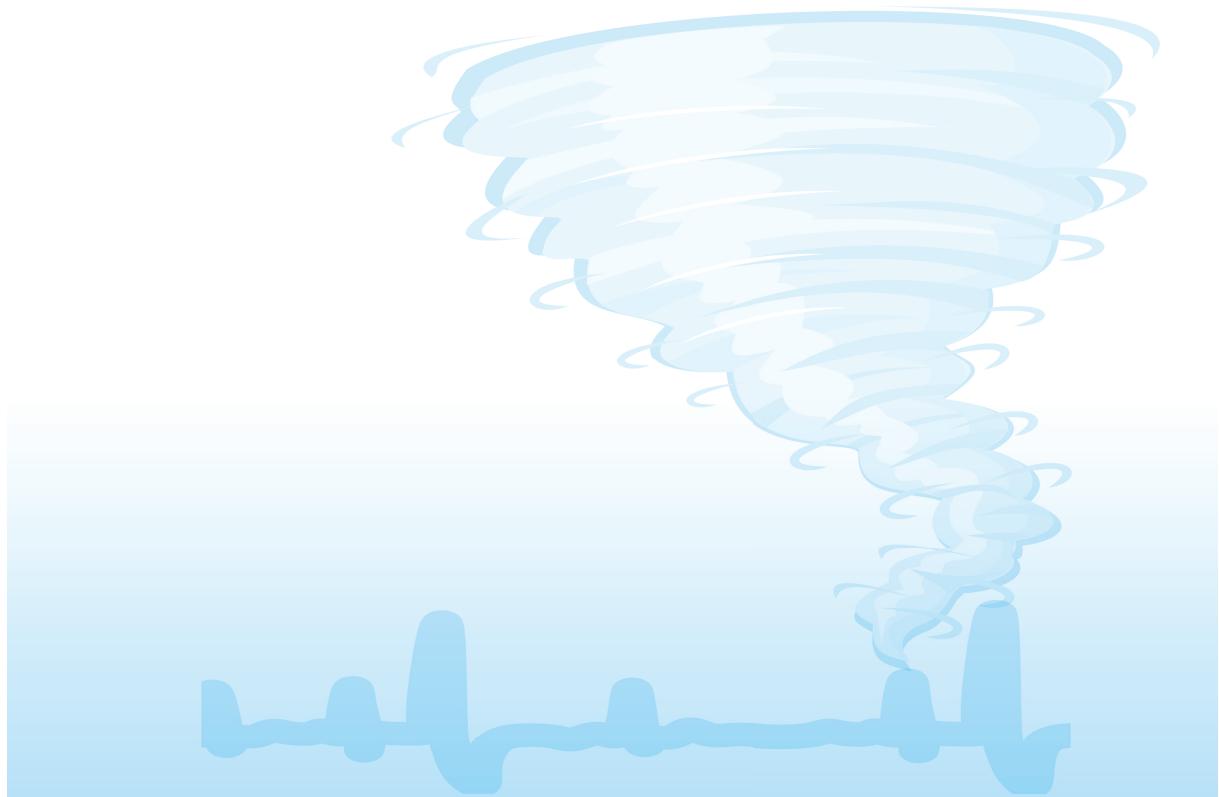
**5883**

# 6 Kanalcodierung

---

In diesem Kapitel werden Sie ...

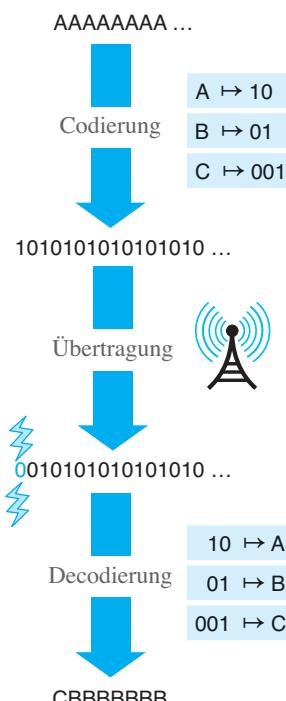
- verschiedene Prüfziffercodes aus der Praxis kennenlernen,
- die Begriffe der Fehlererkennung und der Fehlerkorrektur formal definieren,
- eine Übersicht über wichtige lineare Kanalcodes erhalten,
- das Prinzip von Faltungscodierern verstehen und
- den Viterbi-Algorithmus anwenden.



## 6.1 Motivation

Die bisher angestellten Überlegungen waren allesamt von der Idee geprägt, Information so schnell wie möglich, d. h. am Limit der Kanalkapazität, zu übertragen. Wie dieses Ziel erreicht werden kann, haben wir im letzten Kapitel ausgiebig besprochen. Sie haben dort zahlreiche Codierungen kennengelernt, die eine Nachricht vor der Übertragung in eine kompaktere Darstellung überführen.

Was wir in unserer bisherigen Betrachtung völlig außer Acht gelassen haben, ist die Störanfälligkeit der Übertragungsstrecke; bisher sind wir stillschweigend davon ausgegangen, dass eine Bitsequenz, egal wie diese gestaltet ist, immer fehlerfrei den Empfänger erreicht. Abbildung 6.1 macht deutlich, dass der unbedachte Einsatz einer Codierung zu einer massiven Erhöhung der Fehleranfälligkeit führen kann. In dem dort gezeigten Übertragungsszenario führt bereits ein einziges verfälschtes Bit dazu, dass sämtliche Symbole einer Nachricht falsch decodiert werden.



**Abb. 6.1:** Ein einziges defektes Bit reicht aus, damit die Nachricht vollständig falsch decodiert wird.

Genau an dieser Stelle kommt die *Kanalcodierung* ins Spiel. Ein Kanalcodierer hat die Aufgabe, die Ausgabe eines Quellencodierers so aufzubereiten, dass der tatsächlich gesendete Symbolstrom robuster gegenüber Übertragungsfehlern wird. Die Codierungen, die hierfür in Frage kommen, weisen dabei ganz unterschiedliche Eigenschaften auf. Manche können aufgetretene Übertragungsfehler lediglich erkennen, während andere in der Lage sind, diese sogar in einem gewissen Umfang zu korrigieren.

Wir beginnen unsere Reise durch das Gebiet der Kanalcodierung in Abschnitt 6.2 mit einem Blick in den Alltag. Dort werden wir mehrere Prüfziffercodes kennenlernen, denen wir, bewusst oder unbewusst, täglich begegnen. Die Codierungen werden uns einen ersten Einblick in die Ziele und die Methoden der Kanalcodierung gewähren.

Danach werden wir die Kanalcodierung von ihrer theoretischen Seite beleuchten. Nachdem wir in Abschnitt 6.3 geklärt haben, was unter *fehlererkennenden* und *fehlerkorrigierenden* Codes genau zu verstehen ist, beleuchten wir in den Abschnitten 6.4 und 6.5 viele der heute gängigen Codierungen aus diesem Bereich.

Abschließend setzen wir uns in Kapitel 7 mit den Grenzen der Kanalcodierung auseinander. Unter anderem werden wir dort Shannons berühmtes Kanalcodierungstheorem (*noisy channel theorem*) besprechen, das unzweifelhaft zu den faszinierendsten Erkenntnissen der Informations- und Codierungstheorie gehört.

## 6.2 Prüfziffercodes

Zu Beginn dieses Abschnitts werfen wir erneut einen Blick auf den geraden Paritätscode, mit dessen Konstruktionsschema wir bereits gut vertraut sind. Um einen Übertragungsfehler zu erkennen, wird eine Bitsequenz

$$v_0 v_1 \dots v_{n-1}$$

genau dann als ein gültiges Codewort betrachtet, wenn sie die Eigenschaft

$$v_0 + v_1 + \dots + v_{n-1} \equiv 0 \pmod{2} \quad (6.1)$$

erfüllt. Die Kongruenzbeziehung (6.1) ist die definierende Eigenschaft des Paritätscodes; sie wird von jedem Codewort erfüllt, und umgekehrt gilt, dass jede Bitsequenz mit dieser Eigenschaft ein Codewort ist.

Tatsächlich ist der Paritätscode ein spezieller Repräsentant einer größeren Klasse von Codes, die wir in diesem Buch als *Prüfziffercodes* bezeichnen. Um diese Codes formal zu definieren, müssen wir lediglich die Kongruenz (6.1) in der folgenden Weise verallgemeinern:



### Definition 6.1 (Prüfziffercode)

Ein Code  $C \subseteq \mathbb{N}^n$  heißt *Prüfziffercode*, wenn ein Modul  $p$  und Gewichte  $g_0, \dots, g_{n-1}$  mit der folgenden Eigenschaft existieren:

$$v_0 v_1 \dots v_{n-1} \in C \Leftrightarrow \sum_i g_i v_i \equiv 0 \pmod{p}$$

Setzen wir  $p = 2$  und wählen für alle Gewichte die Zahl 1, so erhalten wir den  $n$ -stelligen Paritätscode als Spezialfall dieser Definition.

Prüfziffercodes kommen in der Praxis häufig vor und werden vor allem dann eingesetzt, wenn Zahlenfolgen manuell über eine Tastatur eingegeben werden. In diesen Fällen ist die Wahrscheinlichkeit hoch, dass entweder eine einzelne Ziffer falsch eingetippt wird oder zwei Ziffern durch einen Zahlendreher in die falsche Reihenfolge gebracht werden.

Als Beispiel zeigt Abbildung 6.2 einen Prüfziffercode über dem Alphabet  $\{0, 1, 2, 3\}$ , der über die Kongruenzbeziehung

$$3 \cdot v_0 + 1 \cdot v_1 + 3 \cdot v_2 \equiv 0 \pmod{4} \quad (6.2)$$

gegeben ist. Der Code erfüllt die Eigenschaft, dass jeder einzelne Ziffernfehler zu einer Symbolsequenz führt, die (6.2) widerspricht. Wir sagen, der Code ist *1-fehlererkennend*. Zifferndreher werden hingegen nur

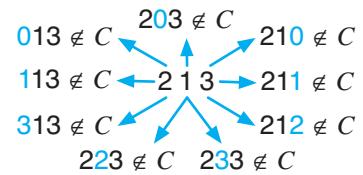
$$3 \cdot v_0 + 1 \cdot v_1 + 3 \cdot v_2 \equiv 0 \pmod{4}$$



$u$	$c(u)$
00	000
01	011
02	022
03	033
10	103
11	110
12	121
13	132
20	202
21	213
22	220
23	231
30	301
31	312
32	323
33	330

Die Verfälschung einer einzelnen Ziffer führt immer zu einer Symbolsequenz, die kein Codewort ist. Dies gilt nicht für jede Vertauschung zweier benachbarter Ziffern.

#### Einzelfehler



#### Zifferndreher



**Abb. 6.2:** Dreistelliger Prüfziffercode mit dem Modul 4 und den Gewichten 3, 1, 3

$$3 \cdot v_0 + 2 \cdot v_1 + 3 \cdot v_2 \equiv 0 \pmod{4}$$



$u$	$c(u)$
00	000
01	012
02	020
03	032
10	103
11	111
12	123
13	131
20	202
21	210
22	222
23	230
30	301
31	313
32	321
33	333

Die Vertauschung zweier benachbarter Ziffern führt immer zu einer Symbolsequenz, die kein Codewort ist. Dies gilt nicht für jede Verfälschung einer einzelnen Ziffer.

#### Zifferndreher



#### Einzelfehler

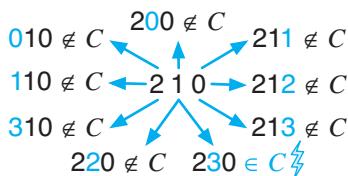


Abb. 6.3: Dreistelliger Prüfziffercode mit dem Modul 4 und den Gewichten 3, 2, 3

manchmal erkannt. Werden innerhalb des Codeworts 213 beispielsweise die erste und die zweite Ziffer vertauscht, so entsteht mit 123 eine Symbolsequenz, die kein Codewort ist; der Fehler wird also erkannt. Werden dagegen die zweite und die dritte Ziffer vertauscht, so entsteht die Sequenz 231, die ebenfalls ein Codewort ist. Der Empfänger hat dann keine Möglichkeit mehr, den Fehler zu erkennen.

Ändern wir die Vorschrift (6.1) geringfügig ab, in

$$3 \cdot v_0 + 2 \cdot v_1 + 3 \cdot v_2 \equiv 0 \pmod{4}, \quad (6.3)$$

so entsteht der Code aus Abbildung 6.3. Er hat die Eigenschaft, dass die Vertauschung zweier benachbarter Ziffern in einem Codewort immer zu einer Symbolsequenz führt, die kein Codewort ist. Zifferndreher werden damit sicher erkannt. Dafür hat dieser Code die Eigenschaft verloren, 1-fehlererkennend zu sein. Beispielsweise ist sowohl die Symbolsequenz 210 als auch die Symbolsequenz 230 ein Codewort, beide unterscheiden sich aber nur in einer Ziffer.

Wir wollen der Frage auf den Grund gehen, weshalb der erste Beispielcode alle Einzelfehler, aber nicht jeden Zahlendreher entdeckt, und der zweite Beispielcode genau die gegenteiligen Eigenschaften aufweist. Tatsächlich können wir mit dem Wissen, das wir uns in Abschnitt 2.2 über die Modulrechnung angeeignet haben, eine präzise Antwort auf diese Frage geben.

### 6.2.1 Erkennung von Einzelfehlern

Wir beginnen mit der Frage, welche Eigenschaft die Definitionsvorschrift

$$g_0v_0 + g_1v_1 + \dots + g_{n-1}v_{n-1} \equiv 0 \pmod{p}$$

aufweisen muss, damit ein 1-fehlererkennender Code entsteht. Ein solcher Code muss alle Fehler erkennen, die durch die Verfälschung einer einzelnen Ziffer hervorgerufen werden. Hierzu sei

$$v_0v_1 \dots v_{n-1}$$

das gesendete Codewort und

$$w_0w_1 \dots w_{n-1}$$

die empfangene Symbolsequenz. Wir nehmen an, dass das Symbol  $v_k$  auf der Übertragungsstrecke verfälscht wurde. Es gilt dann:

$$w_i \neq v_i \quad \text{für } i = k$$

$$w_i = v_i \quad \text{für } i \neq k$$

Wie die seitlich abgedruckte Nebenrechnung zeigt, folgt hieraus die Beziehung

$$\sum_i g_i w_i \equiv 0 \pmod{p} \Leftrightarrow g_k(w_k - v_k) \equiv 0 \pmod{p} \quad (6.4)$$

Damit haben wir die Frage, ob der Code einen Einzelfehler erkennt, auf die Frage reduziert, ob die Kongruenz

$$g_k(w_k - v_k) \equiv 0 \pmod{p} \quad (6.5)$$

neben der trivialen Lösung  $w_k = v_k$  noch andere Lösungen besitzt. Wir unterscheiden zwei Fälle:

### Nebenrechnung zu (6.4)

$$\begin{aligned} \sum_i g_i w_i &\equiv 0 \pmod{p} \\ \Leftrightarrow g_k w_k + \sum_{i \neq k} g_i w_i &\equiv 0 \pmod{p} \\ \Leftrightarrow g_k w_k + \sum_{i \neq k} g_i v_i &\equiv 0 \pmod{p} \\ \Leftrightarrow g_k w_k - g_k v_k + \sum_i g_i v_i &\equiv 0 \pmod{p} \\ \Leftrightarrow g_k w_k - g_k v_k &\equiv 0 \pmod{p} \\ \Leftrightarrow g_k(w_k - v_k) &\equiv 0 \pmod{p} \end{aligned}$$

#### ■ $g_k$ und $p$ sind teilerfremd

In diesem Fall besitzt  $g_k$  nach Satz 2.1 ein (eindeutig bestimmtes) inverses Element  $g_k^{-1}$ , mit dem wir die linke und die rechte Seite von (6.5) multiplizieren können. Wir erhalten dann

$$\begin{aligned} g_k(w_k - v_k) \equiv 0 \pmod{p} &\Leftrightarrow g_k^{-1} g_k(w_k - v_k) \equiv 0 \pmod{p} \\ &\Leftrightarrow w_k - v_k \equiv 0 \pmod{p} \end{aligned}$$

Somit ist  $w_k = v_k$  die einzige Lösung der Kongruenz und der Code damit 1-fehlererkennend.

#### ■ $g_k$ und $p$ sind nicht teilerfremd

In diesem Fall lassen sich  $g_k$  und  $p$  in der Form  $g_k = qg'_k$  und  $p = qp'$  darstellen mit  $1 \leq g'_k < g_k$  und  $1 \leq p' < p$ . Aus der trivialen Beziehung

$$g'_k \cdot qp' \equiv 0 \pmod{qp'}$$

folgt dann

$$g_k p' \equiv 0 \pmod{p}$$

Das bedeutet, dass nicht nur die Kombination  $w_k = v_k$  die Kongruenz (6.5) löst, sondern auch jede Wertekombination, die

$$w_k - v_k = p'$$

erfüllt. Da  $p'$  der Abschätzung  $1 \leq p' < p$  genügt, wird diese Gleichung durch zwei Werte  $w_k$  und  $v_k$  mit  $w_k \neq v_k$  gelöst. Für diese Werte gilt: Wird an der  $k$ -ten Stelle die Ziffer  $v_k$  gesendet und diese auf der Übertragungsstrecke in  $w_k$  verfälscht, so bleibt der Fehler unentdeckt. Kurzum: Der Code ist nicht 1-fehlererkennend.

Damit haben wir eine wichtige Eigenschaft von Prüfziffercodes aufgedeckt:


**Satz 6.1**

Ein Prüfziffercode, der über die Beziehung

$$\sum_i g_i v_i \equiv 0 \pmod{p}$$

gegeben ist, erkennt genau dann alle Einzelfehler an der Stelle  $k$ , wenn das Gewicht  $g_k$  ein Element der Menge  $\mathbb{Z}_p^*$  ist.

Im Lichte dieses Satzes wird klar, warum der Code aus Abbildung 6.2 alle Einzelfehler erkennen kann, der Code aus Abbildung 6.3 dagegen nicht. Der erste verwendet mit  $g_0 = 3, g_1 = 1, g_2 = 3$  ausschließlich Gewichte aus der Menge

$$\mathbb{Z}_4^* = \{1, 3\}$$

und erkennt deshalb jeden Einzelfehler, unabhängig davon, an welcher Stelle er auftritt.

Der zweite Code benutzt mit  $g_1 = 2$  ein Gewicht, das kein Element von  $\mathbb{Z}_4^*$  ist. Das bedeutet, dass ein Einzelfehler nur noch dann sicher erkannt wird, wenn er an der ersten oder der letzten Stelle auftritt.

Beachten Sie, dass die Beziehung  $g_1 \notin \mathbb{Z}_4^*$  nicht bedeutet, dass gar kein Einzelfehler an der mittleren Bitstelle erkannt werden kann. Die Beispiele in Abbildung 6.3 haben gezeigt, dass die meisten Einzelfehler zu Ziffernfolgen führen, die keine Codewörter sind und damit sehr wohl als fehlerhaft erkannt werden können. Satz 6.1 attestiert lediglich, dass dies nicht für alle Einzelfehler funktioniert. Es lässt sich also immer ein gewisses Codewort finden, das durch den Austausch einer einzelnen Ziffer in ein anderes Codewort übergeht.

## 6.2.2 Erkennung von Vertauschungsfehlern

Als Nächstes wollen wir mit einer ähnlichen Überlegung herausarbeiten, unter welchen Bedingungen ein Vertauschungsfehler erkannt werden kann. Wieder sei

$$v_0 v_1 \dots v_{n-1}$$

das gesendete Codewort und

$$w_0 w_1 \dots w_{n-1}$$

die empfangene Symbolsequenz. Wir nehmen an, dass die empfangene Sequenz durch die Vertauschung von  $v_k$  mit  $v_l$  aus dem gesendeten Codewort hervorgegangen ist mit  $0 \leq k < l < n$ . Es gilt dann:

$$\begin{aligned} w_k &= v_l \\ w_l &= v_k \\ w_i &= v_i \quad \text{für } i \neq k \text{ und } i \neq l \end{aligned}$$

Wie die seitlich abgedruckte Nebenrechnung zeigt, folgt hieraus die Beziehung

$$\sum_i g_i w_i \equiv 0 \pmod{p} \Leftrightarrow (g_k - g_l)(v_k - v_l) \equiv 0 \pmod{p} \quad (6.6)$$

Das bedeutet, dass der Code die Vertauschung zweier Ziffern genau dann erkennen kann, wenn die Kongruenz neben der trivialen Lösung  $v_k = v_l$  keine weiteren Lösungen besitzt. Nach der oben angestellten Überlegung ist klar, dass dies genau dann der Fall ist, wenn die Differenz  $g_k - g_l$  und der Modul  $p$  teilerfremd sind, oder, was das Gleiche bedeutet, wenn  $g_k - g_l$  ein Element der Menge  $\mathbb{Z}_p^*$  ist. Wir halten fest:

### Satz 6.2

Ein Prüfziffercode, der über die Beziehung

$$\sum_i g_i v_i \equiv 0 \pmod{p}$$

gegeben ist, erkennt eine Vertauschung von  $v_k$  mit  $v_l$  genau dann, wenn die Zahl  $g_k - g_l$  ein Element der Menge  $\mathbb{Z}_p^*$  ist.



### Nebenrechnung zu (6.6)

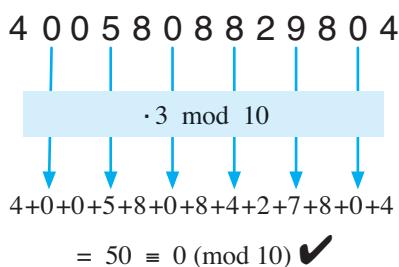
$$\begin{aligned} \sum_i g_i w_i &\equiv 0 \pmod{p} \\ \Leftrightarrow g_k w_k + g_l w_l + \sum_{\substack{i \neq k \\ i \neq l}} g_i w_i &\equiv 0 \pmod{p} \\ \Leftrightarrow g_k v_l + g_l v_k + \sum_{\substack{i \neq k \\ i \neq l}} g_i v_i &\equiv 0 \pmod{p} \\ \Leftrightarrow g_k v_l + g_l v_k - g_k v_k - g_l v_l \\ &\quad + \sum_i g_i v_i \equiv 0 \pmod{p} \\ \Leftrightarrow g_k v_l + g_l v_k - g_k v_k - g_l v_l &\equiv 0 \pmod{p} \\ \Leftrightarrow (g_k - g_l)(v_l - v_k) &\equiv 0 \pmod{p} \\ \Leftrightarrow (g_k - g_l)(v_k - v_l) &\equiv 0 \pmod{p} \end{aligned}$$

Dieser Satz deckt auf, warum der Beispielcode aus Abbildung 6.3 die Vertauschung benachbarter Ziffern erkennen kann, nicht aber der Code aus Abbildung 6.2. Der zweite Code benutzt die Gewichte  $g_0 = 3, g_1 = 2, g_2 = 3$ , und das bedeutet, dass die Differenzen  $g_0 - g_1$  und  $g_1 - g_2$  beide Elemente der Menge  $\mathbb{Z}_4^*$  sind. Mit Satz 6.2 in Händen, können wir deshalb sicher sein, dass jeder Zifferndreher erkannt wird. Der erste Code verwendet hingegen das Gewicht  $g_1 = 1$ , sodass die Differenzen zweier benachbarter Gewichte kongruent zur Zahl 2 sind. Die Zahl 2 ist kein Element von  $\mathbb{Z}_4^*$ , und das bedeutet nach Satz 6.2, dass gewisse Zifferndreher zu Codewörtern führen und daher unerkannt bleiben.

■ EAN-Format



■ EAN-Überprüfung



**Abb. 6.4:** Aufbau der 13-stelligen Europäischen Artikelnummer (EAN-13)

## 6.2.3 Prüfziffercodes aus der Praxis

Nach den eher theoretisch geprägten Überlegungen ist es an der Zeit, einige Prüfziffercodes aus der Praxis zu betrachten.

■ Europäische Artikelnummer (EAN)

In Europa werden alle Handelsprodukte mit einer 8-stelligen oder einer 13-stelligen Nummer ausgezeichnet, die sowohl im Klartext als auch in Form eines Barcodes auf der Verpackung abgedruckt ist. Jedes Produkt ist durch diese *Europäische Artikelnummer*, kurz EAN, eindeutig identifiziert und kann durch den Einsatz von Barcode-Scannern schnell erkannt werden. Abbildung 6.4 erklärt den Aufbau einer EAN-13. Die ersten drei Ziffern bilden zusammen die Länderkennung. Danach folgen 9 Ziffern, die sich in zwei Abschnitte unterschiedlicher Länge aufteilen. Der erste codiert die Unternehmensnummer und der zweite die Artikelnummer.

Die letzte Ziffer ist die Prüfziffer, die eine EAN gegen Eingabefehler absichert. Diese wird so gewählt, dass jede 13-stellige Nummer

$$v_0 v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12}$$

die folgende Bedingung erfüllt:

$$\sum_{i \text{ gerade}} v_i + \sum_{i \text{ ungerade}} 3v_i \equiv 0 \pmod{10}$$

Etwas deutlicher können wir diese Vorschrift so aufschreiben:

$$(v_0 + v_2 + \dots + v_{12}) + (3v_1 + 3v_3 + \dots + 3v_{11}) \equiv 0 \pmod{10}$$

Unser bisher erworbene Wissen reicht aus, um aus dieser Vorschrift auf die Fehlererkennungsfähigkeiten des EAN-Codes schließen zu können. Der Code verwendet mit der Zahl 1 und der Zahl 3 nur zwei verschiedene Gewichte, die beide aus der Menge

$$\mathbb{Z}_{10}^* = \{1, 3, 7, 9\}$$

stammen. Das bedeutet nach Satz 6.1, dass eine einzige falsch eingegebene Ziffer mit Sicherheit zu einer ungültigen Nummer führt; der Fehler wird also in jedem Fall erkannt. Anders sieht es mit Fehlern aus, die durch die Vertauschung zweier Ziffern entstehen. Berechnen wir die Differenz zweier Gewichte, so erhalten wir als Ergebnis entweder die Zahl 0 oder die Zahl 2. Beide sind nicht in der Menge  $\mathbb{Z}_{10}^*$  enthalten, und das bedeutet nach Satz 6.2, dass der Code nicht in jedem Fall in der Lage ist, zwei vertauschte Ziffern zu erkennen.

Tabelle 1		Tabelle 2		Tabelle 3	
0001101 ↳ 0,0	0100111 ↳ 0,1	1110010 ↳ 0	000000 ↳ 0		
0011001 ↳ 1,0	0110011 ↳ 1,1	1100110 ↳ 1	001011 ↳ 1		
0010011 ↳ 2,0	0011011 ↳ 2,1	1101100 ↳ 2	001101 ↳ 2		
0111101 ↳ 3,0	0100001 ↳ 3,1	1000010 ↳ 3	001110 ↳ 3		
0100011 ↳ 4,0	0011101 ↳ 4,1	1011100 ↳ 4	010011 ↳ 4		
0110001 ↳ 5,0	0111001 ↳ 5,1	1001110 ↳ 5	011001 ↳ 5		
0101111 ↳ 6,0	0000101 ↳ 6,1	1010000 ↳ 6	011100 ↳ 6		
0111011 ↳ 7,0	0010001 ↳ 7,1	1000100 ↳ 7	010101 ↳ 7		
0110111 ↳ 8,0	0001001 ↳ 8,1	1001000 ↳ 8	010110 ↳ 8		
0001011 ↳ 9,0	0010111 ↳ 9,1	1110100 ↳ 9	011010 ↳ 9		

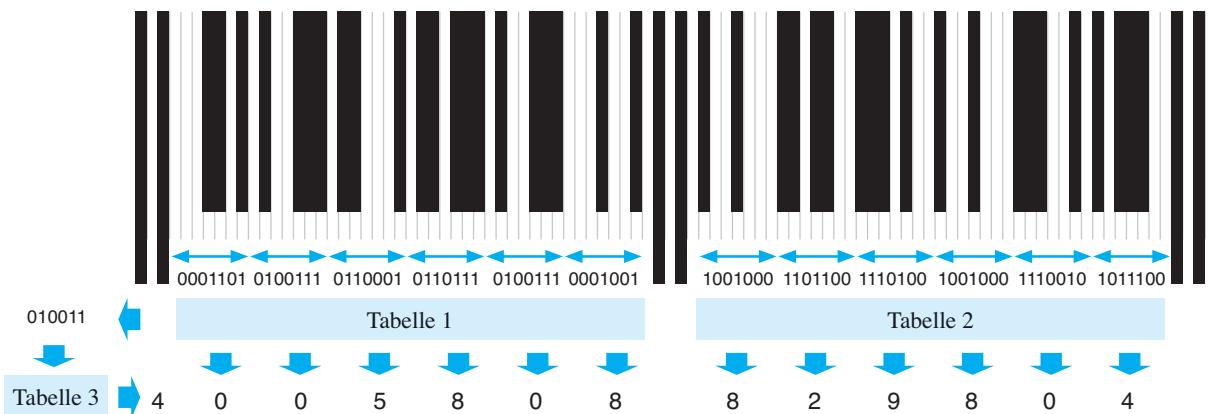


Abb. 6.5: An der Decodierung eines EAN-13-Barcodes sind drei Codetabellen beteiligt.

An dieser Stelle wollen wir einen Blick auf den Barcode werfen, der die EAN grafisch codiert. Abbildung 6.5 zeigt, dass ein EAN-13-Barcode aus zwei Hälften besteht, die durch eine spezielle, etwas nach unten verlängerte Markierung voneinander getrennt werden. Die gleiche Markierung finden wir auch an den Rändern wieder. Innerhalb der beiden Hälften sind jeweils 42 Strichlinien untergebracht, von denen jeweils 7 eine Ziffer codieren. Die Tabellen 1 und 2 in Abbildung 6.5 zeigen, dass in der linken und der rechten Hälfte jeweils andere Strichkombinationen verwendet werden. Im EAN-13-Barcode kommt der linken Hälfte noch eine weitere Bedeutung zu. Dort gibt es für jede Ziffer zwei verschiedene Codierungen, und die Information darüber, welche Strichkombination verwendet wird,

liefert für jede Ziffer ein Zusatzbit. Zusammen codieren diese Zusatzbits eine sechsstellige Binärzahl, die anhand einer weiteren Tabelle – in Abbildung 6.5 ist dies Tabelle 3 – in die führende Ziffer der EAN übersetzt wird. Die implizite Codierung der linken Ziffer ist der Grund, dass diese nicht unter dem EAN-Barcode, sondern etwas links davon abgedruckt wird.

#### ■ Internationale Standard-Buchnummer (ISBN)

Zur eindeutigen Identifikation von Büchern hat sich im Verlagswesen die Internationale Standard-Buchnummer, kurz ISBN, durchgesetzt. Bis Ende 2006 verbarg sich hinter der ISBN ein zehnstelliger Prüfziffercode

$$v_0 v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9,$$

der über die folgende Vorschrift definiert war:

$$\begin{aligned} 10v_0 + 9v_1 + 8v_2 + 7v_3 + 6v_4 + \\ 5v_5 + 4v_6 + 3v_7 + 2v_8 + 1v_9 \equiv 0 \pmod{11} \end{aligned} \quad (6.7)$$

Der ISBN-10-Code ist sowohl gegen Einzelfehler als auch gegen Vertauschungsfehler abgesichert, und mit unserem oben erworbenen Wissen können wir dies auch formal begründen. Mit der Zahl 11 wird eine Primzahl als Modul verwendet, sodass die Menge  $\mathbb{Z}_{11}^*$  alle Zahlen zwischen 1 und 10 umfasst:

$$\mathbb{Z}_{11}^* = \mathbb{Z}_{11} \setminus \{0\} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

Folgerichtig sind in der Menge  $\mathbb{Z}_{11}^*$  ausnahmslos alle Gewichte enthalten, die in der Kongruenz (6.7) auftauchen, sodass der Code nach Satz 6.1 jeden Einzelfehler erkennt. Bilden wir die Differenz zweier Gewichte, so ist das Ergebnis ebenfalls in der Menge  $\mathbb{Z}_{11}$  enthalten, und daraus folgt nach Satz 6.2, dass eine Vertauschung von zwei Ziffern immer erkannt wird, egal an welchen Positionen sich die betroffenen Ziffern befinden.

Die Wahl des Moduls führt zu einer Besonderheit des ISBN-10-Codes, die Ihnen bei der Betrachtung verschiedener Buchnummern vielleicht schon einmal aufgefallen ist. Aufgrund des gewählten Zahlenbereichs kann die Prüfstelle den Wert 10 annehmen und damit außerhalb des Wertebereichs liegen, der sich im Dezimalsystem mit einer einzelnen Ziffer abdecken lässt. Um eine ISBN in diesem Fall nicht auf 11 Stellen verlängern zu müssen, wird der Wert 10 durch den Buchstaben X dargestellt.

Am 1. Januar 2007 wurde die alte ISBN-10 durch die neue ISBN-13 abgelöst (Abbildung 6.6). Tatsächlich ist dieses Format für uns

nichts Neues: Eine ISBN-13 ist eine spezielle EAN, die entweder mit dem Präfix 978 oder mit einem der Präfixe 979-1 bis 979-9 beginnt. Alle alten ISBNs sind im Zahlenraum des neuen Formats enthalten. Um eine ISBN-10 in eine ISBN-13 zu übersetzen, muss lediglich das Präfix 978 vorangestellt und die letzte Stelle, die Prüfziffer, neu berechnet werden. Dies ist der Grund, warum sie die ersten 9 Stellen einer ISBN-10 stets im Klartext in der zugehörigen ISBN-13 wiederfinden.

Die Integration der ISBN in die Artikelnummer (EAN) bringt eine Vereinfachung in der Logistik mit sich, verschlechtert aber die Fehlererkennungseigenschaft. Weiter oben haben wir herausgearbeitet, dass die Vertauschung zweier Ziffern in einer EAN nicht immer erkannt werden kann und diese Limitierung überträgt sich direkt auf die neuen ISBNs. Betrachten wir die ISBNs also lediglich im Hinblick auf deren codierungstheoretische Eigenschaften, so ist das alte Format dem neuen überlegen.

### ■ Pharmazentralnummer (PZN)

In Deutschland sind alle Arzneimittel mit einer eindeutigen Pharmazentralnummer, kurz PZN, ausgezeichnet. Hierbei handelt es sich um einen 7-stelligen Prüfziffercode

$$v_0 v_1 v_2 v_3 v_4 v_5 v_6,$$

der über die folgende Vorschrift definiert ist (Abbildung 6.7):

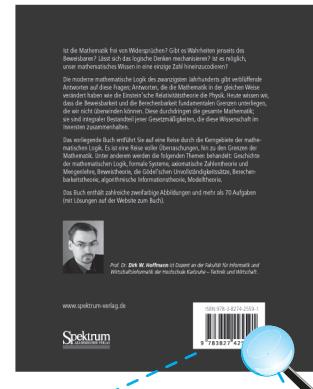
$$2v_0 + 3v_1 + 4v_2 + 5v_3 + 6v_4 + 7v_5 \equiv v_6 \pmod{11} \quad (6.8)$$

Diese Formel weicht geringfügig von dem Aufbau ab, den wir in Definition 6.1 gefordert haben. Dies lässt sich korrigieren, indem wir auf beiden Seiten  $v_6$  subtrahieren, oder was in der Modulo-11-Rechnung dasselbe ist: Indem wir auf beiden Seiten 10-mal  $v_6$  addieren. Die Formel (6.8) nimmt dann die folgende, äquivalente Form an:

$$2v_0 + 3v_1 + 4v_2 + 5v_3 + 6v_4 + 7v_5 + 10v_6 \equiv 0 \pmod{11}$$

Durch den Modul 11 kann, genau wie bei der ISBN-10, die Prüfziffer 10 entstehen. Anders als bei der ISBN, die in diesem Fall das spezielle Symbol X verwendet, löst die PZN das Problem ganz einfach dadurch, dass solche Nummern nicht vergeben werden. Per Definition ist eine Zifferfolge also nur dann eine gültige PZN, wenn die Prüfziffer  $v_6$  die Beziehung  $0 \leq v_6 \leq 9$  erfüllt.

Auch die Fehlererkennungseigenschaften der PZN ähneln jenen der ISBN-10. Alle verwendeten Gewichte sind in der Menge  $\mathbb{Z}_{11}^*$  enthalten, sodass sämtliche Fehler erkannt werden, die durch eine einzelne



ISBN 978-3-8274-2559-1



**Abb. 6.6:** Aufbau des ISBN-13-Formats



**Abb. 6.7:** Die Pharmazentralnummer, kurz PZN, ist ein 7-stelliger Prüfziffercode.

falsch eingegebene Ziffer entstehen. Die Differenz zweier beliebiger Gewichte ist ebenfalls ein Element von  $\mathbb{Z}_{11}^*$ , sodass auch ein Vertauschungsfehler sicher erkannt wird, egal, an welchen Positionen sich die Ziffern befinden.

### ■ Luhn-Codes

In den 60er-Jahren schlug der deutsche Informatiker Hans Peter Luhn einen Prüfziffercode vor, der heute unter anderem für die Absicherung von Kreditkartennummern eingesetzt wird. Der Luhn-Code verwendet den Modul 10 und benutzt abwechselnd die Gewichte 2 und 1. Er ist dem Prüfziffercode, der über die Beziehung

$$\sum_{i \text{ gerade}} 2v_i + \sum_{i \text{ ungerade}} v_i \equiv 0 \pmod{10} \quad (6.9)$$

definiert ist, sehr ähnlich, unterscheidet sich aber in einem zunächst unscheinbaren, aber wesentlichen Punkt. Entsteht im Luhn-Code durch die Multiplikation einer Ziffer mit dem Gewicht 2 eine Zahl  $\geq 10$ , so geht diese Zahl in (6.9) mit ihrer Quersumme ein, d. h., sie wird ziffernweise addiert. Was dies genau bedeutet, demonstriert Abbildung 6.8 an einem konkreten Beispiel.

Mathematisch können wir die Quersummenregel in die Formel (6.9) integrieren, indem wir das Gewicht  $2v_i$  für den Fall  $v_i \geq 5$  um 9 reduzieren:

$$\sum_{\substack{i \text{ gerade} \\ v_i \geq 5}} (2v_i - 9) + \sum_{\substack{i \text{ gerade} \\ v_i < 5}} 2v_i + \sum_{i \text{ ungerade}} v_i \equiv 0 \pmod{10} \quad (6.10)$$

Obwohl die zusätzlich hinzugenommene Quersummenregel verhindert, dass wir die Sätze 6.1 und 6.2 direkt anwenden können, erkennt der Luhn-Code immer noch alle Einzelfehler. Die Vertauschung zweier benachbarter Ziffern wird hingegen nicht mehr in jedem Fall bemerkt. Beginnt die Ziffernfolge beispielsweise mit 09, so gehen diese Ziffern mit  $2 \cdot 0 + 9 = 9$  in die Summe (6.10) ein. Vertauschen wir die beiden Ziffern, so erhalten wir mit  $2 \cdot 9 - 9 + 0 = 9$  genau das gleiche Ergebnis. Der Code ist damit nicht in der Lage, diesen Zahlendreher zu erkennen.

### ■ Seriennummer des deutschen Personalausweises

Jeder deutsche Personalausweis ist mit einer neunstelligen alphanumerischen Seriennummer versehen, die neben den Ziffern 0 bis 9 auch Buchstaben aus der Menge

$$\{C, F, G, H, J, K, L, M, N, P, R, T, V, W, X, Y, Z\}$$

Eine Kreditkartennummer, die den *Luhn-Test* besteht

8	3	3	0	2	3	0	1	7	6	0	2	4	7	6	5
·2	·2	·2	·2	·2	·2	·2	·2	·2	·2	·2	·2	·2	·2	·2	
16	6	4	0	14	0	8	12								
$7+3+6+0 + 4+3+0+1 + 5+6+0+2 + 8+7+3+5$															
$= 60 \equiv 0 \pmod{10} \quad \checkmark$															

Eine plumpfe Fälschung

8	6	6	0	0	1	0	1	2	2	4	2	3	7	1	0
·2	·2	·2	·2	·2	·2	·2	·2	·2	·2	·2	·2	·2	·2	·2	
16	12	0	0	4	8	2	6	2							
$7+6+3+0 + 0+1+0+1 + 4+2+8+2 + 6+7+2+0$															
$= 49 \not\equiv 0 \pmod{10} \quad \times$															

Abb. 6.8: Der Luhn-Algorithmus wird unter anderem für die Verifikation von Kreditkartennummern eingesetzt.

enthalten darf. Auf die Verwendung der Vokale A, E, I, O, U und der Konsonanten B, D, Q, S wurde aus Gründen der OCR-Lesbarkeit verzichtet. Die Seriennummer ist sowohl auf der Vorderseite als auch auf der Rückseite abgedruckt. Wie in Abbildung 6.9 zu sehen ist, wird sie auf der Rückseite durch eine angehängte Prüfziffer  $v_9$  ergänzt, die über die folgende Vorschrift gebildet wird:

$$7v_0 + 3v_1 + 1v_2 + 7v_3 + 3v_4 + 1v_5 + 7v_6 + 3v_7 + 1v_8 \equiv v_9 \pmod{10}$$

Ein alphanumerisches Zeichen geht über die Zuordnung

A	B	C	D	E	F	G	H	I	J	K	L	...
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	...
10	11	12	13	14	15	16	17	18	19	20	21	...

in die Summe ein. Beachten Sie, dass der Prüfziffercode des Personalausweises, genau wie im Falle der Kreditkartennummer, lediglich der Erkennung von Eingabefehlern und nicht der Erhöhung der Fälschungssicherheit dient.

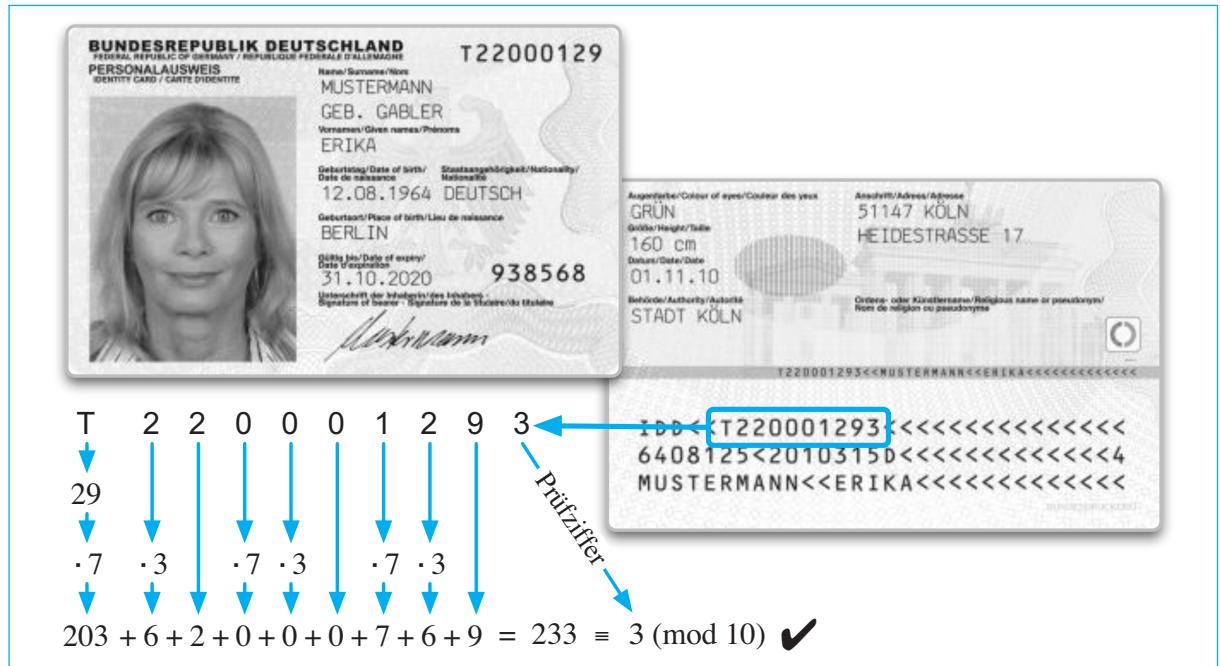


Abb. 6.9: Die Seriennummer eines deutschen Personalausweises ist ein alphanumerischer Prüfziffercode.

Die Fehlererkennungseigenschaften können wir, genau wie bei den anderen Codes, direkt aus der Berechnungsvorschrift ableiten. Der Code verwendet die Gewichte 7, 3 und 1, die allesamt in der Menge  $\mathbb{Z}_{10}^*$  enthalten sind. Das bedeutet nach Satz 6.1, dass die Eingabe einer einzelnen falschen Ziffer mit Sicherheit erkannt wird. Einen Vertauschungsfehler kann der Code dagegen nicht in jedem Fall erkennen. Beispielsweise ist die Differenz der Gewichte 7 und 3 gleich 4 und somit kein Element von  $\mathbb{Z}_{10}^*$ . Aus Satz 6.2 folgt daher, dass nicht alle Eingabefehler, die durch eine Vertauschung von zwei Ziffern entstehen, erkannt werden.

## 6.3 Fehlererkennung und -korrektur

*„I was using a 2-out-of-five encoded machine that had the added feature that when it located an error it would try again: indeed it would try three times in all before giving up on a problem. Two Monday mornings in a row I found that shortly after leaking on Friday night the machine failed, and as a result I got nothing. The second time I was sufficiently angry that I said: 'If the machine can find out that there is an error, why can it not locate its position?'. This was the start, in 1948, of the theory of error-correcting codes.“*

Richard W. Hamming [7]

Um die Kanalcodierung zu motivieren, haben wir im letzten Abschnitt einen Blick in die Praxis geworfen. Wir haben dort mehrere Codierungen betrachtet, mit denen wir nahezu alltäglich konfrontiert werden, und gezeigt, wie durch die Hinzunahme spezieller Prüfziffern falsch eingegebene oder falsch übertragene Ziffernfolgen erkannt werden können. Im Folgenden werden wir die Thematik von ihrer theoretischen Seite beleuchten und neben zahlreichen Codierungen, die Fehler erkennen können, auch solche einführen, die Fehler korrigieren können. Die Fehlererkennung und die Fehlerkorrektur sind so wichtige Begriffe, dass wir sie in einer eigenen Definition formal voneinander abgrenzen wollen:

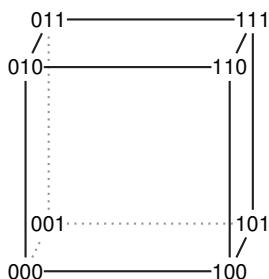


### Definition 6.2

- Ein Code heißt *r-fehlererkennend*, wenn der Empfänger jedes Codewort, in dem bis zu  $r$  Symbole verfälscht wurden, als fehlerhaft identifizieren kann.
- Ein Code heißt *r-fehlererkorrigierend*, wenn der Empfänger jedes Codewort, in dem bis zu  $r$  Symbole verfälscht wurden, wiederherstellen kann.

In den vorherigen Kapiteln haben wir bereits mehrere fehlererkennende und -korrigierende Codes kennengelernt. Einer dieser Codes ist der gerade *Paritätscode* der Länge  $n$ , den wir auf Seite 189 eingeführt haben. Per Definition umfasst er genau jene Bitvektoren aus der Menge  $\{0, 1\}^n$ , in denen die Anzahl der Einsen gerade ist.

Hamming-Würfel der Dimension 3



**Abb. 6.10:** Allgemeine Form eines Hamming-Würfels der Dimension 3. Jede Ecke entspricht einem möglichen Codewort aus der Menge  $\{0, 1\}^3$ .

Ist das Codealphabet  $\Pi$  die Menge  $\{0, 1\}$ , so wird die Anzahl der Einsen eines Codeworts als *Hamming-Gewicht* bezeichnet. Ganz allgemein ist dieser Begriff folgendermaßen definiert:



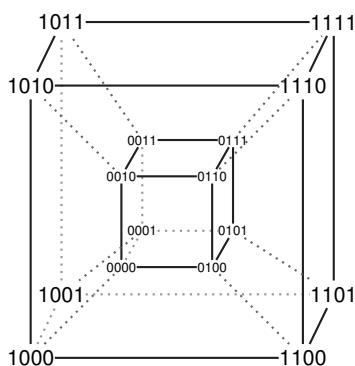
### Definition 6.3 (Hamming-Gewicht)

Das Hamming-Gewicht  $\omega(v)$  gibt an, wie viele Symbole in  $v$  von 0 verschieden sind.

Diese Definition ist unter der Annahme verfasst, dass das Codealphabet  $\Pi$  überhaupt ein dediziertes Nullsymbol enthält; ansonsten verliert der Wortlaut seinen Sinn. Eine große Einschränkung ist dies nicht, da wir fast ausschließlich Codes betrachten, deren Codealphabet ein Körper ist, und in diesen Fällen immer ein Nullelement existiert.

Mit dem neu eingeführten Begriff in Händen können wir den geraden Paritätscode so charakterisieren: Er umfasst genau jene Bitvektoren aus der Menge  $\{0, 1\}^n$ , deren Hamming-Gewicht gerade ist. Es ist leicht einzusehen, dass der Paritätscode 1-fehlererkennend ist. Wird ein einzelnes Bit auf der Übertragungsstrecke verändert, so verliert das Codewort die Eigenschaft, eine gerade Anzahl Einsen aufzuweisen. Der Empfänger kann deshalb in jedem Fall erkennen, ob das Codewort durch einen einzelnen Bitfehler verfälscht wurde oder nicht. Dieses Ergebnis lässt sich noch allgemeiner formulieren: Der Paritätscode erkennt einen Übertragungsfehler genau dann, wenn das gesendete Codewort durch eine ungerade Anzahl an Bitfehlern verfälscht wurde. Trotzdem ist der Code nicht 3-fehlererkennend. Hierzu müsste er nach unserer Definition auch 2-fehlererkennend sein, was er nicht ist; wird das Codewort an einer geraden Anzahl Bitstellen verfälscht, so bleibt die Anzahl der Einsen gerade, und der Empfänger hat keine Möglichkeit, den Fehler zu erkennen.

Hamming-Würfel der Dimension 4



**Abb. 6.11:** Allgemeine Form eines Hamming-Würfels der Dimension 4. Jede Ecke entspricht einem möglichen Codewort aus der Menge  $\{0, 1\}^4$ .

### 6.3.1 Hamming-Distanz

Für die folgenden Betrachtungen ist es hilfreich, sich die Codewörter eines Binärcodes der Länge  $n$  in einer grafischen Anordnung vorzustellen, wie sie in Abbildung 6.10 für den Fall  $n = 3$  dargestellt ist. Sie sehen dort die allgemeine Form des *Hamming-Würfels*, der nach dem amerikanischen Mathematiker Richard W. Hamming benannt ist. Aufgebaut ist der Würfel so, dass jede Achse eine Bitposition repräsentiert; in unserem Fall ist die  $x$ -Achse dem ersten Bit zugeordnet, die  $y$ -Achse

dem zweiten und die  $z$ -Achse dem dritten. Auf diese Weise entspricht jeder Bitvektor der Menge  $\{0, 1\}^3$  einer gewissen Ecke des Würfels. Die Codewörter, die mit einer Kante direkt verbunden sind, besitzen die Eigenschaft, dass sie sich an genau einer Bitstelle unterscheiden. Wir sagen, diese Codewörter haben die *Hamming-Distanz* 1, und verallgemeinern diesen Begriff in naheliegender Weise:



#### Definition 6.4 (Hamming-Distanz)

Es sei  $C$  ein Code fester Länge und  $v_1, v_2$  zwei Codewörter aus  $C$ . Die *Hamming-Distanz*  $\Delta(v_1, v_2)$  ist die Anzahl der Positionen, an denen die Symbole in  $v_1$  und  $v_2$  unterschiedlich sind.

Beispielsweise ist

$$\begin{aligned}\Delta(011, 011) &= 0 & \Delta(011, 001) &= 1 \\ \Delta(101, 000) &= 2 & \Delta(000, 111) &= 3\end{aligned}$$

In der Literatur wird die Hamming-Distanz auch gerne als *Hamming-Abstand* bezeichnet. Im Folgenden werden wir uns meist etwas kürzer fassen und nur noch von der *Distanz* oder dem *Abstand* zweier Codewörter sprechen.

Über den Distanzbegriff können wir dem Hamming-Gewicht eines Codeworts eine weitere intuitive Bedeutung verleihen. Das Hamming-Gewicht  $\omega(v)$  entspricht der Distanz zwischen dem Codewort  $v$  und dem Nullwort  $\mathbf{0}$ , es gilt also

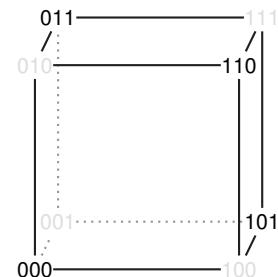
$$\omega(v) = \Delta(v, \mathbf{0}) = \Delta(\mathbf{0}, v)$$

An der Würfeldarstellung lässt sich die Distanz zweier Codewörter direkt ablesen:  $\Delta(v_1, v_2)$  entspricht der Anzahl der Kanten, die mindestens durchlaufen werden müssen, um von  $v_1$  zu  $v_2$  zu gelangen. Leider stößt die Würfeldarstellung an ihre Grenzen, wenn wir dazu übergehen, längere Codewörter zu betrachten. Wie der Graph in Abbildung 6.11 zeigt, ist eine übersichtliche Darstellung von Binärcodes der Länge 4 noch möglich, für größere Bitbreiten wird eine räumliche Darstellung allerdings schnell unübersichtlich.

In Abbildung 6.12 ist der Hamming-Würfel des  $n$ -stelligen Paritätscodes zu sehen, einmal für den Fall  $n = 3$  und einmal für den Fall  $n = 4$ . An der Würfeldarstellung lässt sich mit einem einzigen Blick überprüfen, dass der Paritätscode 1-fehlererkennend ist. Die Codewörter sind so auf die Ecken des Würfels verteilt, dass sich niemals zwei davon

#### Gerader Paritätscode der Länge 3

$$C = \{000, 011, 101, 110\}$$



#### Gerader Paritätscode der Länge 4

$$C = \{0000, 0011, 0101, 0110, 1001, 1010, 1100, 1111\}$$

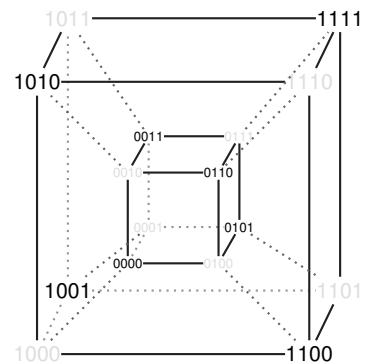
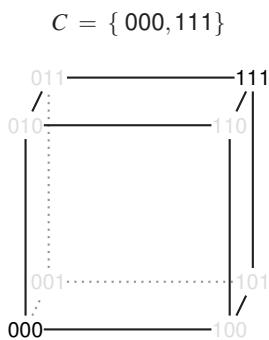


Abb. 6.12: Würfeldarstellung der geraden Paritätscodes der Längen 3 und 4. Die Code-Distanz ist in beiden Fällen gleich 2.

Wiederholungscode der Länge 3

**Abb. 6.13:** Würfeldarstellung des Wiederholungscodes der Länge 3

in unmittelbarer Nähe befinden. Wird in einem gesendeten Codewort ein einzelnes Bit verfälscht, so erhält der Empfänger eine Bitsequenz, die eine Kante von dem ursprünglichen Codewort entfernt ist. Die entsprechende Ecke des Würfels ist aber nicht mit einem Codewort belegt, sodass der Empfänger zweifelsfrei feststellen kann, dass ein Übertragungsfehler aufgetreten ist.

Fehlerkorrigierend ist der Paritätscode nicht. Wird beispielsweise die Bitsequenz 111 empfangen, so kann sie durch die Verfälschung eines einzigen Bits aus drei Codewörtern entstanden sein: den Codewörtern 011, 101 und 110. Im ersten Fall wäre das erste Bit beschädigt worden, im zweiten Fall das zweite und im dritten Fall das dritte. Selbst unter der Annahme, dass nur ein einziges Bit fehlerhaft ist, hat der Empfänger keine Möglichkeit, das gesendete Codewort zu identifizieren.

Ein einfacher 1-fehlerkorrigierender Code ist in Abbildung 6.13 zu sehen. Es ist der sogenannte *Wiederholungscode*, der aus den beiden Codewörtern 000 und 111 besteht und in der Würfeldarstellung zwei gegenüberliegende Ecken belegt. Die Codewörter sind nun so weit voneinander entfernt, dass ein einzelnes verfälschtes Bit korrigiert werden kann. Hierzu muss der Empfänger lediglich die Bitsequenzen 100, 010, 001 auf das Codewort 000 und die Bitsequenzen 011, 101, 110 auf das Codewort 111 abbilden.

Mit den angestellten Überlegungen ist es uns gelungen, das wesentliche Merkmal fehlererkennender und fehlerkorrigierender Codes offenzulegen: In solchen Codes sind die Codewörter so gut verteilt, dass der minimale Abstand zweier Codewörter maximal ist.

### 6.3.2 Code-Distanz

Der minimale Abstand zweier Codewörter wird in der Codierungstheorie als *Code-Distanz* bezeichnet:



#### Definition 6.5 (Code-Distanz)

Für einen Code  $C$  fester Länge ist die Code-Distanz  $\Delta C$  die minimale Distanz zwischen zwei Codewörtern:

$$\Delta C := \min \{ \Delta(v_1, v_2) \mid v_1, v_2 \in C, v_1 \neq v_2 \}$$

Die Code-Distanz des Paritätscodes ist gleich 2, da sich zwei beliebig herausgegriffene Codewörter an mindestens 2 Bitpositionen unterscheiden.

den. Im Wiederholungscode aus Abbildung 6.13 sind die Codewörter noch weiter voneinander entfernt; die Distanz zwischen 000 und 111 beträgt 3, und da diese beiden Bitsequenzen die einzigen Codewörter sind, ist auch die Code-Distanz gleich 3.

Zwischen der Distanz und der Fehlererkennungseigenschaft eines Codes besteht ein elementarer Zusammenhang. Um diesen offenzulegen, definieren wir für jedes Codewort  $v$  und jede natürliche Zahl  $r$  eine Menge  $K_r(v)$ , die wir fortan als *Hamming-Kugel* von  $v$  bezeichnen:



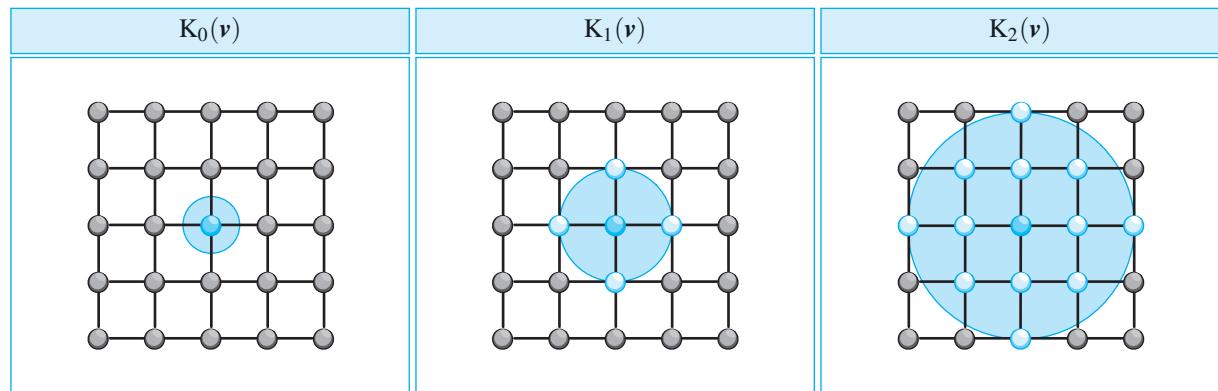
### Definition 6.6 (Hamming-Kugel)

Es seien  $C \subseteq \Pi^n$  ein Code fester Länge,  $v \in C$  und  $r \in \mathbb{N}$ . Die Menge

$$K_r(v) := \{w \in C \mid \Delta(v, w) \leq r\}$$

bezeichnen wir als die *Hamming-Kugel* von  $v$  mit dem Radius  $r$ .

Dieser Definition folgend, enthält die Menge  $K_r(v)$  genau diejenigen Codewörter, die sich von  $v$  an höchstens  $r$  Positionen unterscheiden, oder was dasselbe ist: deren Distanz zu  $v$  höchstens  $r$  beträgt. Abbildung 6.14 zeigt am Beispiel des Codeworts 001, wie diese Mengen konkret aussehen. Ist  $n$  die Länge der Codewörter, so können wir uns die Menge  $K_r(v)$  als eine  $n$ -dimensionale Kugel mit dem Radius  $r$  vorstellen (Abbildung 6.15).



**Abb. 6.15:** Ist  $n$  die Länge der Codewörter, so können wir uns die Menge  $K_r(v)$  als eine Kugel mit dem Zentrum  $v$  vorstellen, die sich mit dem Radius  $r$  im  $n$ -dimensionalen Raum ausbreitet. In den Grafiken steht jeder Punkt für eine Bitsequenz, und eine Kante bedeutet, dass sich die verbundenen Sequenzen an genau einer Position unterscheiden.

$$\Pi = \{0, 1\}$$

$$\begin{aligned} \Pi^3 = \{ &000, 001, 010, 011, \\ &100, 101, 110, 111 \} \end{aligned}$$



$$K_0(001) = \{001\}$$

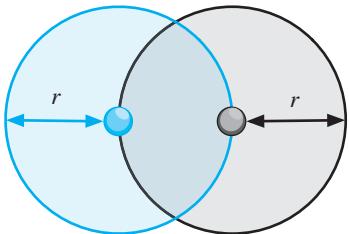
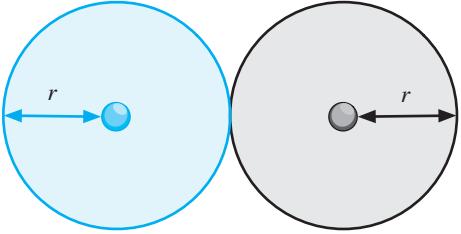
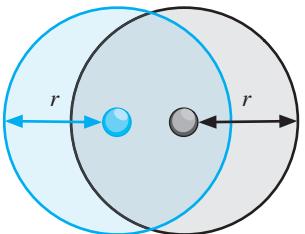
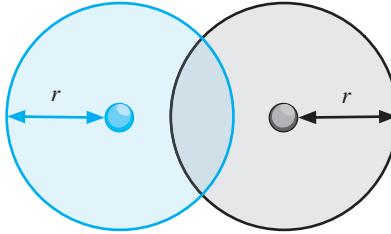
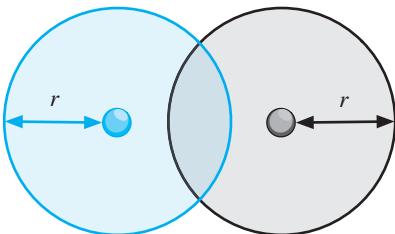
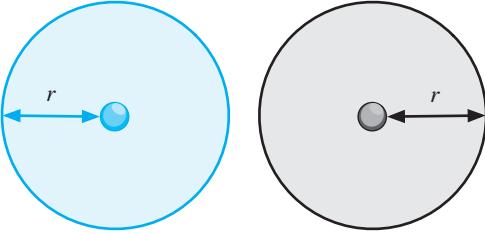
$$K_1(001) = \{001, \\ 000, 011, 101\}$$

$$K_2(001) = \{001, \\ 000, 011, 101, \\ 010, 100, 111\}$$

$$K_3(001) = \{001, \\ 000, 011, 101, \\ 010, 100, 111, \\ 110\}$$

$$K_k(001) = K_3(001) \quad (\text{für } k > 3)$$

**Abb. 6.14:** Hamming-Kugeln

$r$ -fehlererkennende Codes	$r$ -fehlerkorrigierende Codes
<p>Sind zwei Hamming-Kugeln so nahe beisammen, dass sich ein Codewort auf dem Rand befindet</p> 	<p>Sind zwei Hamming-Kugeln so nahe beisammen, dass sie sich berühren</p> 
<p>oder innerhalb der anderen Kugel befindet,</p> 	<p>oder gar schneiden,</p> 
<p>so kann es passieren, dass ein Codewort durch die Veränderung von <math>r</math> oder weniger Bits zu einem anderen Codewort wird. Der Code ist dann nicht <math>r</math>-fehlererkennend.</p> <p>Erst wenn die Code-Distanz den Wert <math>r</math> übersteigt, sind alle Kugeln frei von anderen Codewörtern. Jetzt ist der Code <math>r</math>-fehlererkennend.</p> 	<p>so können die Bitsequenzen aus der Schnittmenge durch die Veränderung von <math>r</math> oder weniger Bits aus beiden Codewörtern entstanden sein. Eine eindeutige Zuordnung ist nicht mehr möglich und der Code nicht <math>r</math>-fehlerkorrigierend.</p> <p>Erst wenn die Code-Distanz den Wert <math>2r</math> übersteigt, ist eine eindeutige Zuordnung möglich. Jetzt ist der Code <math>r</math>-fehlerkorrigierend.</p> 

**Abb. 6.16:** Die Code-Distanz, d.h. die minimale Distanz zweier Codewörter, bestimmt, wie viele verfälschte Bits ein Code erkennen oder korrigieren kann.

Um den Zusammenhang zwischen der Distanz eines Codes und dessen Fehlererkennungseigenschaften zu verstehen, blicken wir auf die linke Seite in Abbildung 6.16. Bilden wir um jedes Codewort eine Kugel mit dem Radius  $r$ , so ist der Code genau dann  $r$ -fehlererkennend, wenn kein Codewort auf dem Rand oder innerhalb einer anderen Kugel liegt. Nur dann ist sichergestellt, dass eine Veränderung an bis zu  $r$  Positionen stets zu einer Symbolsequenz führt, die nicht zu den Codewörtern gehört. Weit genug voneinander entfernt sind die Kugeln also genau dann, wenn die Code-Distanz die Zahl  $r$  übersteigt.

Für die Korrektur von Übertragungsfehlern gilt etwas ganz Ähnliches. Ist die Code-Distanz größer als  $2r$ , so sind die Kugeln so weit voneinander entfernt, dass sie sich weder schneiden noch berühren, und wir können jede Symbolsequenz, die sich an  $r$  oder weniger Positionen von einem Codewort unterscheidet, eindeutig einer einzigen Hamming-Kugel zuordnen. Wurden höchstens  $r$  Symbole verfälscht, ist somit eine eindeutige Korrektur möglich.

Berühren oder schneiden sich zwei Kugeln, so verliert diese Argumentation ihre Gültigkeit. Da für jede Symbolsequenz aus der Schnittmenge mehrere Kugeln als möglicher Ursprung in Frage kommen, lässt sich das ursprünglich gesendete Codewort nicht mehr eindeutig rekonstruieren.

Insgesamt haben uns die Überlegungen ein zentrales Ergebnis der Kanalcodierung in die Hände gespielt, auf das wir in den folgenden Abschnitten immer wieder zurückkommen werden:



### Satz 6.3

Für einen Code  $C$  fester Länge gilt:

$$C \text{ ist } r\text{-fehlererkennend} \Leftrightarrow \Delta C > r$$

$$C \text{ ist } r\text{-fehlerkorrigierend} \Leftrightarrow \Delta C > 2r$$

In Abbildung 6.17 ist für einige ausgewählte Beispielwerte zusammengefasst, wie die Steigerung der Code-Distanz die Fähigkeit eines Codes beeinflusst, Fehler zu erkennen oder zu korrigieren.

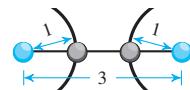
Die Grafiken offenbaren einen weiteren elementaren Zusammenhang: Ist die Code-Distanz eine ungerade Zahl, so wirkt sich deren Erhöhung um eins nicht auf die Fähigkeit aus, Fehler zu korrigieren, aber auf die Fähigkeit, Fehler zu erkennen: Die Anzahl der Fehler, die ein Code erkennen kann, steigt dann ebenfalls um eins.

$$\Delta C = 2$$



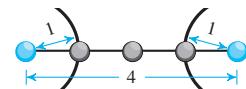
1-fehlererkennend, 0-fehlerkorrigierend

$$\Delta C = 3$$



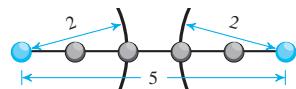
2-fehlererkennend, 1-fehlerkorrigierend

$$\Delta C = 4$$



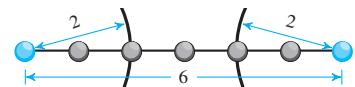
3-fehlererkennend, 1-fehlerkorrigierend

$$\Delta C = 5$$



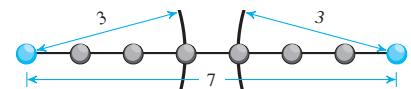
4-fehlererkennend, 2-fehlerkorrigierend

$$\Delta C = 6$$



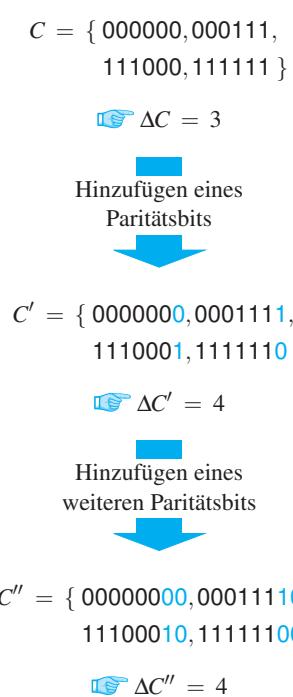
5-fehlererkennend, 2-fehlerkorrigierend

$$\Delta C = 7$$



6-fehlererkennend, 3-fehlerkorrigierend

**Abb. 6.17:** Die Code-Distanz bestimmt, wie viele Fehler ein Code erkennen bzw. korrigieren kann.



**Abb. 6.18:** Da die Code-Distanz von  $C$  ungerade ist, wird sie durch das Anfügen eines Paritätsbits erhöht. Die Hinzunahme eines zweiten Paritätsbits bleibt dagegen ohne Wirkung. Das Beispiel belegt, dass wir in Satz 6.4 nicht auf die Voraussetzung verzichten können, die Code-Distanz von  $C$  sei ungerade.

## Erhöhung der Code-Distanz

Tatsächlich gibt es einen einfachen Trick, um eine ungerade Code-Distanz um eins zu erhöhen. Es reicht in diesem Fall aus, die Codewörter so um ein Paritätsbit zu ergänzen, dass die Anzahl der Einsen in den neuen Codewörtern entweder gerade oder ungerade wird.

Wir wollen uns genauer überlegen, warum dieser Trick funktioniert. Einerseits ist klar, dass sich die Code-Distanz durch das Anfügen zusätzlicher Bits nicht verringern kann. Andererseits kann sie nicht gleich bleiben: Das Paritätsbit sorgt dafür, dass sich zwei Codewörter immer an einer geraden Anzahl Einsen unterscheiden und die Code-Distanz daher eine gerade Zahl sein muss. Da die Code-Distanz vorher ungerade war, gibt es nur eine schlüssige Erklärung: Die Hinzunahme des Paritätsbits muss zwangsläufig zu einer Erhöhung der Code-Distanz geführt haben.

Alles in allem haben unsere Überlegungen ein interessantes Ergebnis hervorgebracht, das wir an späterer Stelle in diesem Buch gewinnbringend einsetzen werden:



### Satz 6.4

Ist  $C$  ein Code mit einer ungeraden Code-Distanz und  $C'$  der um ein Paritätsbit *erweiterte Code*, so gilt die Beziehung

$$\Delta C' = \Delta C + 1$$

Beachten Sie, dass es keine Rolle spielt, ob wir den Code um ein gera des oder ein ungerades Paritätsbit ergänzen. Die vorgetragene Argumentation funktioniert in beiden Fällen gleichermaßen.

Unverzichtbar ist dagegen die in Satz 6.4 genannte Voraussetzung, die Code-Distanz von  $C$  sei ungerade. Warum dies so ist, macht das Beispiel in Abbildung 6.18 deutlich. Dort wird ein Code  $C$  zweimal hintereinander um ein Paritätsbit ergänzt, wobei nur die erste Ergänzung die in Satz 6.4 postulierte Wirkung zeigt. Das zweite Paritätsbit ist für alle Codewörter gleich und wirkt sich daher nicht auf die Code-Distanz aus.

## 6.4 Lineare Kanalcodes

In Abschnitt 3.4.3 haben wir uns ausführlich mit *linearen Codes* beschäftigt und anhand ausgewählter Beispiele deren Praxisrelevanz unter Beweis gestellt. Die willkommene Eigenschaft, einen Vektorraum zu bilden, hat uns dabei mehrfach in die Lage versetzt, lineare Codes in hoher Präzision zu untersuchen, und auch in diesem Abschnitt werden wir den mathematischen Werkzeugkasten wieder öffnen. Wir wollen eine algebraische Antwort für die Frage erarbeiten, wie Codewörter, die auf der Übertragungsstrecke verfälscht wurden, auf der Empfängerseite korrigiert werden können.

Bevor wir in die Tiefen dieser Materie eintauchen, blicken wir erneut auf Satz 6.3. Dieser Satz stellt einen direkten Zusammenhang zwischen der Code-Distanz und der Anzahl der Fehler her, die ein Code erkennen bzw. korrigieren kann, und offenbart, warum der Begriff der Code-Distanz für die Kanalcodierung so wichtig ist. Wir wollen dies auch in der benutzten Schreibweise ausdrücken und greifen zu diesem Zweck eine in Definition 3.4 vereinbarte Notation auf. Dort hatten wir von einem  $[n, k]$ -Code gesprochen, wenn die Menge der Codewörter einen  $k$ -dimensionalen Untervektorraum von  $\mathbb{F}^n$  bildete. Weiter unten werden wir das Präfix  $[n, k]$  an manchen Stellen um eine dritte Komponente erweitern und etwas ausführlicher von einem  $[n, k, d]$ -Code sprechen. Der zusätzliche Parameter  $d$  ist die Distanz des betrachteten Codes, die uns, mit Satz 6.3 im Gedächtnis, einen direkten Rückschluss auf dessen Kanalcodierungseigenschaften erlaubt.

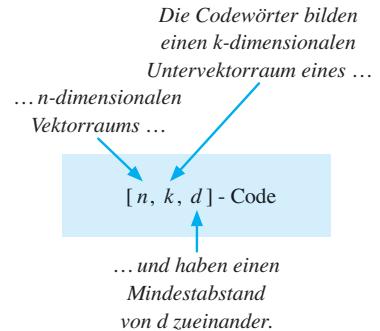
In analoger Weise werden wir ab und an von einer  $[n, k, d]$ -Codierung sprechen und damit eine lineare Codierung meinen, die jeweils  $k$  Quellensymbole so auf ein Codewort der Länge  $n$  abbildet, dass zwei beliebige Codewörter einen Mindestabstand von  $d$  zueinander aufweisen (Abbildung 6.19).

Bei systematischen Binärcodierungen, d. h. bei Binärcodierungen, in denen die Nachricht innerhalb des generierten Codeworts im Klartext enthalten ist, hat die Differenz  $n - k$  eine intuitive Bedeutung: Sie drückt in diesem Fall aus, wie viele *Prüfbits* der Nachricht im Zuge der Codierung hinzugefügt werden.

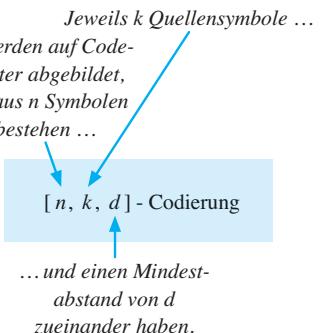
### 6.4.1 Syndromdecodierung

Mit  $c$  sei eine lineare Codierung gegeben, die Nachrichten auf Codewörter der Länge  $n$  abbildet. Da  $c$  linear ist, bilden die Codewörter per

#### ■ Linearer $[n, k, d]$ -Code



#### ■ Lineare $[n, k, d]$ -Codierung



**Abb. 6.19:** Notation für lineare Codes und Codierungen

Definition einen Untervektorraum von  $\mathbb{F}^n$ . Wir beschränken uns an dieser Stelle, wie so oft in diesem Buch, auf binäre Codes, d. h., wir wählen für den endlichen Körper  $\mathbb{F}$  die zweielementige Menge  $\mathbb{Z}_2 = \{0, 1\}$ .

In den folgenden Betrachtungen bezeichnen wir mit  $v$  das Codewort, das der Sender auf die Übertragungsstrecke geschickt hat, und mit  $w$  die Bitsequenz, die an der Empfängerseite angekommen ist. Ist  $v = w$ , so ist eine fehlerfreie Übertragung gelungen. Ist  $v \neq w$ , so wurde das Codewort auf der Übertragungsstrecke beschädigt, und genau für diesen Fall interessieren wir uns in diesem Kapitel.

Mathematisch können wir einen Übertragungsfehler sehr einfach modellieren, indem wir  $w$  in der Form

$$w = v + e \quad (6.11)$$

angeben. Der Vektor  $e$  heißt *Fehlervektor* und enthält genau dort eine 1, wo das gesendete Bit auf der Übertragungsstrecke verfälscht wurde. Die Anzahl der verfälschten Bits ist eine Größe, die uns bereits formal bekannt ist: Sie ist das Hamming-Gewicht  $\omega(e)$ . Manchmal wird  $e$  auch als *Korrekturvektor* bezeichnet, da wir die ursprünglich gesendete Nachricht durch die Addition von  $e$  auf die empfangene Bitsequenz rekonstruieren können. Dem Empfänger nützt diese Erkenntnis erst einmal nichts; ihm ist zunächst weder  $v$  noch  $e$  bekannt.

Um das gesendete Codewort  $v$  zu rekonstruieren, muss ein Kanaldecoder die Lösungsmenge der Gleichung (6.11) bestimmen und unter allen in Frage kommenden Vektoren für  $e$  denjenigen auswählen, der das wahrscheinlichste Fehlerszenario beschreibt. In den meisten Fällen werden für diesen Zweck *Maximum-Likelihood-Decoder* eingesetzt, die das Fehlerszenario mit der geringsten Anzahl an gekippten Bits als das wahrscheinlichste ansehen. Das bedeutet, dass ein solcher Decoder aus den Lösungen von (6.11) denjenigen Vektor  $e$  auswählt, dessen Hamming-Gewicht am geringsten ist.

Wie lässt sich ein Maximum-Likelihood-Decoder konstruieren? Eine naheliegende Möglichkeit besteht darin, die zu bestimmenden Fehlervektoren  $e$  in einer Tabelle zu speichern, in der alle Bitsequenzen aufgelistet sind, die den Empfänger erreichen können. Tabelle 6.1 zeigt, wie diese für den zyklischen Hamming-Code aussieht, den wir in der mittleren Spalte in Abbildung 3.30 kennengelernt haben. In der ersten Spalte der Korrekturtabelle sind alle Bitsequenzen der Länge 7 aufgelistet, und in der mittleren Spalte ist für jede Sequenz derjenige Vektor mit dem kleinsten Hamming-Gewicht vermerkt, mit dem der Fehler korrigiert werden kann. Die dritte Spalte wollen wir momentan noch ignorieren, wohlwissend, dass sie gleich sehr wichtig werden wird.

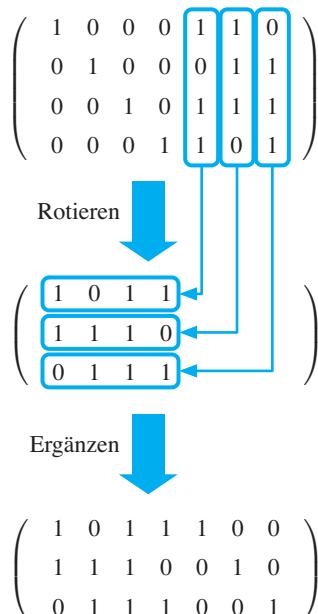
	$w$	$e$	$w \cdot H^T$		$w$	$e$	$w \cdot H^T$		$w$	$e$	$w \cdot H^T$
0	0000000	0000000	000	43	0101011	0001000	101	86	1010110	0010000	111
1	0000001	0000001	001	44	0101100	0000010	010	87	1010111	1000000	110
2	0000010	0000010	010	45	0101101	0100000	011	88	1011000	0000100	100
3	0000011	0100000	011	46	0101110	0000000	000	89	1011001	0001000	101
4	0000100	0000100	100	47	0101111	0000001	001	90	1011010	1000000	110
5	0000101	0001000	101	48	0110000	0000100	100	91	1011011	0010000	111
6	0000110	1000000	110	49	0110001	0001000	101	92	1011100	0000000	000
7	0000111	0010000	111	50	0110010	1000000	110	93	1011101	0000001	001
8	0001000	0001000	101	51	0110011	0010000	111	94	1011110	0000010	010
9	0001001	0000100	100	52	0110100	0000000	000	95	1011111	0100000	011
10	0001010	0010000	111	53	0110101	0000001	001	96	1100000	0001000	101
11	0001011	1000000	110	54	0110110	0000010	010	97	1100001	0000100	100
12	0001100	0000001	001	55	0110111	0100000	011	98	1100010	0010000	111
13	0001101	0000000	000	56	0111000	0000001	001	99	1100011	1000000	110
14	0001110	0100000	011	57	0111001	0000000	000	100	1100100	0000001	001
15	0001111	0000010	010	58	0111010	0100000	011	101	1100101	0000000	000
16	0010000	0010000	111	59	0111011	0000010	010	102	1100110	0100000	011
17	0010001	1000000	110	60	0111100	0001000	101	103	1100111	0000010	010
18	0010010	0001000	101	61	0111101	0000100	100	104	1101000	0000000	000
19	0010011	0000100	100	62	0111110	0010000	111	105	1101001	0000001	001
20	0010100	0100000	011	63	0111111	1000000	110	106	1101010	0000010	010
21	0010101	0000010	010	64	1000000	1000000	110	107	1101011	0100000	011
22	0010110	0000001	001	65	1000001	0010000	111	108	1101100	0000100	100
23	0010111	0000000	000	66	1000010	0000100	100	109	1101101	0001000	101
24	0011000	0000010	010	67	1000011	0001000	101	110	1101110	1000000	110
25	0011001	0100000	011	68	1000100	0000010	010	111	1101111	0010000	111
26	0011010	0000000	000	69	1000101	0100000	011	112	1110000	0000010	010
27	0011011	0000001	001	70	1000110	0000000	000	113	1110001	0100000	011
28	0011100	1000000	110	71	1000111	0000001	001	114	1110010	0000000	000
29	0011101	0010000	111	72	1001000	0100000	011	115	1110011	0000001	001
30	0011110	0000100	100	73	1001001	0000010	010	116	1110100	1000000	110
31	0011111	0001000	101	74	1001010	0000001	001	117	1110101	0010000	111
32	0100000	0100000	011	75	1001011	0000000	000	118	1110110	0000100	100
33	0100001	0000010	010	76	1001100	0010000	111	119	1110111	0001000	101
34	0100010	0000001	001	77	1001101	1000000	110	120	1111000	0010000	111
35	0100011	0000000	000	78	1001110	0001000	101	121	1111001	1000000	110
36	0100100	0010000	111	79	1001111	0000100	100	122	1111010	0001000	101
37	0100101	1000000	110	80	1010000	0000001	001	123	1111011	0000100	100
38	0100110	0001000	101	81	1010001	0000000	000	124	1111100	0100000	011
39	0100111	0000100	100	82	1010010	0100000	011	125	1111101	0000010	010
40	0101000	1000000	110	83	1010011	0000010	010	126	1111110	0000001	001
41	0101001	0010000	111	84	1010100	0001000	101	127	1111111	0000000	000
42	0101010	0000100	100	85	1010101	0000100	100				

Tab. 6.1: Fehlerkorrekturtabelle für den zyklischen [7,4]-Hamming-Code

Um eine empfangene Bitsequenz  $w$  zu überprüfen und gegebenenfalls zu korrigieren, muss der Empfänger jetzt lediglich in der zu  $w$  gehörenden Tabellenzeile den Fehlervektor  $e$  auslesen. Ist  $e$  der Nullvektor, so wurde das Codewort fehlerfrei übertragen. Ist  $e \neq 0$ , so kann die ursprünglich gesendete Nachricht durch die Addition von  $e$  wiederhergestellt werden.

In der geschilderten Form ist die Decodierung zwar konzeptionell einsichtig, aufgrund der großen Tabelle aber sehr umständlich. Glücklicherweise können wir die Decodierung deutlich vereinfachen, wenn wir uns an eine zentrale Eigenschaft von Untervektorräumen erinnern, die wir in Abschnitt 2.5.2 erarbeitet haben. Von dort wissen wir, dass zu jedem Untervektorraum  $V$  ein Orthogonalraum existiert, dessen Generatormatrix als *Kontrollmatrix* benutzt werden kann. Diese Matrix, wir nennen sie  $H$ , hat die Eigenschaft, dass ein Vektor  $w$  genau dann zu  $V$  gehört, wenn die Multiplikation von  $H$  mit dem transponierten Vektor  $w^T$  den Nullvektor hervorbringt:

$$w \in V \Leftrightarrow H \cdot w^T = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} \quad (6.12)$$



Anstatt den Vektor  $w$  zu transponieren, können wir genauso gut  $H$  transponieren und  $w$  von links mit der Matrix multiplizieren. (6.12) liest sich dann wie folgt:

$$w \in V \Leftrightarrow w \cdot H^T = (0 \dots 0) \quad (6.13)$$

Wie sich die Kontrollmatrix berechnen lässt, wissen wir ebenfalls aus Abschnitt 2.5.2. Zunächst wird die Generatormatrix in die reduzierte Form gebracht. Anschließend werden die rechts stehenden Spalten als Zeilenvektoren aufgeschrieben und das Ergebnis um die Einheitsmatrix ergänzt. Abbildung 6.20 zeigt, wie diese Transformation für den zyklischen [7,4]-Hamming-Code konkret aussieht.

Der Schlüssel für unser Vorhaben ist der Vektor  $w \cdot H^T$ , der in (6.13) bereits ein erstes Mal aufgetaucht ist. In der Informations- und Codierungstheorie trägt dieser Vektor einen speziellen Namen: Er wird als das *Syndrom* von  $w$  bezeichnet.



### Definition 6.7 (Syndrom)

Es sei  $H$  die Kontrollmatrix eines linearen Codes.

Der Vektor  $w \cdot H^T$  heißt *Syndrom* von  $w$ .

**Abb. 6.20:** Berechnung der Kontrollmatrix für den zyklischen [7,4]-Hamming-Code

Für uns entscheidend ist die Tatsache, dass die Bildung des Syndroms zu einer Dimensionsreduktion führt. Betrachten wir beispielsweise den zyklischen [7,4]-Hamming-Code, so ist  $w$  einer von insgesamt 128 verschiedenen Vektoren, der durch die Multiplikation mit der Kontrollmatrix auf eines der folgenden 8 Syndrome abgebildet wird:

$$(000), (001), (010), (011), (100), (101), (110), (111)$$

Welche Syndrome im Einzelnen entstehen, ist in der dritten Spalte von Tabelle 6.1 vermerkt. Die Bedeutung dieser Spalte hatten wir weiter oben noch offengelassen.

Wir wollen uns an dieser Stelle genauer überlegen, was das Syndrom  $w \cdot H^T$  über den Vektor  $w$  verrät. Aus (6.13) wissen wir, dass  $w$  genau dann korrekt übertragen wurde, wenn sein Syndrom der Nullvektor ist. Wir wollen nun untersuchen, welche Eigenschaften das Syndrom aufweist, wenn das Codewort auf der Übertragungsstrecke verfälscht wurde. In diesem Fall wird die Kontrollmatrix mit einem Vektor multipliziert, der kein Codewort ist.

Für unsere Untersuchungen nehmen wir an,  $w_1$  und  $w_2$  seien zwei Vektoren, die das gleiche Syndrom erzeugen, es gilt also

$$w_1 \cdot H^T = w_2 \cdot H^T \quad (6.14)$$

Ferner sei  $e$  der Fehlervektor von  $w_1$ . Addieren wir  $e$  und  $w_1$ , so entsteht ein Codewort, das nach (6.13) die Beziehung

$$(w_1 + e) \cdot H^T = \mathbf{0} \quad (6.15)$$

erfüllt. Was wird passieren, wenn wir versuchen,  $w_2$  mit dem Vektor  $e$  zu korrigieren? Die folgende Rechnung gibt die Antwort:

$$\begin{aligned} (w_2 + e) \cdot H^T &= w_2 \cdot H^T + e \cdot H^T \\ &\stackrel{(6.14)}{=} w_1 \cdot H^T + e \cdot H^T \\ &\stackrel{(6.15)}{=} (w_1 + e) \cdot H^T \\ &= \mathbf{0} \end{aligned}$$

Die Rechnung macht deutlich, dass der Fehlervektor  $e$  nicht nur  $w_1$ , sondern auch  $w_2$  korrigiert. Damit haben wir ein wichtiges Ergebnis erzielt: Sind die Syndrome zweier Vektoren gleich, so ist auch die Menge ihrer Fehlervektoren gleich.

An Tabelle 6.1 können wir dieses Phänomen gut beobachten, wenn wir eine Umsortierung vornehmen und die Einträge, wie es in Tabelle 6.2 gezeigt ist, entsprechend ihren Syndromen gruppieren. Das Ergebnis



In der Literatur wird das Syndrom eines Vektors  $w$  häufig in der Form  $H \cdot w^T$  angegeben und nicht, wie hier, in der Form  $w \cdot H^T$ . Diese alternative Definition unterscheidet sich von unserer nur marginal. Als Ergebnis der Multiplikation entsteht dann ein Spaltenvektor und nicht, wie bei uns, ein Zeilenvektor. Die Vektorelemente sind in beiden Fällen gleich. Da wir die Syndrome zumeist in Tabellen angeben, ist die Zeilenschreibweise für uns übersichtlicher, und genau aus diesem Grund haben wir sie gewählt.

Übrigens: Der Begriff *Syndrom* stammt aus der Medizin. Schlagen wir in einem der bekannten klinischen Wörterbücher nach, so finden wir dort die folgende Definition:

„**Syndrom** (gr. σύνδρομος mitlaufend, begleitend) n: (engl.) *syndrome*; Gruppe von Krankheitszeichen, die für ein bestimmtes Krankheitsbild [...] charakteristisch sind.“

Pschyrembel [73]

In Anbetracht der Rolle, die das Syndrom in der Codierungstheorie spielt, ist der Begriff wahrhaft treffend gewählt.

	$w$	$e$	$w \cdot H^T$		$w$	$e$	$w \cdot H^T$		$w$	$e$	$w \cdot H^T$
0	0000000	0000000	000	94	1011110	0000010	010	49	0110001	0001000	101
13	0001101	0000000	000	103	1100111	0000010	010	60	0111100	0001000	101
23	0010111	0000000	000	106	1101010	0000010	010	67	1000011	0001000	101
26	0011010	0000000	000	112	1110000	0000010	010	78	1001110	0001000	101
35	0100011	0000000	000	125	1111101	0000010	010	84	1010100	0001000	101
46	0101110	0000000	000	3	0000011	0100000	011	89	1011001	0001000	101
52	0110100	0000000	000	14	0001110	0100000	011	96	1100000	0001000	101
57	0111001	0000000	000	20	0010100	0100000	011	109	1101101	0001000	101
70	1000110	0000000	000	25	0011001	0100000	011	119	1110111	0001000	101
75	1001011	0000000	000	32	0100000	0100000	011	122	1111010	0001000	101
81	1010001	0000000	000	45	0101101	0100000	011	6	0000110	1000000	110
92	1011100	0000000	000	55	0110111	0100000	011	11	0001011	1000000	110
101	1100101	0000000	000	58	0111010	0100000	011	17	0010001	1000000	110
104	1101000	0000000	000	69	1000101	0100000	011	28	0011100	1000000	110
114	1110010	0000000	000	72	1001000	0100000	011	37	0100101	1000000	110
127	1111111	0000000	000	82	1010010	0100000	011	40	0101000	1000000	110
1	0000001	0000001	001	95	1011111	0100000	011	50	0110010	1000000	110
12	0001100	0000001	001	102	1100110	0100000	011	63	0111111	1000000	110
22	0010110	0000001	001	107	1101011	0100000	011	64	1000000	1000000	110
27	0011011	0000001	001	113	1110001	0100000	011	77	1001101	1000000	110
34	0100010	0000001	001	124	1111100	0100000	011	87	1010111	1000000	110
47	0101111	0000001	001	4	0000100	0000100	100	90	1011010	1000000	110
53	0110101	0000001	001	9	0001001	0000100	100	99	1100011	1000000	110
56	0111000	0000001	001	19	0010011	0000100	100	110	1101110	1000000	110
71	1000111	0000001	001	30	0011110	0000100	100	116	1110100	1000000	110
74	1001010	0000001	001	39	0100111	0000100	100	121	1111001	1000000	110
80	1010000	0000001	001	42	0101010	0000100	100	7	0000111	0010000	111
93	1011101	0000001	001	48	0110000	0000100	100	10	0001010	0010000	111
100	1100100	0000001	001	61	0111101	0000100	100	16	0010000	0010000	111
105	1101001	0000001	001	66	1000010	0000100	100	29	0011101	0010000	111
115	1110011	0000001	001	79	1001111	0000100	100	36	0100100	0010000	111
126	1111110	0000001	001	85	1010101	0000100	100	41	0101001	0010000	111
2	0000010	0000010	010	88	1011000	0000100	100	51	0110011	0010000	111
15	0001111	0000010	010	97	1100001	0000100	100	62	0111110	0010000	111
21	0010101	0000010	010	108	1101100	0000100	100	65	1000001	0010000	111
24	0011000	0000010	010	118	1110110	0000100	100	76	1001100	0010000	111
33	0100001	0000010	010	123	1111011	0000100	100	86	1010110	0010000	111
44	0101100	0000010	010	5	0000101	0001000	101	91	1011011	0010000	111
54	0110110	0000010	010	8	0001000	0001000	101	98	1100010	0010000	111
59	0111011	0000010	010	18	0010010	0001000	101	111	1101111	0010000	111
68	1000100	0000010	010	31	0011111	0001000	101	117	1110101	0010000	111
73	1001001	0000010	010	38	0100110	0001000	101	120	1111000	0010000	111
83	1010011	0000010	010	43	0101011	0001000	101				

Tab. 6.2: Sortierte Fehlerkorrekturtabelle für den zyklischen [7,4]-Hamming-Code

macht deutlich, dass es ausreicht, das Syndrom zu kennen, um den richtigen Fehlervektor abzuleiten. Anstatt für jeden der 128 möglichen Vektoren  $w$  einen eigenen Tabelleneintrag vorzuhalten, können wir uns damit begnügen, für jedes der 8 möglichen Syndrome einen Eintrag anzulegen. Als Ergebnis dieser Reduktion entsteht Tabelle 6.3.

Abbildung 6.21 demonstriert das Prinzip der Syndromdecodierung anhand eines konkreten Beispiels. In dem abgebildeten Szenario wurde die Nachricht 0100011 gesendet und an der vierten Bitposition verfälscht. Auf der Empfängerseite wird durch die Multiplikation der Kontrollmatrix mit der empfangenen Bitsequenz zunächst das Syndrom berechnet. Anschließend wird in der Syndromtabelle der Fehlervektor  $e$  nachgeschlagen und die ursprünglich gesendete Nachricht durch die Addition von  $e$  wiederhergestellt.

Im Gegensatz zu unserem ersten Ansatz, der mit einer großen Tabelle arbeitet, erfordert der neue Ansatz ein wenig Zusatzaufwand, da wir den Fehlervektor erst auslesen können, nachdem das Syndrom bestimmt wurde.

In  $\mathbb{Z}_2$  lässt sich die Matrixmultiplikation mit einem Hardwareschaltnetz ausführen, das die Vektorelemente mit einer Reihe von XOR-Gattern verknüpft. Um dieses Schaltnetz für unser Beispiel zu konstruieren, müssen wir den Ausdruck  $w \cdot H^T$  lediglich in eine geeignete Schreibweise bringen. Ist  $w = (w_0, \dots, w_6)$ , so können wir die Syndromberechnung

$$w \cdot H^T = w \cdot \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}^T = w \cdot \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

ganz einfach so aufschreiben:

$$w \cdot H^T = (w_0 \oplus w_2 \oplus w_3 \oplus w_4, w_0 \oplus w_1 \oplus w_2 \oplus w_5, w_1 \oplus w_2 \oplus w_3 \oplus w_6)$$

Aus dieser Darstellung können wir unmittelbar das Schaltnetz ableiten, das in Abbildung 6.22 zu sehen ist.

An dieser Stelle kommen wir auf den Hamming-Code zurück, der auf Seite 192 eingeführt wurde und durch die folgende Generatormatrix definiert war:

	$w \cdot H^T$	$e$
0	000	0000000
1	001	0000001
2	010	0000010
3	011	0100000
4	100	0000100
5	101	0001000
6	110	1000000
7	111	0010000

Tab. 6.3: Reduzierte Fehlerkorrekturtabelle (Syndromtabelle) für den zyklischen [7,4]-Hamming-Code

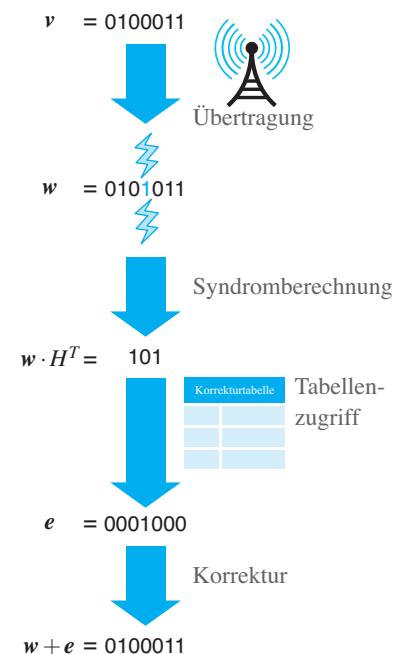
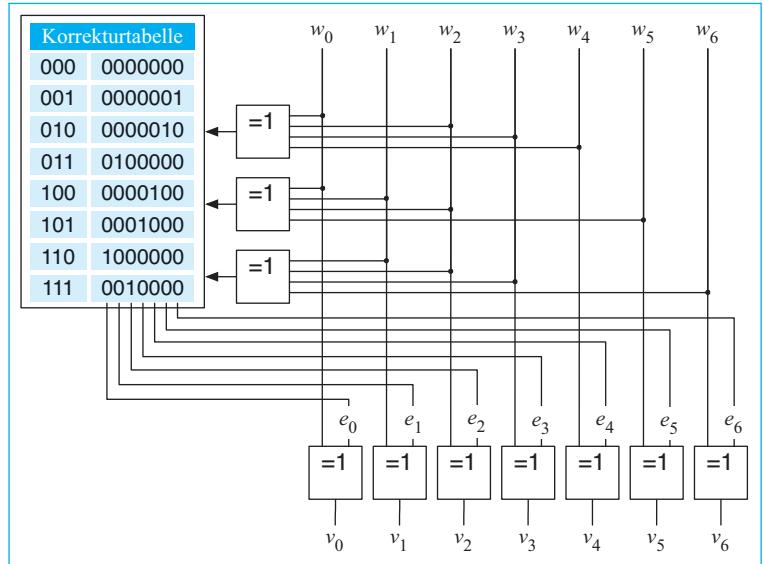


Abb. 6.21: Syndromdecodierung am Beispiel des Codeworts 0100011 und einem Fehler an der vierten Bitposition



**Abb. 6.22:** Syndromdecoder für den zyklischen [7,4]-Hamming-Code. Die drei vertikal angeordneten XOR-Gatter berechnen das Syndrom  $w \cdot H^T$ . Dieses wird als Index verwendet und aus der Korrekturtabelle der Fehlervektor  $e$  ausgelesen. Die horizontal angeordneten XOR-Gatter verknüpfen das empfangene Codewort  $w$  mit dem Fehlervektor  $e$  und stellen auf diese Weise das ursprünglich gesendete Codewort wieder her.

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Zeile 2 und 3 auf Zeile 1 addieren  
Zeile 3 auf Zeile 2 addieren

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Zeilen 1 und 2 auf Zeile 3 addieren

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

**Abb. 6.23:** Umformung der Kontrollmatrix

$$G = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \quad (6.16)$$

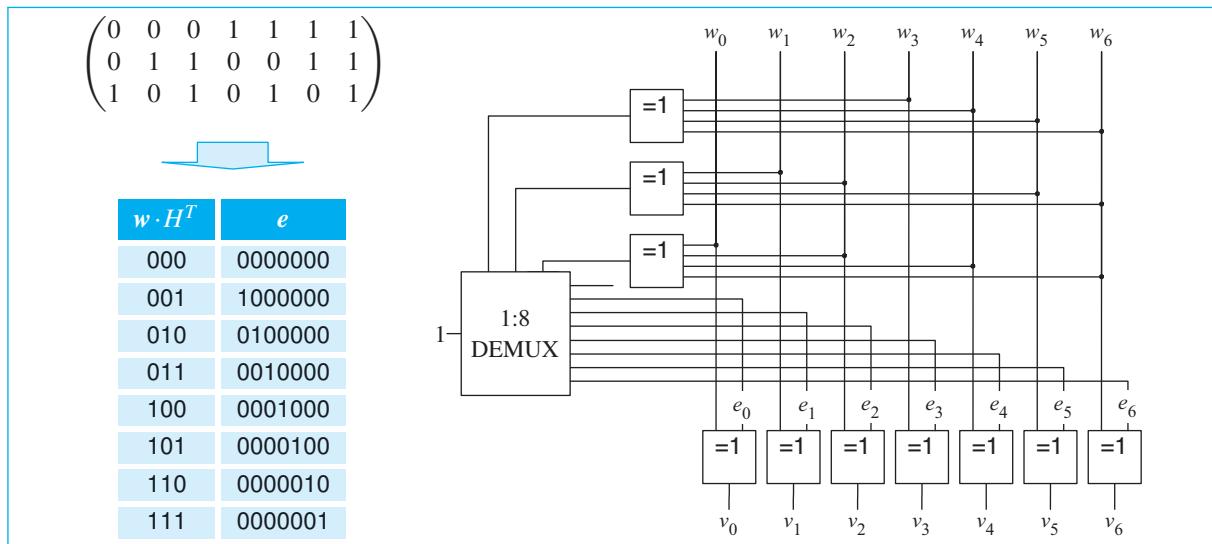
Eine Kontrollmatrix für diesen Code ist folgende:

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \quad (6.17)$$

Auf Seite 192 wurde erwähnt, dass diesem Code eine ganz besondere Bedeutung zukommt, und wir sind nun in der Lage, sie aufzudecken. Hierzu bringen wir die Kontrollmatrix mit den in Abbildung 6.23 dargestellten Zeilenumformungen in die folgende Form:

$$H = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \quad (6.18)$$

Benutzen wir diese Kontrollmatrix für die Bildung der Syndrome, so erhalten wir als Ergebnis die Tabelle, die in Abbildung 6.24 links zu sehen ist. Ein direkter Vergleich mit den Fehlervektoren zeigt, dass die Syndrome direkt Auskunft über die Bitposition des Übertragungsfehlers geben: Das Bitmuster des Syndroms ist die Binärdarstellung der



**Abb. 6.24:** Die abgebildete Kontrollmatrix zeichnet sich durch die besondere Eigenschaft aus, dass wir das Syndrom direkt als die Position des fehlerhaften Bits interpretieren können. Ein einfacher Demultiplexer reicht deshalb aus, um das Korrekturbit an die richtige Stelle zu schieben.

Fehlerposition. Das bedeutet, dass wir in diesem Fall gänzlich auf die Speicherung einer Syndromtabelle verzichten und die Logik für die Berechnung des Fehlervektors durch einen einfachen Demultiplexer ersetzen können. Die vereinfachte Schaltung ist in Abbildung 6.24 rechts abgedruckt.

Der verbaute Demultiplexer-Baustein verfügt über acht Datenausgänge und einen Dateneingang, der permanent mit dem Signal 1 beschaltet ist. Ferner existieren drei Steuerleitungen, die darüber entscheiden, auf welchen der acht Ausgänge das Eingangssignal durchgeschaltet wird. Die Steuerkombination 000 aktiviert den ersten Ausgang, 001 den zweiten, 010 den dritten und so fort. Da das Syndrom für den betrachteten Hamming-Code mit der Position des Übertragungsfehlers übereinstimmt, erzeugt der Demultiplexer an seinen Ausgängen den Fehlervektor  $e$ . Dieser wird eins zu eins an die XOR-Gatter weitergegeben, die durch das Kippen des fehlerhaften Bits die ursprünglich gesendete Nachricht wiederherstellen.

Dass der oberste Ausgang des Demultiplexers nicht verdrahtet wurde, ist kein Fehler. Er würde aktiv werden, wenn das Syndrom der Nullvektor ist. In diesem Fall ist die empfangene Bitsequenz korrekt, sodass sämtliche XOR-Gatter mit einer 0 beschaltet werden müssen.



Richard Wesley Hamming wurde am 11. Februar 1915 in Chicago geboren, wo er nicht nur seine Kindheit und Schulzeit verbrachte, sondern auch bis zum Bachelorabschluss seines Mathematikstudiums wohnhaft blieb. Das Masterstudium absolvierte er an der University of Nebraska. 1939 wechselte er an die University of Illinois at Urbana-Champaign, die ihm drei Jahre später die Doktorwürde verlieh. Im selben Jahr heiratete Hamming seine Frau und nahm eine Stelle als wissenschaftlicher Assistent an, zunächst an der University of Illinois und danach an der University of Louisville.

1945 zog Hamming nach Los Alamos und stieg dort, auf Anraten eines engen Freundes, in das *Manhattan-Projekt* ein. Im Rahmen des Atombombenprogramms der US-Regierung beschäftigte er sich mit computergestützten Simulationen der Kernspaltung. Hamming fand in Los Alamos ein hochkarätiges wissenschaftliches Umfeld vor und wurde dort unter anderem mit Richard Feynman, Enrico Fermi und Robert Oppenheimer bekannt. Der Kontakt mit den ersten digitalen Rechenmaschinen war für Hamming ein Wendepunkt in seiner wissenschaftlichen Ausrichtung und lenkte sein Interesse auf informations- und codierungstheoretische Fragestellungen.

Nach dem Ende des zweiten Weltkriegs wechselte Hamming an die Bell Labs in Murray Hill, wo er mit Claude Shannon sein Büro teilte. An den Bell Labs feierte Hamming

seine größten wissenschaftlichen Erfolge. Den Höhepunkt markiert seine 1950 erschienene Arbeit mit dem Titel „Error Detecting and Error Correcting Codes“, die im „Bell System Technical Journal“ erschien (Abbildung 6.25). In dieser Arbeit begründete Hamming nicht nur die in der Codierungstheorie so grundlegenden Distanzbegriffe, sondern auch jene Codes, die heute seinen Namen tragen.

Hamming war bereits 61 Jahre alt, als er die Bell Labs 1976 verließ und eine Professur an der Naval Postgraduate School im kalifornischen Monterey annahm. Dort zog er sich weitgehend aus der Forschung zurück und widmete sich intensiv der Lehre. In Monterey arbeitete er noch über 20 Jahre bis zu seiner Emeritierung im Jahr 1997. Ein langes Leben im verdienten Ruhestand war Hamming nicht mehr vergönnt. Bereits ein Jahr später erlitt er einen Herzinfarkt und verstarb am 7. Januar 1998 in seiner kalifornischen Wahlheimat.

Hamming wurde für sein Lebenswerk mehrfach geehrt. Im Jahr 1968 erhielt er den Turing-Award, der jährlich von der Association for Computing Machinery, kurz ACM, für ausgezeichnete Leistungen in der Informatik verliehen wird. Im Bereich der Informations- und Codierungstheorie wird seit dem Jahr 1988 eine eigene Medaille vergeben, die unseren Protagonisten in doppelter Hinsicht ehrt. Hamming war nicht nur der erste Preisträger, sondern gleichzeitig der Namensgeber der vom Institute of Electrical and Electronics Engineers (IEEE) ausgerufenen *Richard-W.-Hamming-Medaille*.

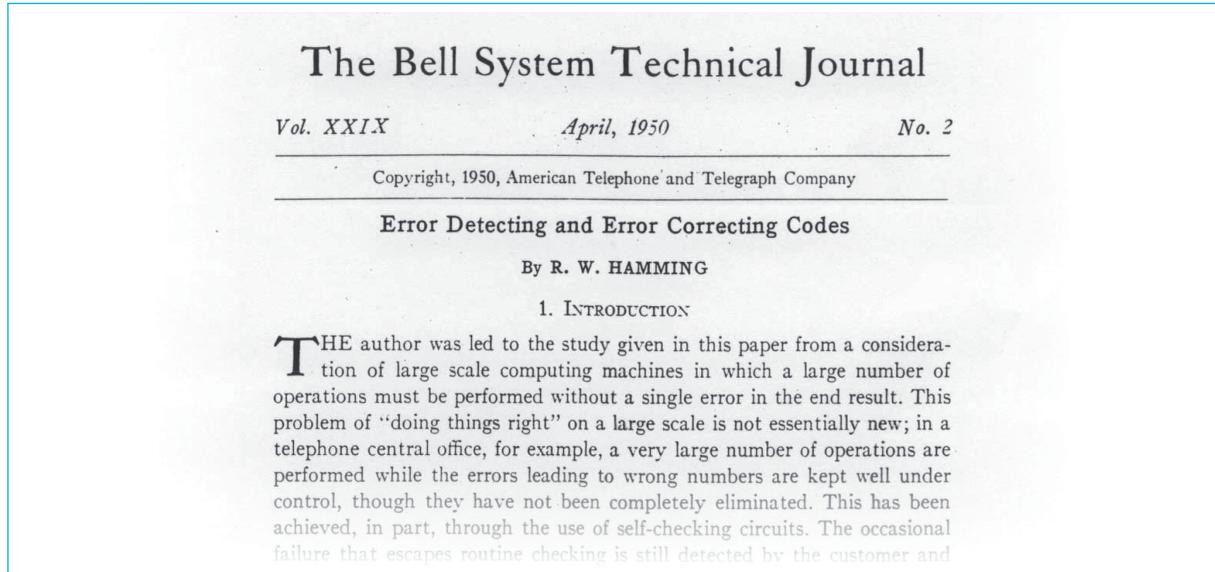
Die Eigenschaft der Syndrome, unmittelbar die Position eines auftretenden Fehlers zu verraten, machen den betrachteten Hamming-Code einzigartig. Sie ist der Grund, warum unter den vielen Möglichkeiten, einen Hamming-Code zu definieren, zumeist diese gewählt wird.

Im nächsten Abschnitt werden wir uns den Aufbau dieses speziellen Hamming-Codes noch ein wenig genauer ansehen und im gleichen Atemzug auf beliebige Bitbreiten verallgemeinern.

## 6.4.2 Hamming-Codes

Gerade haben wir gesehen, dass der Kontrollmatrix

$$H = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$



**Abb. 6.25:** Ein Meilenstein der Codierungstheorie: Richard Hammings Publikation aus dem Jahr 1950

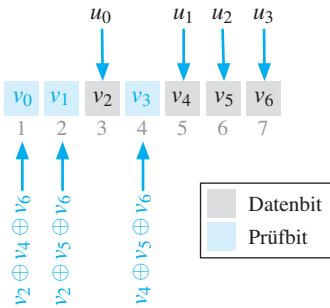
unter den vielen Möglichkeiten, einen Hamming-Code zu definieren, eine ganz besondere Bedeutung zukommt: Sie besitzt die Eigenschaft, dass das Syndrom eines Vektors, in dem ein einziges Bit verfälscht wurde, mit der Position des defekten Bits identisch ist. Wir wollen jetzt auf einem intuitiven Weg ergründen, warum die Kontrollmatrix diese Eigenschaft besitzt. Der Aufwand wird sich lohnen: Am Ende dieses Abschnitts werden Sie nicht nur den Aufbau dieser Matrix verstehen, sondern gleichsam in der Lage sein, Kontrollmatrizen mit dieser Eigenschaft direkt niederzuschreiben.

Wir beginnen damit, die Gleichung  $H \cdot v^T = \mathbf{0}$ , die jedes Hamming-Codewort erfüllen muss, etwas ausführlich aufzuschreiben. Aus

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_6 \end{pmatrix} = \begin{pmatrix} v_3 \oplus v_4 \oplus v_5 \oplus v_6 \\ v_1 \oplus v_2 \oplus v_5 \oplus v_6 \\ v_0 \oplus v_2 \oplus v_4 \oplus v_6 \end{pmatrix}$$

folgt, dass  $H \cdot v^T = \mathbf{0}$  das Gleiche ist wie

$$\begin{pmatrix} v_3 \oplus v_4 \oplus v_5 \oplus v_6 \\ v_1 \oplus v_2 \oplus v_5 \oplus v_6 \\ v_0 \oplus v_2 \oplus v_4 \oplus v_6 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$



**Abb. 6.26:** Aufbau eines Hamming-Codeworts der Länge 7. An den Positionen, die einer Zweierpotenz entsprechen, befinden sich die Prüfbits. Die Datenbits liegen dazwischen.

Etwas übersichtlicher können wir dies in der Form

$$v_3 \oplus v_4 \oplus v_5 \oplus v_6 = 0 \quad (6.19)$$

$$v_1 \oplus v_2 \oplus v_5 \oplus v_6 = 0 \quad (6.20)$$

$$v_0 \oplus v_2 \oplus v_4 \oplus v_6 = 0 \quad (6.21)$$

aufschreiben, was direkt zu den folgenden Beziehungen führt:

$$v_3 = v_4 \oplus v_5 \oplus v_6 \quad (6.22)$$

$$v_1 = v_2 \oplus v_5 \oplus v_6 \quad (6.23)$$

$$v_0 = v_2 \oplus v_4 \oplus v_6 \quad (6.24)$$

Aus diesen drei Gleichungen können wir eine konkrete Codierungsvorschrift ableiten, die Nachrichten der Länge 4 auf Hamming-Codewörter der Länge 7 abbildet. Hierzu interpretieren wir die Bits auf der rechten Seite ( $v_2, v_4, v_5, v_6$ ) ganz einfach als vier Nachrichtenbits ( $u_0, u_1, u_2, u_3$ ) und die Bits auf der linken Seite ( $v_0, v_1, v_3$ ) als Prüfbits. Um eine Nachricht zu codieren, müssen jetzt lediglich die drei Prüfbits berechnet und, wie in Abbildung 6.26 gezeigt, an den Zweierpotenzstellen zwischen den Nachrichtenbits eingefügt werden.

Was es mit den Gleichungen (6.22) bis (6.24) tatsächlich auf sich hat, wird deutlich, wenn wir die Berechnung in einer Tabelle ausführen, wie sie in Abbildung 6.27 dargestellt ist. In der ersten Spalte werden die Nachrichtenbits von oben nach unten aufgelistet, und links davon die Positionen vermerkt, an denen die Nachrichtenbits später innerhalb des Codeworts auftauchen. Wie oben bereits erwähnt wurde, befinden sich die Prüfbits an den Zweierpotenzpositionen, sodass die Werte 1, 2, 4 etc. nicht in der Positionsspalte auftauchen.

Die restlichen Spalten dienen zur Berechnung der Prüfbits. Beachten Sie, dass die Bits in umgekehrter Reihenfolge angeordnet sind, sodass die am weitesten links stehende Spalte demjenigen Prüfbits entspricht, das im Codewort am weitesten rechts steht. In den Spalten sind genau jene Felder farblich markiert, die für die Berechnung der Prüfbits benötigt werden. Um diese zu bestimmen, müssen wir lediglich die Nachrichtenbits Zeile für Zeile in die farblich markierten Felder eintragen und anschließend für jede Spalte ein Paritätsbit berechnen. Die Paritätsbits sind die Prüfbits: Schieben wir sie an die Zweierpotenzpositionen, so erhalten wir das Codewort.

Abbildung 6.28 demonstriert die Codierung am Beispiel der Nachricht 1001. Im ersten Schritt werden die Nachrichtenbits in die mit  $u$  markierte Spalte geschrieben und rechts in die betreffenden Prüfbitfelder

kopiert. Anschließend wird durch die XOR-Verknüpfung der eingetragenen Werte in jeder Spalte ein Paritätsbit erzeugt. Fügen wir diese Bits jetzt in umgekehrter Reihenfolge an den Zweierpotenzstellen zwischen den Nachrichtenbits ein, so entsteht das Hamming-Codewort; in unserem Beispiel ist dies das Codewort 0011001.

Abbildung 6.29 zeigt, wie eine Hamming-codierte Bitsequenz überprüft und gegebenenfalls korrigiert werden kann. Hierzu baut der Empfänger zunächst die gleiche Tabelle auf wie der Sender und berechnet daraus eigene Prüfbits. Diese werden anschließend mit den empfangenen Prüfbits XOR-verknüpft. Sind die berechneten Werte alle gleich 0, stimmen die berechneten und die empfangenen Prüfbits also überein, so ist die Übertragung fehlerfrei verlaufen. Ist das Ergebnis ungleich 0, so muss der Empfänger das Bitmuster lediglich in eine Dezimalzahl konvertieren, um die Position des defekten Bits zu erhalten.

Dass sich die Position des fehlerhaften Bits derart einfach berechnen lässt, wirkt auf den ersten Blick wie ein fragwürdiger Taschenspielertrick. Auf den zweiten Blick wird aber schnell klar, dass hier in keiner Weise gemogelt wird und wir tatsächlich für jeden möglichen Einzelfehler das richtige Ergebnis erhalten. Verantwortlich ist die Tatsache, dass wir in den Prüfbitspalten genau dort Werte eintragen, wo in der Binärdarstellung der zugeordneten Position eine Eins auftritt.

Wir wollen an dieser Stelle genauer hinsehen und betrachten als Beispiel das Nachrichtenbit in der zweiten Zeile von oben. Dieses Bit erscheint im Codewort an der Position 5. Die Zahl 5 hat die Binärdarstellung 101 und genau diese Felder sind in den Prüfbitspalten markiert:

5 (Dezimal)  $\leftrightarrow$  101 (Binär)  $\leftrightarrow$    (Tabelle)

Tritt nun an Position 5 bei der Übertragung ein Bitfehler auf, so trägt der Empfänger in den gefärbten Feldern einen falschen Wert ein. Da die Tabelle ansonsten exakt jener entspricht, die der Sender erzeugt hat, entsteht genau in jenen Spalten der Empfängertabelle ein unterschiedliches Prüfbit, in denen ein falscher Wert eingetragen wurde. Die Tatsache, dass die Felder in der Hamming-Tabelle das Binärmuster der Bitposition widerspiegeln, sorgt dafür, dass durch die XOR-Verknüpfung die Position des fehlerhaften Bits entsteht.

Das dritte Beispiel in Abbildung 6.29 macht deutlich, dass die Korrektur auch dann funktioniert, wenn bei der Übertragung ein Prüfbit verfälscht wurde. In diesem Fall baut der Empfänger zunächst die gleiche Tabelle auf, die auch der Sender aufgebaut hat, und die berechneten Prüfbits entsprechen exakt jenen, die der Sender berechnet hat. Wurde eines der

<b><math>u</math></b>	<b><math>2^2</math></b>	<b><math>2^1</math></b>	<b><math>2^0</math></b>
3	$v_2$		$v_2$
5	$v_4$	$v_4$	
6	$v_5$	$v_5$	$v_5$
7	$v_6$	$v_6$	$v_6$
	$\oplus$	$\oplus$	$\oplus$
	$v_3$	$v_1$	$v_0$
Positionen der Nachrichtenbits ↑	Nachrichtenbits ↑	Prüfbit an der Position $2^2$ ↑	Prüfbit an der Position $2^1$ ↑
			Prüfbit an der Position $2^0$ ↑

**Abb. 6.27:** Allgemeine Form einer Hamming-Tabelle

$u$	$2^2$	$2^1$	$2^0$
1		1	1
0	0		0
0	0	0	
1	1	1	1
	$\oplus$	$\oplus$	$\oplus$
	1	0	0

  $c(u) = 0011001$

**Abb. 6.28:** Hamming-Codierung am Beispiel der Nachricht 1001

Beispiel 1	Beispiel 2	Beispiel 3																																																																																																			
$0011001 \mapsto 0011\underline{1}01$	$0011001 \mapsto 001100\underline{0}$	$0011001 \mapsto 0010\underline{0}001$																																																																																																			
<ul style="list-style-type: none"> <li>■ Empfangene Datenbits: 1101</li> <li>■ Empfangene Prüfbits: 001</li> </ul>	<ul style="list-style-type: none"> <li>■ Empfangene Datenbits: 1000</li> <li>■ Empfangene Prüfbits: 001</li> </ul>	<ul style="list-style-type: none"> <li>■ Empfangene Datenbits: 1001</li> <li>■ Empfangene Prüfbits: 000</li> </ul>																																																																																																			
<table border="1" style="margin-bottom: 10px;"> <thead> <tr> <th><math>u</math></th><th><math>2^2</math></th><th><math>2^1</math></th><th><math>2^0</math></th></tr> </thead> <tbody> <tr> <td>3</td><td>1</td><td></td><td>1</td></tr> <tr> <td>5</td><td>1</td><td>1</td><td></td></tr> <tr> <td>6</td><td>0</td><td>0</td><td>0</td></tr> <tr> <td>7</td><td>1</td><td>1</td><td>1</td></tr> </tbody> </table> <table border="1" style="margin-bottom: 10px;"> <thead> <tr> <th><math>\oplus</math></th><th><math>\oplus</math></th><th><math>\oplus</math></th></tr> </thead> <tbody> <tr> <td>0</td><td>0</td><td>1</td></tr> <tr> <td>1</td><td>0</td><td>0</td></tr> <tr> <td>=</td><td>1</td><td>0</td><td>1</td></tr> </tbody> </table> <p>👉 Fehler an Position 5</p>	$u$	$2^2$	$2^1$	$2^0$	3	1		1	5	1	1		6	0	0	0	7	1	1	1	$\oplus$	$\oplus$	$\oplus$	0	0	1	1	0	0	=	1	0	1	<table border="1" style="margin-bottom: 10px;"> <thead> <tr> <th><math>u</math></th><th><math>2^2</math></th><th><math>2^1</math></th><th><math>2^0</math></th></tr> </thead> <tbody> <tr> <td>3</td><td>1</td><td></td><td>1</td></tr> <tr> <td>5</td><td>0</td><td>0</td><td></td></tr> <tr> <td>6</td><td>0</td><td>0</td><td>0</td></tr> <tr> <td>7</td><td>0</td><td>0</td><td>0</td></tr> </tbody> </table> <table border="1" style="margin-bottom: 10px;"> <thead> <tr> <th><math>\oplus</math></th><th><math>\oplus</math></th><th><math>\oplus</math></th></tr> </thead> <tbody> <tr> <td>0</td><td>1</td><td>1</td></tr> <tr> <td>1</td><td>0</td><td>0</td></tr> <tr> <td>=</td><td>1</td><td>1</td><td>1</td></tr> </tbody> </table> <p>👉 Fehler an Position 7</p>	$u$	$2^2$	$2^1$	$2^0$	3	1		1	5	0	0		6	0	0	0	7	0	0	0	$\oplus$	$\oplus$	$\oplus$	0	1	1	1	0	0	=	1	1	1	<table border="1" style="margin-bottom: 10px;"> <thead> <tr> <th><math>u</math></th><th><math>2^2</math></th><th><math>2^1</math></th><th><math>2^0</math></th></tr> </thead> <tbody> <tr> <td>3</td><td>1</td><td></td><td>1</td></tr> <tr> <td>5</td><td>0</td><td>0</td><td></td></tr> <tr> <td>6</td><td>0</td><td>0</td><td>0</td></tr> <tr> <td>7</td><td>1</td><td>1</td><td>1</td></tr> </tbody> </table> <table border="1" style="margin-bottom: 10px;"> <thead> <tr> <th><math>\oplus</math></th><th><math>\oplus</math></th><th><math>\oplus</math></th></tr> </thead> <tbody> <tr> <td>1</td><td>0</td><td>0</td></tr> <tr> <td>0</td><td>0</td><td>0</td></tr> <tr> <td>=</td><td>1</td><td>0</td><td>0</td></tr> </tbody> </table> <p>👉 Fehler an Position 4</p>	$u$	$2^2$	$2^1$	$2^0$	3	1		1	5	0	0		6	0	0	0	7	1	1	1	$\oplus$	$\oplus$	$\oplus$	1	0	0	0	0	0	=	1	0	0
$u$	$2^2$	$2^1$	$2^0$																																																																																																		
3	1		1																																																																																																		
5	1	1																																																																																																			
6	0	0	0																																																																																																		
7	1	1	1																																																																																																		
$\oplus$	$\oplus$	$\oplus$																																																																																																			
0	0	1																																																																																																			
1	0	0																																																																																																			
=	1	0	1																																																																																																		
$u$	$2^2$	$2^1$	$2^0$																																																																																																		
3	1		1																																																																																																		
5	0	0																																																																																																			
6	0	0	0																																																																																																		
7	0	0	0																																																																																																		
$\oplus$	$\oplus$	$\oplus$																																																																																																			
0	1	1																																																																																																			
1	0	0																																																																																																			
=	1	1	1																																																																																																		
$u$	$2^2$	$2^1$	$2^0$																																																																																																		
3	1		1																																																																																																		
5	0	0																																																																																																			
6	0	0	0																																																																																																		
7	1	1	1																																																																																																		
$\oplus$	$\oplus$	$\oplus$																																																																																																			
1	0	0																																																																																																			
0	0	0																																																																																																			
=	1	0	0																																																																																																		

**Abb. 6.29:** Korrektur einer Hamming-codierten Nachricht. Zunächst wird die gleiche Tabelle erstellt, die auch der Sender aufgebaut hat. Danach lässt sich die Fehlerposition berechnen, indem die berechneten und die empfangenen Prüfbits XOR-verknüpft werden.

Prüfbits verfälscht, so entsteht durch die XOR-Verknüpfung ein Bitmuster, das an genau einer Stelle eine 1 aufweist. Ein solches Muster ist die Binärdarstellung einer Zweierpotenz, und genau an diesen Positionen haben wir, in weiser Voraussicht, die Prüfbits positioniert. Damit entpuppt sich der Hamming-Code als eine wahrhaft faszinierende Konstruktion!

Auf die gezeigte Weise können wir Hamming-Codes mit beliebigen Codewortlängen konstruieren. Wir beginnen damit, die Nachrichtenbits in einer Spalte niederzuschreiben, und übersetzen die Positionen, an denen die Nachrichtenbits später im Codewort platziert sind, in ihr Binärmuster. Untereinander geschrieben ergeben diese Muster die Hamming-Tabelle, mit der wir die Prüfbits ausrechnen können.

Beispiel 1					Beispiel 2					Beispiel 3				
$\mathbf{u} = 11010111000$					$\mathbf{u} = 10101101101$					$\mathbf{u} = 00011100000$				
$\mathbf{u}$	$2^3$	$2^2$	$2^1$	$2^0$	$\mathbf{u}$	$2^3$	$2^2$	$2^1$	$2^0$	$\mathbf{u}$	$2^3$	$2^2$	$2^1$	$2^0$
3	1		1	1	3	1		1	1	3	0		0	0
5	1		1		5	0		0		5	0		0	0
6	0		0	0	6	1		1	1	6	0		0	0
7	1		1	1	7	0		0	0	7	1		1	1
9	0	0		0	9	1	1		1	9	1	1		1
10	1	1		1	10	1	1		1	10	1	1		1
11	1	1		1	11	0	0		0	11	0	0		0
12	1	1	1		12	1	1	1		12	0	0	0	
13	0	0	0		13	1	1	1		13	0	0	0	0
14	0	0	0	0	14	0	0	0	0	14	0	0	0	0
15	0	0	0	0	15	1	1	1	1	15	0	0	0	0
$\oplus$					$\oplus$					$\oplus$				
<u>1</u>					<u>1</u>					<u>0</u>				

$c(\mathbf{u}) = 00110110111000$

$c(\mathbf{u}) = 001001011101101$

$c(\mathbf{u}) = 000100101100000$

Abb. 6.30: Codewortkonstruktion am Beispiel des [15,11]-Hamming-Codes

Wie die entstehende Tabelle für den [15,11]-Hamming-Code aussieht, ist in Abbildung 6.30 zu sehen. Um die 11 Nachrichtenbits zu codieren, werden insgesamt 4 Prüfbits erzeugt, die dann zu Codewörtern der Länge 15 erweitert werden. Betrachten wir die Verteilung der Felder genauer, so lässt sich auch hier das charakteristische Merkmal der Hamming-Tabellen erkennen: In jeder Zeile gibt es genau dort ein Prüfbitfeld, wo die Binärdarstellung der Bitposition eine 1 enthält.

Die Hamming-Tabelle erfüllt noch einen anderen Zweck. Wir können sie verwenden, um die Kontrollmatrix eines Hamming-Codes abzuleiten. Hierzu brauchen wir lediglich für jede Prüfbitspalte eine Formel der Form (6.22) - (6.24) aufzustellen und diese anschließend in eine Formel der Form (6.19) - (6.21) zurückzuübersetzen. Erzeugen wir danach

$u$	$2^3$	$2^2$	$2^1$	$2^0$								
3	$v_2$			$v_2$	$v_2$							
5	$v_4$		$v_4$		$v_4$							
6	$v_5$		$v_5$	$v_5$								
7	$v_6$		$v_6$	$v_6$	$v_6$							
9	$v_8$	$v_8$				$v_8$						
10	$v_9$	$v_9$				$v_9$						
11	$v_{10}$	$v_{10}$		$v_{10}$	$v_{10}$							
12	$v_{11}$	$v_{11}$	$v_{11}$									
13	$v_{12}$	$v_{12}$	$v_{12}$			$v_{12}$						
14	$v_{13}$	$v_{13}$	$v_{13}$	$v_{13}$								
15	$v_{14}$	$v_{14}$	$v_{14}$	$v_{14}$	$v_{14}$	$v_{14}$						
	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$							
						$v_7$	$v_3$	$v_1$	$v_0$			

$v_7 = v_8 \oplus v_9 \oplus v_{10} \oplus v_{11} \oplus v_{12} \oplus v_{13} \oplus v_{14}$   
 $v_3 = v_4 \oplus v_5 \oplus v_6 \oplus v_{11} \oplus v_{12} \oplus v_{13} \oplus v_{14}$   
 $v_1 = v_2 \oplus v_5 \oplus v_6 \oplus v_9 \oplus v_{10} \oplus v_{13} \oplus v_{14}$   
 $v_0 = v_2 \oplus v_4 \oplus v_6 \oplus v_8 \oplus v_{10} \oplus v_{12} \oplus v_{14}$



$v_7 \oplus v_8 \oplus v_9 \oplus v_{10} \oplus v_{11} \oplus v_{12} \oplus v_{13} \oplus v_{14} = 0$   
 $v_3 \oplus v_4 \oplus v_5 \oplus v_6 \oplus v_{11} \oplus v_{12} \oplus v_{13} \oplus v_{14} = 0$   
 $v_1 \oplus v_2 \oplus v_5 \oplus v_6 \oplus v_9 \oplus v_{10} \oplus v_{13} \oplus v_{14} = 0$   
 $v_0 \oplus v_2 \oplus v_4 \oplus v_6 \oplus v_8 \oplus v_{10} \oplus v_{12} \oplus v_{14} = 0$



$H_{[15,11]} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$

**Abb. 6.31:** Aus der Hamming-Tabelle lässt sich mit wenig Aufwand die Kontrollmatrix für den [15,11]-Hamming-Code ableiten.

$$G = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

**Abb. 6.32:** Generatormatrix des [15,11]-Hamming-Codes

für jede enthaltene Variable eine 1 und für jede nicht enthaltene Variable eine 0, so entsteht die Kontrollmatrix des Hamming-Codes. Abbildung 6.31 zeigt, wie sich die Kontrollmatrix des [15,11]-Hamming-Codes auf die geschilderte Weise ableiten lässt.

Mit der Kontrollmatrix in Händen können wir ohne Schwierigkeiten eine Generatormatrix für den [15,11]-Hamming-Code erzeugen. Hierzu müssen wir lediglich in den Orthogonalraum übergehen, was an dieser Stelle des Buchs fast eine Routineätigkeit ist. In diesem Beispiel gelingt dies sogar besonders einfach, da wir die Kontrollmatrix durch die Vertauschung von Spalten in die systematische Form bringen können. Für diesen neuen Code können wir die Generatormatrix direkt erzeugen, indem wir die Spalten rechts der Einheitsmatrix als Zeilenvektoren aufschreiben und das Ergebnis anschließend rechts um die Einheitsmatrix ergänzen. Stellen wir danach die ursprüngliche Spaltenreihenfolge wieder her, so entsteht die Matrix in Abbildung 6.32. Sie ist das, wonach wir gesucht haben: eine Generatormatrix des [15,11]-Hamming-Codes.

### 6.4.3 Zyklische Codes

In diesem Abschnitt besprechen wir eine leistungsfähige Gruppe von fehlererkennenden Codes, die in der Literatur als CRC-Codes bezeichnet werden. Die Abkürzung CRC steht für *Cyclic Redundancy Check* und weist schon im Namen auf eine charakteristische Eigenschaft hin: Nahezu alle Codes, die wir in diesem Abschnitt untersuchen werden, sind *zyklisch*. Das bedeutet, dass jede zyklische Verschiebung eines Codeworts wieder zu einem Codewort führt.

Die CRC-Codierung ist untrennbar mit der Theorie der Polynome verbunden, und genau dies ist der Grund, warum wir uns in Kapitel 2 in akribischer Vorarbeit so intensiv mit diesen algebraischen Strukturen auseinandergesetzt haben. Falls Sie das Kapitel über die mathematischen Grundlagen beim ersten Lesen übersprungen haben, ist jetzt eine gute Gelegenheit, dahin zurückzukehren und die relevanten Abschnitte über den Aufbau der Polynome und deren Ringeigenschaften nachzuarbeiten.

Die CRC-Codierung stellt einen einfachen Zusammenhang zwischen Codewörtern und Polynomen her, indem die Symbole eines Codeworts als die Koeffizienten eines Polynoms interpretiert werden. Abbildung 6.33 demonstriert die geschilderte Dualität anhand mehrerer ausgewählter Beispiele.

Im Zusammenhang mit der CRC-Codierung sind ausschließlich solche Codes interessant, die von einem dedizierten Polynom, dem sogenannten *Generatorpolynom*, erzeugt werden. Was das genau bedeutet, klärt die folgende Definition:



#### Definition 6.8 (Generierter Code)

Es seien  $g(x)$  ein Polynom aus der Menge  $\mathbb{F}[x]$  und  $n \in \mathbb{N}^+$ . Der von  $g(x)$  generierte Code der Länge  $n$  ist die Menge

$$C := \{v(x) \mid \deg v(x) < n \text{ und } g(x) \text{ teilt } v(x) \text{ ohne Rest}\}$$

$g(x)$  heißt das *Generatorpolynom* von  $C$ .

In den weiteren Betrachtungen werden ausschließlich Binärcodes behandelt, d. h., wir verwenden für  $\mathbb{F}$  den endlichen Körper  $\mathbb{Z}_2 = \{0, 1\}$ .

Als Beispiele sind in den Abbildungen 6.34 und 6.35 insgesamt vier Codes der Länge 7 zu sehen, die durch verschiedene Generatorpolyno-

#### Beispiel 1

Codewort: 0 0 0 0

Polynom:  $0 \cdot x^3 + 0 \cdot x^2 + 0 \cdot x^1 + 0 \cdot x^0 = 0$

#### Beispiel 2

Codewort: 0 1 0 1

Polynom:  $0 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x^1 + 1 \cdot x^0 = x^2 + 1$

#### Beispiel 3

Codewort: 1 1 0 1

Polynom:  $1 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x^1 + 1 \cdot x^0 = x^3 + x^2 + 1$

**Abb. 6.33:** Ist das Quellenalphabet  $\Sigma$  ein endlicher Körper  $\mathbb{F}$ , so lässt sich jedes Codewort  $v$  der Länge  $n$  als ein Polynom  $v(x) \in \mathbb{F}[x]$  mit  $\deg v(x) < n$  interpretieren. Umgekehrt entspricht jedes Polynom  $v(x) \in \mathbb{F}[x]$  mit  $\deg v(x) < n$  einem Codewort  $v$  der Länge  $n$ .

Generatorpolynom  $g(x) = x^3 + x^2 + 1, n = 7$

$(x^3 + x^2 + 1) \cdot 0$	$= 0$	0000000
$(x^3 + x^2 + 1) \cdot 1$	$= x^3 + x^2 + 1$	0001101
$(x^3 + x^2 + 1) \cdot x$	$= x^4 + x^3 + x$	0011010
$(x^3 + x^2 + 1) \cdot (x + 1)$	$= x^4 + x^2 + x + 1$	0010111
$(x^3 + x^2 + 1) \cdot x^2$	$= x^5 + x^4 + x^2$	0110100
$(x^3 + x^2 + 1) \cdot (x^2 + 1)$	$= x^5 + x^4 + x^3 + 1$	0111001
$(x^3 + x^2 + 1) \cdot (x^2 + x)$	$= x^5 + x^3 + x^2 + x$	0101110
$(x^3 + x^2 + 1) \cdot (x^2 + x + 1)$	$= x^5 + x + 1$	0100011
$(x^3 + x^2 + 1) \cdot x^3$	$= x^6 + x^5 + x^3$	1101000
$(x^3 + x^2 + 1) \cdot (x^3 + 1)$	$= x^6 + x^5 + x^2 + 1$	1100101
$(x^3 + x^2 + 1) \cdot (x^3 + x)$	$= x^6 + x^5 + x^4 + x$	1110010
$(x^3 + x^2 + 1) \cdot (x^3 + x + 1)$	$= x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$	1111111
$(x^3 + x^2 + 1) \cdot (x^3 + x^2)$	$= x^6 + x^4 + x^3 + x^2$	1011100
$(x^3 + x^2 + 1) \cdot (x^3 + x^2 + 1)$	$= x^6 + x^4 + 1$	1010001
$(x^3 + x^2 + 1) \cdot (x^3 + x^2 + x)$	$= x^6 + x^2 + x$	1000110
$(x^3 + x^2 + 1) \cdot (x^3 + x^2 + x + 1)$	$= x^6 + x^3 + x + 1$	1001011

Generatorpolynom  $g(x) = x^3 + x + 1, n = 7$

$(x^3 + x + 1) \cdot 0$	$= 0$	0000000
$(x^3 + x + 1) \cdot 1$	$= x^3 + x + 1$	0001011
$(x^3 + x + 1) \cdot x$	$= x^4 + x^2 + x$	0010110
$(x^3 + x + 1) \cdot (x + 1)$	$= x^4 + x^3 + x^2 + 1$	0011101
$(x^3 + x + 1) \cdot x^2$	$= x^5 + x^3 + x^2$	0101100
$(x^3 + x + 1) \cdot (x^2 + 1)$	$= x^5 + x^2 + x + 1$	0100111
$(x^3 + x + 1) \cdot (x^2 + x)$	$= x^5 + x^4 + x^3 + x$	0111010
$(x^3 + x + 1) \cdot (x^2 + x + 1)$	$= x^5 + x^4 + 1$	0110001
$(x^3 + x + 1) \cdot x^3$	$= x^6 + x^4 + x^3$	1011000
$(x^3 + x + 1) \cdot (x^3 + 1)$	$= x^6 + x^4 + x + 1$	1010011
$(x^3 + x + 1) \cdot (x^3 + x)$	$= x^6 + x^3 + x^2 + x$	1001110
$(x^3 + x + 1) \cdot (x^3 + x + 1)$	$= x^6 + x^2 + 1$	1000101
$(x^3 + x + 1) \cdot (x^3 + x^2)$	$= x^6 + x^5 + x^4 + x^2$	1110100
$(x^3 + x + 1) \cdot (x^3 + x^2 + 1)$	$= x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$	1111111
$(x^3 + x + 1) \cdot (x^3 + x^2 + x)$	$= x^6 + x^5 + x$	1100010
$(x^3 + x + 1) \cdot (x^3 + x^2 + x + 1)$	$= x^6 + x^5 + x^3 + 1$	1101001

**Abb. 6.34:** Die Generatorpolynome  $x^3 + x^2 + 1$  und  $x^3 + x + 1$  erzeugen den zyklischen [7,4]-Hamming-Code bzw. seine gespiegelte Variante.

Generatorpolynom  $g(x) = x^3 + 1, n = 7$

$(x^3 + 1) \cdot 0$	$= 0$	0000000
$(x^3 + 1) \cdot 1$	$= x^3 + 1$	0001001
$(x^3 + 1) \cdot x$	$= x^4 + x$	0010010
$(x^3 + 1) \cdot (x + 1)$	$= x^4 + x^3 + x + 1$	0011011
$(x^3 + 1) \cdot x^2$	$= x^5 + x^2$	0100100
$(x^3 + 1) \cdot (x^2 + 1)$	$= x^5 + x^3 + x^2 + 1$	0101101
$(x^3 + 1) \cdot (x^2 + x)$	$= x^5 + x^4 + x^2 + x$	0110110
$(x^3 + 1) \cdot (x^2 + x + 1)$	$= x^5 + x^4 + x^3 + x^2 + x + 1$	0111111
$(x^3 + 1) \cdot x^3$	$= x^6 + x^3$	1001000
$(x^3 + 1) \cdot (x^3 + 1)$	$= x^6 + 1$	1000001
$(x^3 + 1) \cdot (x^3 + x)$	$= x^6 + x^4 + x^3 + x$	1011010
$(x^3 + 1) \cdot (x^3 + x + 1)$	$= x^6 + x^4 + x + 1$	1010011
$(x^3 + 1) \cdot (x^3 + x^2)$	$= x^6 + x^5 + x^3 + x^2$	1101100
$(x^3 + 1) \cdot (x^3 + x^2 + 1)$	$= x^6 + x^5 + x^2 + 1$	1100101
$(x^3 + 1) \cdot (x^3 + x^2 + x)$	$= x^6 + x^5 + x^4 + x^3 + x^2 + x$	1111110
$(x^3 + 1) \cdot (x^3 + x^2 + x + 1)$	$= x^6 + x^5 + x^4 + x^2 + x + 1$	1110111

Generatorpolynom  $g(x) = x^3 + x^2 + x + 1, n = 7$

$(x^3 + x^2 + x + 1) \cdot 0$	$= 0$	0000000
$(x^3 + x^2 + x + 1) \cdot 1$	$= x^3 + x^2 + x + 1$	0001111
$(x^3 + x^2 + x + 1) \cdot x$	$= x^4 + x^3 + x^2 + x$	0011110
$(x^3 + x^2 + x + 1) \cdot (x + 1)$	$= x^4 + 1$	0010001
$(x^3 + x^2 + x + 1) \cdot x^2$	$= x^5 + x^4 + x^3 + x^2$	0111100
$(x^3 + x^2 + x + 1) \cdot (x^2 + 1)$	$= x^5 + x^4 + x + 1$	0110011
$(x^3 + x^2 + x + 1) \cdot (x^2 + x)$	$= x^5 + x$	0100010
$(x^3 + x^2 + x + 1) \cdot (x^2 + x + 1)$	$= x^5 + x^3 + x^2 + 1$	0101101
$(x^3 + x^2 + x + 1) \cdot x^3$	$= x^6 + x^5 + x^4 + x^3$	1111000
$(x^3 + x^2 + x + 1) \cdot (x^3 + 1)$	$= x^6 + x^5 + x^4 + x^2 + x + 1$	1110111
$(x^3 + x^2 + x + 1) \cdot (x^3 + x)$	$= x^6 + x^5 + x^2 + x$	1100110
$(x^3 + x^2 + x + 1) \cdot (x^3 + x + 1)$	$= x^6 + x^5 + x^3 + 1$	1101001
$(x^3 + x^2 + x + 1) \cdot (x^3 + x^2)$	$= x^6 + x^2$	1000100
$(x^3 + x^2 + x + 1) \cdot (x^3 + x^2 + 1)$	$= x^6 + x^3 + x + 1$	1001011
$(x^3 + x^2 + x + 1) \cdot (x^3 + x^2 + x)$	$= x^6 + x^4 + x^3 + x$	1011010
$(x^3 + x^2 + x + 1) \cdot (x^3 + x^2 + x + 1)$	$= x^6 + x^4 + x^2 + 1$	1010101

**Abb. 6.35:** Die Generatorpolynome  $x^3 + 1$  und  $x^3 + x^2 + x + 1$  offenbaren, dass nicht jedes Polynom einen zyklischen Code erzeugt.

$$C = \{ \begin{array}{l} 0101 \\ 0000 \\ 1100 \\ 0011 \\ 1111 \\ 1010 \\ 1001 \\ 0110 \end{array} \}$$

Nach Binärwert sortieren



$$C = \{ \begin{array}{l} 0000 \\ \textcolor{blue}{0011} \\ 0101 \\ 0110 \\ 1001 \\ 1010 \\ 1100 \\ 1111 \end{array} \}$$

Generatorpolynom ablesen



$$\begin{aligned} g(x) &= \textcolor{blue}{0} \cdot x^3 + \textcolor{blue}{0} \cdot x^2 + \textcolor{blue}{1} \cdot x^1 + \textcolor{blue}{1} \cdot x^0 \\ &= x + 1 \end{aligned}$$

**Abb. 6.36:** An den Codewörtern eines generierten Codes lässt sich ohne Mühe das Generatorpolynom ablesen. Es verbirgt sich hinter dem Codewort, das der kleinsten von 0 verschiedenen Binärzahl entspricht.

me erzeugt werden. Die Codes in Abbildung 6.34 sind alte Bekannte: Es handelt sich um die beiden zyklischen [7,4]-Hamming-Codes aus Abbildung 3.30.

Betrachten wir die Codewörter genauer, so werden mehreren Eigenschaften sichtbar, die jeder generierte Code, unabhängig von der Wahl des Generatorpolynoms, erfüllt:

- Das Nullpolynom ist ein Vielfaches eines jeden Generatorpolynoms; wir erhalten es ganz einfach durch die Multiplikation mit 0. Das bedeutet, dass die Symbolsequenz 00 ... 0 immer ein Codewort ist.
- Werden uns die Codewörter eines generierten Codes vorgelegt, so können wir ohne Schwierigkeiten das Generatorpolynom ablesen. Hierzu müssen wir die Codewörter lediglich als Binärzahlen interpretieren und nach der kleinsten Zahl ungleich 0 suchen. Die Ziffern dieser Zahl sind die Koeffizienten des Generatorpolynoms (Abbildung 6.36).
- Verschieben wir die Koeffizienten des Generatorpolynoms um  $i$  Positionen nach links, ohne dabei die Wortlänge  $n$  zu überschreiten, so erhalten wir wieder ein Codewort. Formal entstehen diese Codewörter durch die Multiplikation von  $g(x)$  mit den Polynomen  $x^i$ .

Die letzte Eigenschaft liefert ein erstes Indiz dafür, dass die sukzessive Multiplikation des Generatorpolynoms zu einer zyklischen Wiederholung gewisser Bitmuster führt. In den Codes aus Abbildung 6.34 ist diese Eigenschaft besonders ausgeprägt. Dort sind nicht nur diejenigen Bitmuster Codewörter, die aus der Verschiebung des Generatorpolynoms hervorgehen, sondern sämtliche Symbolsequenzen, die durch die Rotation irgendeines Codeworts entstehen. Mit anderen Worten: Die Codes sind *zyklisch*. Dass nicht jedes Generatorpolynom einen zyklischen Code erzeugt, belegen die beiden Beispiele in Abbildung 6.35. Beide Male ist die Symbolfolge 1110111 ein Codewort, nicht aber die um eine Position nach links rotierte Folge 1101111.

Wir wollen der Frage nachgehen, welche Generatorpolynome einen zyklischen Code erzeugen und welche nicht. Tatsächlich sind wir der Antwort bereits sehr nahe; wir müssen uns die Rotation eines Codeworts

$$\mathbf{v} = v_0 v_1 v_2 \dots v_{n-1} \quad (v_i \in \{0, 1\})$$

lediglich auf der Polynomebene ansehen. Das Codewort  $\mathbf{v}$  entspricht dort dem Polynom

$$v(x) = v_0 x^{n-1} + v_1 x^{n-2} + \dots + v_{n-2} x + v_{n-1}$$

Da  $v$  ein Codewort ist, muss  $v(x)$  per Definition durch das Generatorpolynom  $g(x)$  teilbar sein. Folgerichtig existiert ein Polynom  $h(x)$ , sodass sich  $v(x)$  in der Form

$$v(x) = h(x)g(x) \quad (6.25)$$

darstellen lässt. Das um eine Stelle nach links rotierte Codewort

$$v_1 v_2 \dots v_{n-2} v_{n-1} v_0$$

bezeichnen wir mit  $v \ll 1$  und unterscheiden zwei Fälle:

■ Das Codewort  $v$  beginnt mit 0

$$( \rightarrow v_0 = 0 )$$

In diesem Fall erhalten wir  $v \ll 1$  ganz einfach durch die Multiplikation von  $v(x)$  mit  $x$ :

$$v \ll 1 = x \cdot v(x) \stackrel{(6.25)}{=} x \cdot (h(x)g(x)) = (x \cdot h(x)) \cdot g(x)$$

Das bedeutet, dass  $v \ll 1$  durch  $g(x)$  ohne Rest teilbar ist und damit ebenfalls zu den Codewörtern gehört (Abbildung 6.37 oben).

■ Das Codewort  $v$  beginnt mit 1

$$( \rightarrow v_0 = 1 )$$

In diesem Fall können wir die Rotation folgendermaßen darstellen:

$$v \ll 1 = x \cdot v(x) - x^n + 1$$

Die Subtraktion von  $x^n$  sorgt dafür, dass die führende 1 gelöscht wird, und die Addition von 1 fügt sie an der rechten Stelle wieder hinzu.

Da zwischen der Addition und der Subtraktion in  $\mathbb{Z}_2$  kein Unterschied besteht, können wir diesen Zusammenhang auch so aufschreiben (Abbildung 6.37 unten):

$$v \ll 1 = x \cdot v(x) + (x^n + 1) \stackrel{(6.25)}{=} (x \cdot h(x)) \cdot g(x) + (x^n + 1)$$

Damit haben wir die Antwort schwarz auf weiß vor Augen. Teilt das Generatorpolynom  $g(x)$  das Polynom  $x^n + 1$  ohne Rest, so teilt es auch  $v \ll 1$ . In diesem Fall ist  $v \ll 1$  ein Codewort. Teilt  $g(x)$  das Polynom  $x^n + 1$  nicht ohne Rest, so bleibt auch bei der Division von  $v \ll 1$  durch  $g(x)$  ein Rest übrig. In diesem Fall ist  $v \ll 1$  kein Codewort.

1. Fall: Das Codewort beginnt mit 0

$$\begin{array}{rcl} x^3 + x^2 & & (\rightarrow 001100) \\ \downarrow \cdot x & & \\ x^4 + x^3 & & (\rightarrow 011000) \end{array}$$

2. Fall: Das Codewort beginnt mit 1

$$\begin{array}{rcl} x^5 + x^3 + x^2 & & (\rightarrow 101100) \\ \downarrow \cdot x & & \\ x^6 + x^4 + x^3 & & (\rightarrow 1011000) \\ \downarrow + x^6 & & \\ x^4 + x^3 & & (\rightarrow 011000) \\ \downarrow + 1 & & \\ x^4 + x^3 + 1 & & (\rightarrow 011001) \end{array}$$

**Abb. 6.37:** Die Rotation eines Codeworts lässt sich auf der Polynomebene durch die Anwendung einfacher algebraischer Operationen simulieren.

Fassen wir die Ergebnisse unserer Fallunterscheidung zusammen, so können wir die Frage, ob ein vorgelegtes Generatorpolynom  $g(x)$  einen zyklischen Code erzeugt, mit dem folgenden Satz beantworten:


**Satz 6.5**

Das Generatorpolynom  $g(x)$  erzeugt genau dann einen zyklischen Code der Länge  $n$ , wenn es die folgende Teilbarkeitseigenschaft aufweist:

$$g(x) \text{ teilt } x^n + 1 \text{ ohne Rest}$$

Beachten Sie, dass wir in diesem Satz stillschweigend vorausgesetzt haben, dass die Codewortlänge  $n$  den Grad des Generatorpolynoms übersteigt. Ist dies nicht der Fall, so erzeugt  $g(x)$  einen Code, der ausschließlich aus dem Nullwort  $0 \dots 0$  besteht. Dieser Code erfüllt zwar ebenfalls die Eigenschaft, zyklisch zu sein, eine praktische Bedeutung hat er freilich nicht.

Aus Satz 6.5 und der Zerlegung

$$x^7 + 1 = (x+1)(x^3+x^2+1)(x^3+x+1) \quad (6.26)$$

folgt unmittelbar, warum die Codes in Abbildung 6.34 zyklisch sind, die Codes in Abbildung 6.35 dagegen nicht. Aus der Zerlegung geht hervor, dass es genau drei Generatorpolynome gibt, die einen zyklischen Code der Länge 7 generieren. Zwei dieser Polynome haben wir in Abbildung 6.34 verwendet und das dritte ist das Polynom  $(x+1)$ . Abbildung 6.38 zeigt, dass dieses Polynom einen Code hervorbringt, den wir gut kennen:  $(x+1)$  erzeugt den geraden Paritätscode, der genau jene Bitsequenzen umfasst, die eine gerade Anzahl Einsen aufweisen.

#### 6.4.3.1 Codierung

Bis jetzt haben wir uns ausschließlich über die Codewörter eines generierten Codes Gedanken gemacht, d. h., wir haben lediglich festgelegt, welche Symbolsequenzen ein gültiges Codewort bilden und welche nicht. Über die Codierung, also die konkrete Zuordnung von den Nachrichten zu den Codewörtern, haben wir dagegen noch gar nicht gesprochen. Dies wollen wir jetzt ändern und zwei Möglichkeiten aufzeigen, wie solche Codierungen konstruiert werden können.

#### Multiplikationsmethode

In den folgenden Betrachtungen nehmen wir an, dass  $C$  ein Code ist, der von einem Generatorpolynom  $g(x)$  der Form

$$g_0x^r + g_1x^{r-1} + \dots + g_{r-1}x + g_r$$

Generatorpolynom  $g(x) = x + 1, n = 7$

$(x+1) \cdot 0$	$= 0$	0000000
$(x+1) \cdot 1$	$= x+1$	0000011
$(x+1) \cdot x$	$= x^2+x$	0000110
$(x+1) \cdot (x+1)$	$= x^2+1$	0000101
$(x+1) \cdot x^2$	$= x^3+x^2$	0001100
$(x+1) \cdot (x^2+1)$	$= x^3+x^2+x+1$	0001111
$(x+1) \cdot (x^2+x)$	$= x^3+x$	0001010
$(x+1) \cdot (x^2+x+1)$	$= x^3+1$	0001001
$(x+1) \cdot x^3$	$= x^4+x^3$	0011000
$(x+1) \cdot (x^3+1)$	$= x^4+x^3+x+1$	0011011
$(x+1) \cdot (x^3+x)$	$= x^4+x^3+x^2+x$	0011110
$(x+1) \cdot (x^3+x+1)$	$= x^4+x^3+x^2+1$	0011101
$(x+1) \cdot (x^3+x^2)$	$= x^4+x^2$	0010100
$(x+1) \cdot (x^3+x^2+1)$	$= x^4+x^2+x+1$	0010111
$(x+1) \cdot (x^3+x^2+x)$	$= x^4+x$	0010010
$(x+1) \cdot (x^3+x^2+x+1)$	$= x^4+1$	0010001
$(x+1) \cdot x^4$	$= x^5+x^4$	0110000
$(x+1) \cdot (x^4+1)$	$= x^5+x^4+x+1$	0110011
$(x+1) \cdot (x^4+x)$	$= x^5+x^4+x^2+x$	0110110
$(x+1) \cdot (x^4+x+1)$	$= x^5+x^4+x^2+1$	0110101
$(x+1) \cdot (x^4+x^2)$	$= x^5+x^4+x^3+x^2$	0111100
$(x+1) \cdot (x^4+x^2+1)$	$= x^5+x^4+x^3+x^2+x+1$	0111111
$(x+1) \cdot (x^4+x^2+x)$	$= x^5+x^4+x^3+x$	0111010
$(x+1) \cdot (x^4+x^2+x+1)$	$= x^5+x^4+x^3+1$	0111001
$(x+1) \cdot (x^4+x^3)$	$= x^5+x^3$	0101000
$(x+1) \cdot (x^4+x^3+1)$	$= x^5+x^3+x+1$	0101011
$(x+1) \cdot (x^4+x^3+x)$	$= x^5+x^3+x^2+x$	0101110
$(x+1) \cdot (x^4+x^3+x+1)$	$= x^5+x^3+x^2+1$	0101101
$(x+1) \cdot (x^4+x^3+x^2)$	$= x^5+x^2$	0100100
$(x+1) \cdot (x^4+x^3+x^2+1)$	$= x^5+x^2+x+1$	0100111
$(x+1) \cdot (x^4+x^3+x^2+x)$	$= x^5+x$	0100010
$(x+1) \cdot (x^4+x^3+x^2+x+1)$	$= x^5+1$	0100001
...		
$(x+1) \cdot (x^5+x^4+x^3+x^2+x+1) = x^6+1$		1000001

Abb. 6.38: Das Polynom  $x+1$  generiert den geraden Paritätscode.

$$c : u(x) \mapsto u(x)(x^3 + x^2 + 1)$$



$u$	$c(u)$
0000	0000000
0001	0001101
0010	0011010
0011	0010111
0100	0110100
0101	0111001
0110	0101110
0111	0100011
1000	1101000
1001	1100101
1010	1110010
1011	1111111
1100	1011100
1101	1010001
1110	1000110
1111	1001011

**Abb. 6.39:** Um das Codewort auszurechnen, müssen wir die Nachrichtenbits lediglich als Polynom auffassen, dieses mit dem Generatorpolynom multiplizieren, und anschließend die Koeffizienten extrahieren.

$$\left( \begin{array}{ccccc} g(x) & 0 & 0 & \dots & 0 \\ 0 & g(x) & 0 & \dots & 0 \\ \vdots & \ddots & & & \vdots \\ 0 & \dots & 0 & g(x) & 0 \\ 0 & \dots & 0 & 0 & g(x) \end{array} \right)$$

**Abb. 6.40:** Auf der Ebene der Koeffizienten verwandelt sich die Multiplikation mit einem Polynom  $g(x)$  zu einer Multiplikation mit dieser Matrix.

erzeugt wird, und jedes Codewort aus  $n$  Binärziffern besteht. Die Zahl  $r$  ist der Grad des Generatorpolynoms. Die einfachste Möglichkeit, eine Nachricht

$$u = u_0 u_1 u_2 \dots u_{n-r-1}$$

der Länge  $n - r$  auf die Codewörter aus  $C$  abzubilden, besteht darin,  $u$  als ein Polynom  $u(x)$  mit  $\deg u(x) < n - r$  aufzufassen und mit dem Generatorpolynom  $g(x)$  zu multiplizieren:

$$c : u(x) \mapsto u(x)g(x) \quad (6.27)$$

Dass die Abbildung  $c$  injektiv und damit eine Codierung im Sinne von Definition 3.1 ist, folgt sofort aus der Tatsache, dass zwei unterschiedliche Polynome nicht durch die Multiplikation mit einem anderen Polynom angeglichen werden können. In den Abbildungen 6.34 und 6.35 konnten wir dies gut beobachten. Die Codewörter, die dort durch die sukzessive Multiplikation des Generatorpolynoms entstanden sind, waren stets paarweise verschieden.

In Abbildung 6.39 ist eine Codierung zu sehen, die auf die geschilderte Weise für das Generatorpolynom  $g(x) = x^3 + x^2 + 1$  entsteht. Dass die Codewörter in der rechten Tabellenspalte in genau derselben Reihenfolge vorkommen wie in der oberen Hälfte von Abbildung 6.34, hat einen einfachen Grund. Fassen wir die Binärsequenzen in der linken Spalte der Codetabelle als die Koeffizientenvektoren von Polynomen auf, so entstehen der Reihe nach die Polynome, mit denen wir in Abbildung 6.34 nacheinander das Generatorpolynom multipliziert haben.

Auch wenn die Zuordnungsvorschrift (6.27) simpel erscheint, lohnt sich ein genauerer Blick darauf. Wechseln wir nämlich von der Ebene der Polynome auf die Ebene der Codewörter zurück, so wird aus der Polynommultiplikation eine Matrixmultiplikation. Abbildung 6.40 zeigt, wie die betreffende Matrix aufgebaut ist. Sie besteht aus  $n - r$  Zeilen und  $n$  Spalten und lässt sich nach einem einfachen Schema konstruieren. Um sie zu erhalten, müssen wir lediglich das Generatorpolynom versetzt untereinander schreiben und die verbleibenden Stellen mit Nullen füllen. Für unsere Beispielcodierung sieht die Matrix so aus:

$$\left( \begin{array}{ccccccc} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{array} \right) \quad (6.28)$$

Die Tatsache, dass wir die Zuordnung mithilfe einer Matrixmultiplikation berechnen können, ist bedeutend. Aus ihr folgt unmittelbar, dass Generatorpolynome *lineare Codes* erzeugen.


**Satz 6.6**

Jeder von einem Generatorpolynom erzeugte Code ist linear.

Aus Abschnitt 3.4.3 wissen wir, dass ein linearer Binärkode ein Untervektorraum von  $\mathbb{Z}_2^n$  ist und die Größe der Generatormatrix Auskunft über dessen Dimension gibt. Ist  $r$  der Grad des Generatorpolynoms und  $n$  die Länge der Codewörter, so hat der Untervektorraum die Dimension  $n - r$ . Aus Abschnitt 3.4.3 wissen wir ferner, dass die Zeilen der Generatormatrix eine Basis dieses Untervektorraums bilden. Notieren wir die Vektoren in Spaltenschreibweise, so erhalten wir für unser Beispiel die folgende Menge:

$$\left\{ \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} \right\}$$

Auf der Polynomebene können wir die Basis des erzeugten Untervektorräums besonders einfach angeben. Wir erhalten die Basisvektoren, indem wir das Generatorpolynom sukzessive mit  $1, x, x^2$  usw. multiplizieren. Das Ergebnis ist diese Menge:

$$\{x^{n-r-1} \cdot g(x), \dots, x^2 \cdot g(x), x \cdot g(x), g(x)\}$$

So einfach die bisher gewählte Codierungsvorschrift auch ist: Sie ist mit einem Makel behaftet, den wir an dieser Stelle nicht übergehen wollen. Die Codierung ist nicht *systematisch*, d. h., wir finden die Symbole der Originalnachricht nicht eins zu eins in dem generierten Codewort wieder. Glücklicherweise ist es nicht schwer, die Zuordnungsvorschrift so abzuwandeln, dass eine systematische Codierung entsteht.

Aus Abschnitt 3.4.3 wissen wir, dass elementare Zeilenumformungen keinen Einfluss darauf haben, welche Codewörter eine Generatormatrix erzeugt. Um einen systematischen Code zu erzeugen, brauchen wir die Matrix also lediglich so umzuformen, dass links die Einheitsmatrix entsteht. Die sich ergebende Codierung ist so gestaltet, dass jedes Codewort mit den Nachrichtenbits beginnt und mit den Prüfbits endet.

Abbildung 6.41 zeigt, wie die Umformung für unser Beispiel aussieht. Als Ergebnis erhalten wir eine Matrix, der wir bereits auf Seite 197 in

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Addiere Zeile 4  
auf Zeile 1 und Zeile 3

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Addiere Zeile 3 auf Zeile 2

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Addiere Zeile 2 auf Zeile 1

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

**Abb. 6.41:** Durch elementare Zeilenumformungen lässt sich eine systematische Codierung erzeugen.

$$\left( \begin{array}{cccccc} 1 & 0 & \dots & 0 & 0 & x^{n-1} \text{ mod } g(x) \\ 0 & 1 & \dots & 0 & 0 & x^{n-2} \text{ mod } g(x) \\ & & \dots & & & \\ 0 & 0 & \dots & 1 & 0 & x^{r+1} \text{ mod } g(x) \\ 0 & 0 & \dots & 0 & 1 & x^r \text{ mod } g(x) \end{array} \right)$$

**Abb. 6.42:** Die Koeffizientenmatrix in systematischer Form

Abbildung 3.30 begegnet sind. Es ist die Generatormatrix des zyklischen [7,4]-Hamming-Codes in systematischer Form.

Tatsächlich lässt sich die Generatormatrix aus Abbildung 6.41 noch viel einfacher erzeugen. Um dies zu erkennen, erinnern wir uns zunächst daran, dass jede Zeile der Generatormatrix auch ein Codewort ist. Folglich gelten auf der Polynomebene die folgenden Gleichungen:

$$\begin{aligned} x^6 + x^2 + x &\equiv 0 & (\text{mod } x^3 + x^2 + 1) \\ x^5 + x + 1 &\equiv 0 & (\text{mod } x^3 + x^2 + 1) \\ x^4 + x^2 + x + 1 &\equiv 0 & (\text{mod } x^3 + x^2 + 1) \\ x^3 + x^2 + 1 &\equiv 0 & (\text{mod } x^3 + x^2 + 1) \end{aligned}$$

Durch eine einfache Addition können wir die Terme  $x^3$  bis  $x^6$  auf der linken Seite isolieren:

$$\begin{aligned} x^6 &\equiv x^2 + x & (\text{mod } x^3 + x^2 + 1) \\ x^5 &\equiv x + 1 & (\text{mod } x^3 + x^2 + 1) \\ x^4 &\equiv x^2 + x + 1 & (\text{mod } x^3 + x^2 + 1) \\ x^3 &\equiv x^2 + 1 & (\text{mod } x^3 + x^2 + 1) \end{aligned}$$

Da auf der rechten Seite ausnahmslos Polynome stehen, die einen kleineren Grad als das Generatorpolynom aufweisen, können wir den Zusammenhang auch so aufschreiben:

$$\begin{aligned} x^6 \text{ mod } x^3 + x^2 + 1 &= x^2 + x & = 1 \cdot x^2 + 1 \cdot x^1 + 0 \cdot x^0 \\ x^5 \text{ mod } x^3 + x^2 + 1 &= x + 1 & = 0 \cdot x^2 + 1 \cdot x^1 + 1 \cdot x^0 \\ x^4 \text{ mod } x^3 + x^2 + 1 &= x^2 + x + 1 & = 1 \cdot x^2 + 1 \cdot x^1 + 1 \cdot x^0 \\ x^3 \text{ mod } x^3 + x^2 + 1 &= x^2 + 1 & = 1 \cdot x^2 + 0 \cdot x^1 + 1 \cdot x^0 \end{aligned}$$

Damit ist klar, wie sich die Generatormatrix direkt erzeugen lässt. Um die Koeffizienten rechts der Einheitsmatrix zu erhalten, müssen wir lediglich die Polynome  $x^{n-1}, x^{n-2}, \dots, x^r$  durch das Generatorpolynom dividieren und die Koeffizienten des Divisionsrests in die Matrix eintragen (Abbildung 6.42).

Wir haben an dieser Stelle ein bedeutendes Zwischenergebnis erzielt. Wir wissen jetzt, wie sich mithilfe von Polynomen lineare Codes erzeugen lassen, und wir sind gleichsam in der Lage, die Generatormatrizen so zu konstruieren, dass systematische Codierungen entstehen. Für die praktische Anwendung ist die Codierungsvorschrift allerdings nicht immer die beste. Um das Codewort zu berechnen, müssen wir eine Matrixmultiplikation durchführen, die bei größeren Bitbreiten einen merklichen Aufwand verursachen kann.

## Divisionsmethode

Wir wollen uns auf die Suche nach einer effizienteren Codierungs methode begeben, und im Vorgriff auf das Folgende sei erwähnt: Wir werden fündig werden! Dennoch sind die Überlegungen, die zu diesem Ergebnis führen, im Kern sehr einfach: Möchten wir eine systematische Codierung erhalten, so muss das Codewort die Nachrichtenbits im Klartext enthalten. Konstruieren wir die Codewörter so, dass die Prüfbits an die Nachricht angehängt werden, dann sind die ersten  $n - r$  Bits Nachrichtenbits und die restlichen  $r$  Bits Prüfbits. Das bedeutet auf der Polynomebene, dass die ersten  $n - r$  Koeffizienten des Codewortpolynoms  $v(x)$  mit dem Polynom  $x^r u(x)$  übereinstimmen müssen, wobei  $u(x)$  das Nachrichtenpolynom ist.

Als Nächstes wollen wir  $x^r u(x)$  durch das Generatorpolynom dividieren. Natürlich können wir nicht erwarten, dass die Division ohne Rest möglich ist, und so erhalten wir lediglich eine Zerlegung der Form

$$x^r u(x) = h(x)g(x) + r(x).$$

Diese Gleichung können wir in

$$x^r u(x) + r(x) = h(x)g(x)$$

umformen, und damit sind wir schon am Ziel: Einerseits steht auf der linken Seite ein Polynom, das durch  $g(x)$  teilbar ist. Andererseits ist der Grad von  $r(x)$  kleiner als  $r$ , sodass die Addition von  $r(x)$  nur die  $r$  rechts stehenden Koeffizienten verändert kann. Mit anderen Worten: Sämtliche Koeffizienten, die zu den Nachrichtenbits gehören, bleiben durch die Addition unverändert. Damit ist dieses Polynom genau jenes, das wir suchen. Es repräsentiert dasjenige Codewort, das die Nachrichtenbits im Klartext enthält.

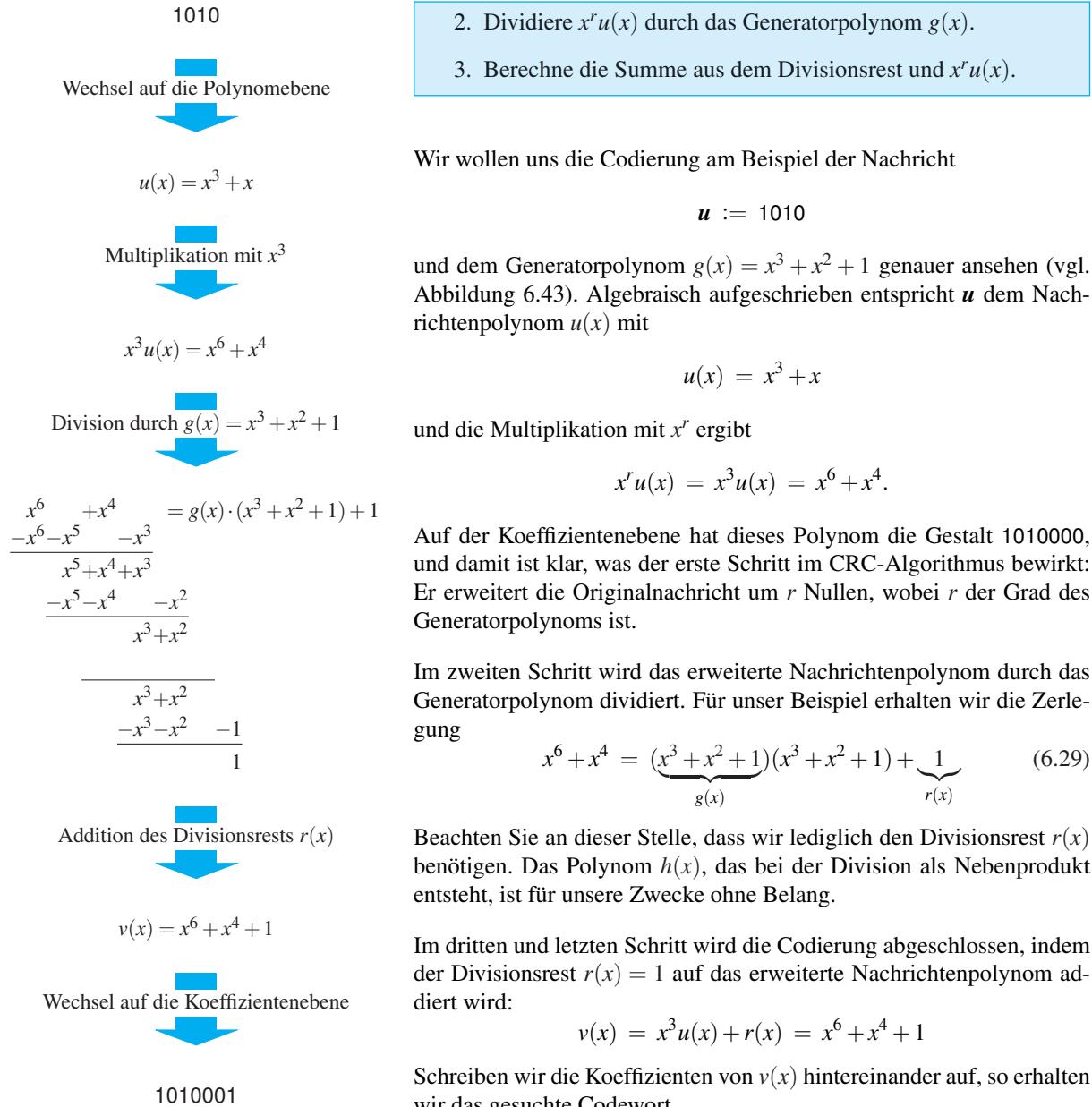
Damit haben wir einen verblüffend einfachen Codierungsalgorithmus gefunden, der in der Literatur als *CRC-Algorithmus* bezeichnet wird. Zusammengefasst liest er sich wie folgt:



### CRC-Algorithmus

- Gegeben:      Generatorpolynom  $g(x)$  mit  $\deg g(x) = r$   
                 Nachrichtenpolynom  $u(x)$  mit  $\deg u(x) < n - r$
- Gesucht:     Codewortpolynom  $v(x)$  mit  $\deg v(x) < n$

1. Multipliziere das Nachrichtenpolynom  $u(x)$  mit  $x^r$ .



**Abb. 6.43:** Der CRC-Algorithmus mit  $n = 7$  und dem Generatorpolynom  $g(x) = x^3 + x^2 + 1$ , angewendet auf die Nachricht 1010

Insgesamt bringt die Divisionsmethode die in Abbildung 6.44 enthaltene Codierungstabelle hervor.

### 6.4.3.2 Hardware-Implementierung

Unser Rechenbeispiel hat gezeigt, dass die drei Schritte des CRC-Algorithmus einen ganz unterschiedlichen Aufwand verursachen. Der erste und der dritte Schritt gingen mit Leichtigkeit von der Hand, und so waren wir die meiste Zeit damit beschäftigt, die vergleichsweise komplexe Polynomdivision auszuführen. Dass die CRC-Codierung eine so große Praxisbedeutung besitzt, liegt nicht zuletzt daran, dass sich eine Polynomdivision, so komplex sie auch wirken mag, mit einer erstaunlich platzsparenden Hardwareschaltung realisieren lässt.

Um den Schaltungsaufbau zu verstehen, werfen wir vorab einen Blick auf Abbildung 6.45. Dort sind die Einzelschritte der Polynomdivision zu sehen, die wir weiter oben für die Konstruktion unseres Beispielcodeworts durchgeführt haben. Jeder Divisionsschritt ist zweimal aufgeführt, links in der gewöhnlichen mathematischen Notation und rechts in einer Schreibweise, die auf die Angabe von Termen der Form  $x^i$  verzichtet und nur noch die Koeffizientenbits aufführt. Es ist wichtig zu sehen, dass die Divisionen in der linken und der rechten Spalte, trotz ihres unterschiedlichen Erscheinungsbilds, exakt gleich verlaufen; sie unterscheiden sich nur in der Schreibweise, und es ist an jeder Stelle möglich, von einer Notation in die andere zu wechseln.

Die Koeffizientenschreibweise ist der Schlüssel, um die Hardware-Implementierung zu verstehen. Sie macht deutlich, dass wir die Division von Polynomen aus der Menge  $\mathbb{Z}_2[x]$  auf die Manipulation von Bitmustern reduzieren können. Auf der Koeffizientenebene funktioniert die Division folgendermaßen: Von links beginnend, wird das Bitmuster des Generatorpolynoms an dem Bitmuster des Nachrichtenpolynoms vorbewegt und jedes Mal, wenn eine führende Eins angetroffen wird, eine Subtraktion ausgeführt. Die Koeffizientenschreibweise macht gleichsam deutlich, was die Subtraktion von Polynomen auf der Bitebene bedeutet: Sie entspricht dort der XOR-Verknüpfung der Koeffizienten.

In Hardware können wir das Divisionsschema mit einer Schaltung nachbilden, wie sie in Abbildung 6.46 zu sehen ist. Sie besteht aus einem seriellen Schieberegister, das von rechts gespeist wird und den Registerinhalt mit jeder steigenden Taktflanke eine Position nach links schiebt. Zwischen den Speicherelementen sind XOR-Gatter eingebracht, die über zusätzlich dazwischen geschaltete UND-Gatter mit den Koeffizientenbits des Generatorpolynoms verbunden sind. Die Verdrahtung stellt sicher, dass die Koeffizientenbits des Generatorpolynoms genau dann auf die XOR-Gatter durchgereicht werden, wenn das linke Speicherelement eine 1 enthält.

$$c : u(x) \mapsto x^3 u(x) + (x^3 u(x) \bmod g(x))$$

mit

$$g(x) = x^3 + x^2 + 1$$

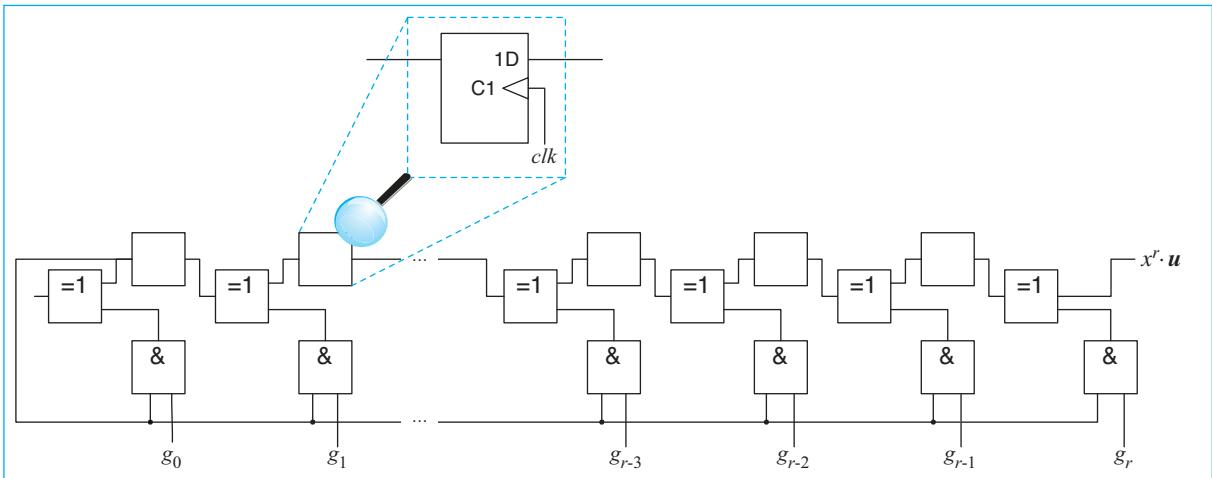


$u$	$c(u)$
0000	0000000
0001	0001101
0010	0010111
0011	0011010
0100	0100011
0101	0101110
0110	0110100
0111	0111001
1000	1000110
1001	1001011
1010	1010001
1011	1011100
1100	1100101
1101	1101000
1110	1110010
1111	1111111

**Abb. 6.44:** Die Divisionsmethode führt zu einer systematischen Codierung. Zur Verdeutlichung sind die Prüfbits in den Codewörtern farblich hervorgehoben.

Mathematische Notation	Koeffizientendarstellung
1. Divisionsschritt $\begin{array}{r} x^6 + x^4 \\ -x^6 - x^5 \\ \hline x^5 + x^4 + x^3 \end{array} = (x^3 + x^2 + 1) \cdot (1 \cdot x^3 \dots)$	1. Divisionsschritt $\begin{array}{r} 1010000 \\ 1101 \\ \hline 1110 \end{array} = 1101 \cdot 1$
2. Divisionsschritt $\begin{array}{r} x^6 + x^4 \\ -x^6 - x^5 \\ \hline x^5 + x^4 + x^3 \\ -x^5 - x^4 \\ \hline x^3 + x^2 \end{array} = (x^3 + x^2 + 1) \cdot (x^3 + 1 \cdot x^2 \dots)$	2. Divisionsschritt $\begin{array}{r} 1010000 \\ 1101 \\ \hline 1110 \end{array} = 1101 \cdot 11$
3. Divisionsschritt $\begin{array}{r} x^6 + x^4 \\ -x^6 - x^5 \\ \hline x^5 + x^4 + x^3 \\ -x^5 - x^4 \\ \hline x^3 + x^2 \\ \hline x^3 + x^2 \end{array} = (x^3 + x^2 + 1) \cdot (x^3 + x^2 + 0 \cdot x \dots)$	3. Divisionsschritt $\begin{array}{r} 1010000 \\ 1101 \\ \hline 1110 \\ 1101 \\ \hline 0110 \end{array} = 1101 \cdot 110$
4. Divisionsschritt $\begin{array}{r} x^6 + x^4 \\ -x^6 - x^5 \\ \hline x^5 + x^4 + x^3 \\ -x^5 - x^4 \\ \hline x^3 + x^2 \\ \hline x^3 + x^2 \\ -x^3 - x^2 \\ \hline 1 \end{array} = (x^3 + x^2 + 1) \cdot (x^3 + x^2 + 1) + 1$	4. Divisionsschritt $\begin{array}{r} 1010000 \\ 1101 \\ \hline 1110 \\ 1101 \\ \hline 0110 \\ 0000 \\ \hline 1100 \\ 1101 \\ \hline 001 \end{array} = 1101 \cdot 1101 + 001$

**Abb. 6.45:** Schrittweise Durchführung der Division von  $x^6 + x^4$  durch  $x^3 + x^2 + 1$  in  $\mathbb{Z}_2[x]$ , links in der gewöhnlichen mathematischen Darstellung und rechts in der platzsparenden Koeffizientennotation

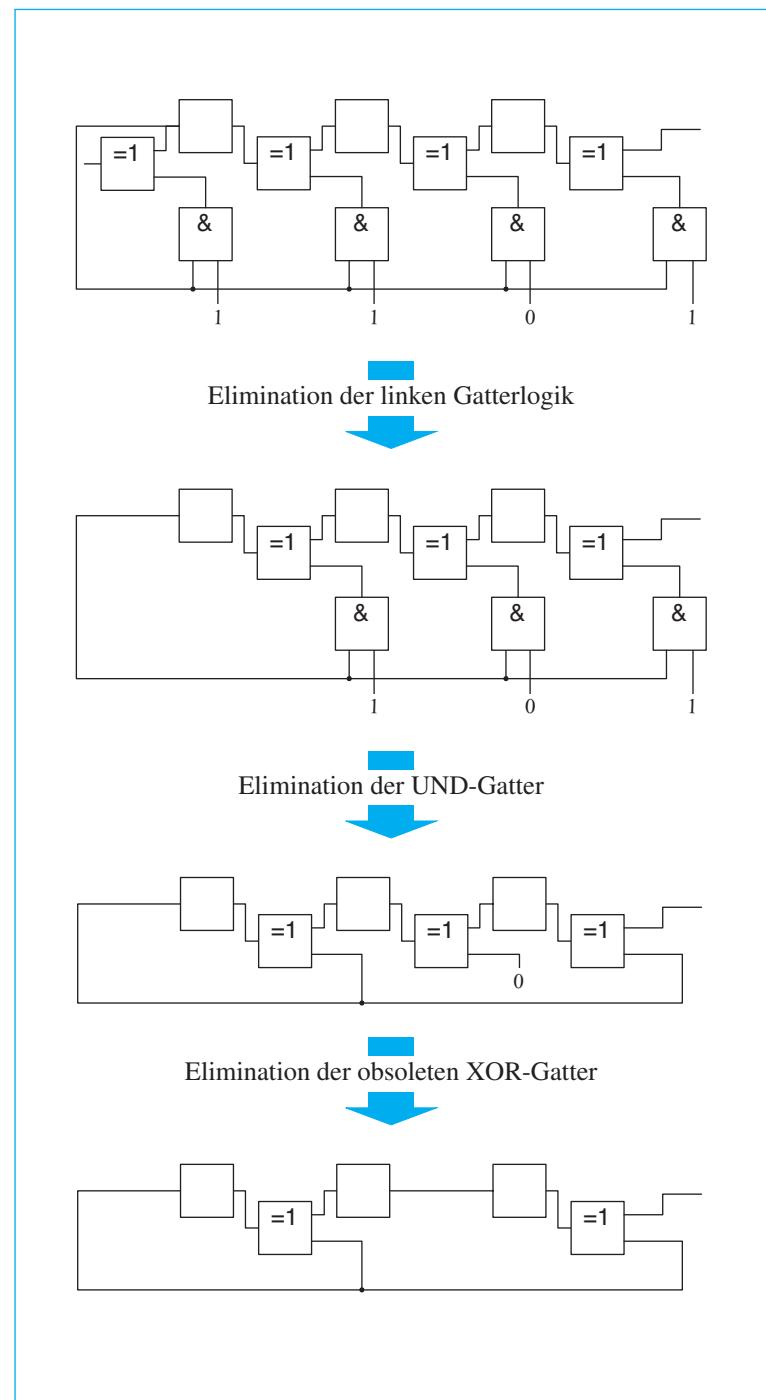


**Abb. 6.46:** Allgemeines Schema eines CRC-Encoders

Folgerichtig wird der Inhalt des Schieberegisters mit dem Generatorpolynom XOR-Verknüpft, wenn in dem linken Speicherelement eine 1 vorhanden ist. Enthält das linke Speicherelement eine 0, so blendet die UND-Gatter die Koeffizientenbits des Generatorpolynoms aus, und das Register führt im nächsten Schritt eine gewöhnliche Schiebeoperation durch.

Abbildung 6.47 zeigt, dass sich die Divisionsschaltung noch weiter vereinfachen lässt. Zunächst fällt auf, dass der Ausgang des am weitesten links stehenden XOR-Gatters gar nicht in die Berechnung einbezogen ist und daher, zusammen mit seinem angeschlossenen UND-Gatter, ersatzlos gestrichen werden darf. Noch weiter lässt sich die Schaltung vereinfachen, wenn sie für ein bestimmtes Generatorpolynom ausgelegt ist. In diesem Fall können wir auf sämtliche UND-Gatter verzichten und überall dort, wo der Koeffizient des Generatorpolynoms 0 ist, auch das XOR-Gatter eliminieren. Auf diese Weise entsteht eine äußerst kompakte Hardwareschaltung, die aus einer Reihe von Speicherelementen und einigen wenigen XOR-Gattern aufgebaut ist.

Dass die Schaltung in der entwickelten Form tatsächlich ihren Dienst erfüllt, stellt der Simulationslauf in Abbildung 6.48 unter Beweis. Vergleichen wir die jeweiligen Inhalte des Schieberegisters mit den händisch ausgerechneten Zwischenergebnissen, so wird klar, dass die Hardware schaltung eins zu eins die Division nachbildet, die wir weiter oben, in Abbildung 6.45, durchgeführt haben.



**Abb. 6.47:** Bleibt das Generatorpolynom unverändert, lässt sich die Hardwareschaltung deutlich vereinfachen.

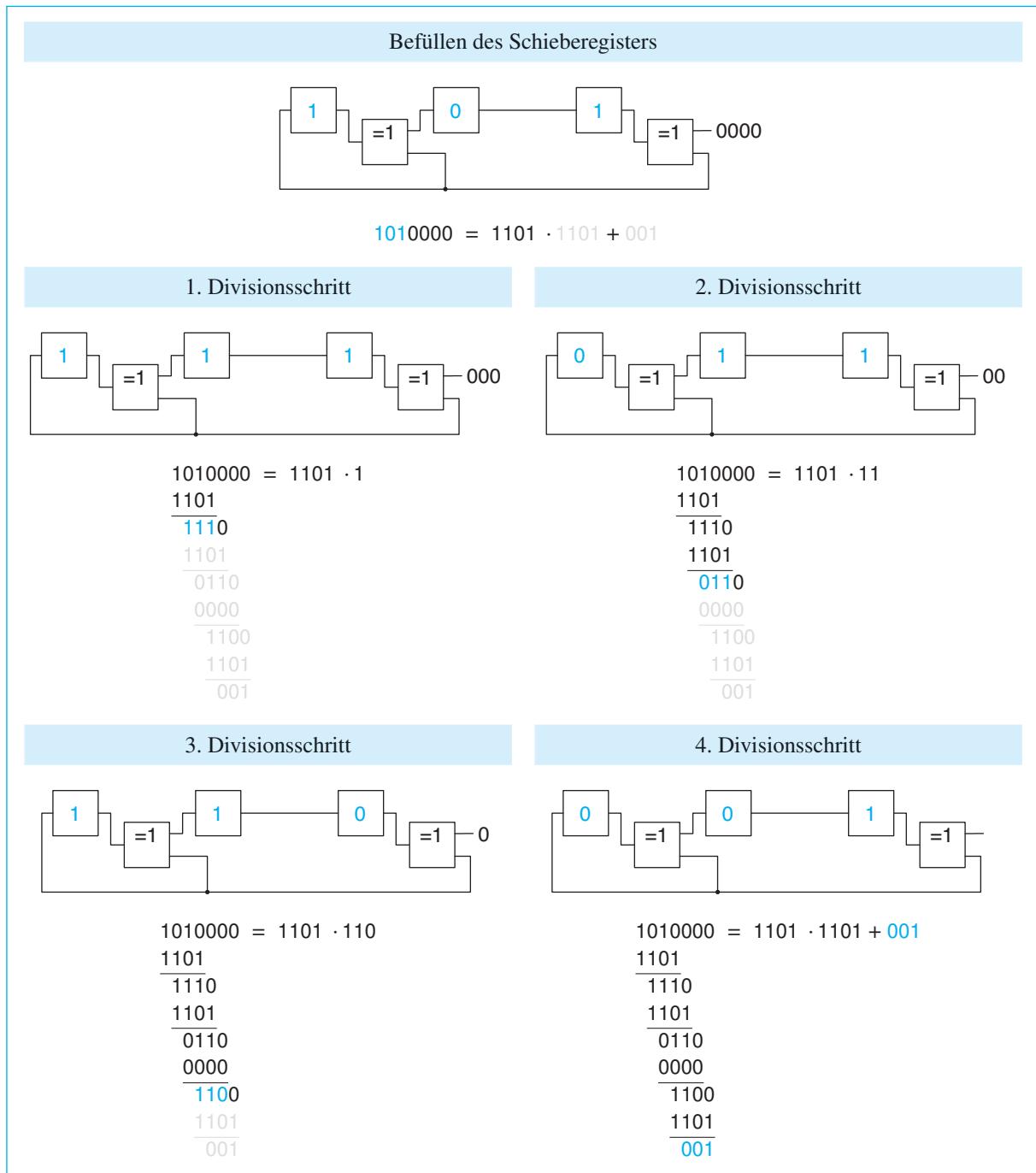


Abb. 6.48: Hardware-Implementierung der Polynomdivision mithilfe eines Schieberegisters

#### 6.4.4 BCH-Codes

In diesem Abschnitt besprechen wir eine Klasse von Codes, die fast zeitgleich, in den Jahren 1959 und 1960, von dem Franzosen Alexis Hocquenghem und den beiden Indern Dinen Kumar Ray-Chaudhuri und Raj Chandra Bose entdeckt wurde. Ihren geistigen Vätern zu Ehren sprechen wir heute von den *Bose-Chaudhuri-Hocquenghem-Codes*, oder kürzer: von den *BCH-Codes*.

Die BCH-Codes besitzen eine Eigenschaft, die sie für die Praxis besonders wertvoll machen: Sie basieren auf einem Konstruktionsprinzip, das für jede frei wählbare Distanz  $d$  einen Code  $C$  mit  $\Delta C \geq d$  hervorbringt. Wir können das Verfahren somit gezielt einsetzen, um beispielsweise einen 3-fehlerkorrigierenden Code zu erhalten. Um zu zeigen, wie dies funktioniert (vgl. [59]), beginnen wir mit etwas Wohlvertrautem: dem [15,11]-Hamming-Code. Dieser Code umfasst 2<sup>11</sup> binäre Codewörter der Länge 15, die einen Mindestabstand von 3 zueinander aufweisen. Es handelt sich also um einen 1-fehlerkorrigierenden Code.

Die Kontrollmatrix des [15,11]-Hamming-Codes kennen wir aus Abschnitt 6.4.2. Auf Seite 376 haben wir diese in der folgenden Form hergeleitet:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

Die Kontrollmatrix besitzt die Eigenschaft, dass ein Vektor  $v$  genau dann ein Codewort ist, wenn die Multiplikation von  $H$  mit  $v^T$  den Nullvektor ergibt:

$$v \in C \Leftrightarrow Hv^T = \mathbf{0}$$

Wir wollen versuchen, aus dem [15,11]-Hamming-Code einen 2-fehlerkorrigierenden Code zu erzeugen, ohne die Länge der Codewörter zu verändern. Da wir keine zusätzlichen Bits hinzufügen dürfen, kann dies nur gelingen, wenn wir die Menge der Codewörter so ausdünnen, dass die verbleibenden eine minimale Hamming-Distanz von 5 aufweisen. Aber wie kann die Menge der Codewörter systematisch verringert werden? Der Schlüssel für unser Vorhaben ist die Modifikation der Kontrollmatrix. Da jedes Codewort die Eigenschaft erfüllt, dass ihr Produkt mit jeder Zeile der Kontrollmatrix den Nullvektor ergibt, können wir die Codewörter ausdünnen, indem wir die Kontrollmatrix um weitere Zeilen ergänzen. Es bleibt, die Frage zu klären, wie diese zusätzlichen Zeilen aussehen müssen.

Die Antwort liegt in einer Eigenschaft von Kontrollmatrizen verborgen, die wir noch nicht kennen. Wir wollen diese Eigenschaft an einem konkreten Beispiel herausarbeiten und benutzen dazu den oben eingeführten [15,11]-Hamming-Code. Dieser Code weist die Distanz 3 auf, und zwei Codewörter, die in dieser Entfernung zueinander liegen, sind z. B. diese hier:

$$\begin{aligned}v_1 &= 001100011100111 \\v_2 &= 001110011000110\end{aligned}$$

Die beiden Codewörter sind zwei Vektoren  $v_1$  und  $v_2$  im Vektorraum des Hamming-Codes.

Da wir einen linearen Code vor uns haben, entspricht auch der Differenzvektor

$$v = v_1 - v_2$$

einem Codewort. In diesem kommt an genau jenen Positionen eine 1 vor, an denen sich  $v_1$  und  $v_2$  unterscheiden.

In unserem Beispiel gilt

$$v = 000010000100001$$

Ist  $\mathbf{h}_i$  die  $i$ -te Spalte der Kontrollmatrix ( $0 \leq i < n$ ), dann können wir die Multiplikation von  $H$  mit dem Vektor  $v^T$  in unserem Beispiel folgendermaßen aufschreiben:

$$Hv^T = \mathbf{h}_4 + \mathbf{h}_9 + \mathbf{h}_{14}$$

Wegen  $Hv^T = \mathbf{0}$  ist

$$\mathbf{h}_4 + \mathbf{h}_9 + \mathbf{h}_{14} = \mathbf{0},$$

und das bedeutet, dass die Vektoren  $\mathbf{h}_4, \mathbf{h}_9, \mathbf{h}_{14}$  linear abhängig sind.

Wir rekapitulieren: Aufgrund der Eigenschaft des Hamming-Codes, die Distanz 3 aufzuweisen, waren wir in der Lage, ein Codewort  $v$  mit dem Hamming-Gewicht 3 zu konstruieren. Daraus konnten wir wiederum schließen, dass in der Kontrollmatrix 3 linear abhängige Spalten vorkommen müssen. Verallgemeinert führt diese Argumentation zu dem folgenden Satz:

### Satz 6.7

In der Kontrollmatrix eines linearen Codes  $C$  gibt es  $\Delta C$  linear abhängige Spalten.

Mit diesem Satz können wir formal überprüfen, dass die Distanz des Hamming-Codes größer als 2 ist. Wäre die Distanz gleich 2, so müssten wir in der Kontrollmatrix zwei linear abhängige Spalten finden können. Vergleichen wir alle Spalten paarweise miteinander, so stellen wir aber jedes Mal fest, dass diese linear unabhängig sind. Das bedeutet nach Satz 6.7, dass die Distanz größer als 2 sein muss.

Wir wissen jetzt, dass die Kontrollmatrix eines linearen Codes  $\Delta C$  linear abhängige Spalten enthält. Dies wirft die Frage auf, ob auch weniger als  $\Delta C$  Spalten linear abhängig sein können. Der folgende Satz beantwortet diese Frage negativ:

### Satz 6.8

Existieren  $d$  linear abhängige Spalten in der Kontrollmatrix eines linearen Codes  $C$ , so ist  $\Delta C \leq d$ .

*Beweis:* Wie oben bezeichnen wir die Spalten der Kontrollmatrix mit  $\mathbf{h}_0, \dots, \mathbf{h}_{n-1}$ . Existieren  $d$  linear abhängige Spalten, so lässt sich der Nullvektor als eine Linearkombination

$$a_{i_1}\mathbf{h}_{i_1} + a_{i_2}\mathbf{h}_{i_2} + \dots + a_{i_d}\mathbf{h}_{i_d} = \mathbf{0}$$

darstellen, wobei nicht alle  $a_i$  gleich null sind. Dann erfüllt die Bitsequenz  $v$ , die genau an jenen Positionen eine 1 aufweist, an denen der entsprechende  $a$ -Koeffizient gleich 1 ist, die Beziehung

$$Hv^T = \mathbf{0}$$

Das bedeutet, dass  $v$  ein Codewort ist. Da die Entfernung von  $v$  zu  $\mathbf{0}$  höchstens  $d$  beträgt, kann die Code-Distanz nicht größer als  $d$  sein, was zu beweisen war.  $\square$

#### 6.4.4.1 Vandermonde-Matrizen

Wir kommen nun auf unseren ursprünglichen Plan zurück: Wir wollen den [15,11]-Hamming-Code so ausdünnen, dass sich die Code-Distanz auf 5 erhöht. Satz 6.7 gibt uns den ersten Hinweis darauf, was hierfür zu tun ist. Wir müssen die Kontrollmatrix so um weitere Zeilen ergänzen, dass 4 beliebig ausgewählte Spalten stets linear unabhängig sind. Aber wie lassen sich solche Zeilen systematisch finden? Die Lösung für unser Problem ist in einer speziellen Klasse von Matrizen verborgen, die nach dem französischen Mathematiker Alexandre-Théophile Vandermonde benannt sind.

**Definition 6.9 (Matrix vom Vandermonde-Typ)**

Sind  $\lambda_1, \dots, \lambda_n$  Elemente eines Körpers, dann nennen wir die folgende Matrix eine Matrix vom *Vandermonde-Typ*:

$$\begin{pmatrix} \lambda_1 & \lambda_2 & \dots & \lambda_n \\ \lambda_1^2 & \lambda_2^2 & \dots & \lambda_n^2 \\ \vdots & \vdots & \ddots & \\ \lambda_1^n & \lambda_2^n & \dots & \lambda_n^n \end{pmatrix}$$



Für die Berechnung der Determinante einer Vandermonde-Matrix werden Sie in der Literatur mit großer Sicherheit eine Formel vorfinden, die etwas einfacher ist als unsere Formel (6.30). Sie lautet dort folgendermaßen:

$$\det M = \prod_{1 \leq i < j \leq n} (\lambda_j - \lambda_i)$$

Die beiden Formeln unterscheiden sich, weil wir uns in Definition 6.9 erlaubt haben, von der üblichen Definition einer Vandermonde-Matrix geringfügig abzuweichen. Für gewöhnlich wird in der Mathematik unter einer Vandermonde-Matrix eine Matrix der folgenden Gestalt verstanden:

$$\begin{pmatrix} 1 & 1 & \dots & 1 \\ \lambda_1 & \lambda_2 & \dots & \lambda_n \\ \vdots & \vdots & \ddots & \\ \lambda_1^{n-1} & \lambda_2^{n-1} & \dots & \lambda_n^{n-1} \end{pmatrix}$$

Die Matrizen aus Definition 6.9 unterscheiden sich von den echten Vandermonde-Matrizen also dadurch, dass die  $i$ -te Spalte zusätzlich mit  $\lambda_i$  multipliziert ist. Dies ist auch der Grund, warum wir in Definition 6.9 nicht von Vandermonde-Matrizen, sondern, ein wenig vorsichtiger, von Matrizen vom Vandermonde-Typ sprechen.

Die grundlegende Eigenschaft, die wir für unser Vorhaben benötigen, ist aber bei beiden Matrizontypen gleich: Sind die Elemente  $\lambda_1, \dots, \lambda_n$  alle ungleich 0, so sind die Spalten der Matrix genau dann linear unabhängig, wenn die Elemente  $\lambda_1, \dots, \lambda_n$  paarweise verschieden sind.

Matrizen vom Vandermonde-Typ haben interessante Eigenschaften. Beispielsweise lässt sich zeigen, dass sich deren Determinante über die folgende Formel berechnen lässt:

$$\det M = \lambda_1 \cdot \dots \cdot \lambda_n \cdot \prod_{1 \leq i < j \leq n} (\lambda_j - \lambda_i) \quad (6.30)$$

Aus der Determinantenformel folgt sofort eine zentrale Eigenschaft, die wir weiter unten benötigen werden:

**Satz 6.9**

Sind die Körperelemente  $\lambda_1, \dots, \lambda_n$  alle ungleich 0, so sind die Spalten einer Matrix vom Vandermonde-Typ genau dann linear unabhängig, wenn  $\lambda_1, \dots, \lambda_n$  paarweise verschieden sind.

So weit, so gut, aber wie können wir dieses Ergebnis in unser ursprüngliches Vorhaben einbringen? Wir standen weiter oben vor der Aufgabe, die Kontrollmatrix des [15,11]-Hamming-Codes so um weitere Zeilen zu ergänzen, dass 4 beliebig ausgewählte Spalten stets linear unabhängig sind. Satz 6.9 stellt zweifellos einen Zusammenhang zwischen Vandermonde-Matrizen und der linearen Unabhängigkeit ihrer Spalten her, doch die Kontrollmatrix hat mit einer Vandermonde-Matrix auf den ersten Blick wenig gemeinsam.

An dieser Stelle kommt eine Finesse ins Spiel, die wir mit Fug und Recht als die gedankliche Kernidee der BCH-Codierung bezeichnen dürfen. Sie besteht darin, einen Übergang von  $\mathbb{Z}_2$  in den Erweiterungskörper  $\mathbb{F}_{2^4}$  mit 16 Elementen zu vollziehen, indem jeweils vier Bits einer Spalte zu einem einzelnen Element aus  $\mathbb{F}_{2^4}$  verschmolzen werden. Die Kontrollmatrix sieht dann so aus:

$$(1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 15)$$

Wie sich ein endlicher Körper mit 16 Elementen erzeugen lässt, wissen wir aus Abschnitt 2.4. Wir brauchen hierzu lediglich ein irreduzibles Polynom  $m(x) \in \mathbb{Z}_2[x]$  mit  $\deg m(x) = 4$  zu wählen; danach können wir  $\mathbb{F}_{2^4}$  direkt mit dem Polynomring  $\mathbb{Z}_2[x]_{m(x)}$  identifizieren. Die Eigenschaft von  $m(x)$ , irreduzibel zu sein, sorgt dafür, dass der Polynomring  $\mathbb{Z}_2[x]_{m(x)}$  zu einem Körper wird.

In den folgenden Rechnungen verwenden wir das irreduzible Polynom

$$m(x) := x^4 + x + 1$$

und schreiben die Körperelemente, wie oben, als natürliche Zahlen auf. Behalten Sie dabei stets im Gedächtnis, dass sich hinter einer solchen natürlichen Zahl  $\lambda$  in Wirklichkeit ein Polynom verbirgt, dessen Koeffizientenvektor mit der Binärdarstellung von  $\lambda$  übereinstimmt.

Jetzt kommt ein entscheidender Schritt: Wir erweitern die eben konstruierte Matrix so um drei neue Zeilen, dass jede Spalte die Form

$$\begin{pmatrix} \lambda_i \\ \lambda_i^2 \\ \lambda_i^3 \\ \lambda_i^4 \end{pmatrix}$$

aufweist. Die fertige Matrix sieht dann so aus:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 1 & 4 & 5 & 3 & 2 & 7 & 6 & 12 & 13 & 8 & 9 & 15 & 14 & 11 & 10 \\ 1 & 8 & 15 & 12 & 10 & 1 & 1 & 10 & 15 & 15 & 12 & 8 & 10 & 8 & 12 \\ 1 & 3 & 2 & 5 & 4 & 6 & 7 & 15 & 14 & 12 & 13 & 10 & 11 & 9 & 8 \end{pmatrix}$$

Beachten Sie, dass die Wahl des irreduziblen Polynoms  $m(x)$  die Multiplikationstabelle von  $\mathbb{F}_{2^4}$  bestimmt. Hätten wir für  $m(x)$  ein anderes irreduzibles Polynom ausgewählt, so wäre zwar die Werteverteilung eine andere gewesen, die entstandene Matrix für unsere Zwecke aber gleichermaßen gut geeignet.

Die erweiterte Matrix können wir als die Generatormatrix einer linearen Codierung über dem Körper  $\mathbb{F}_{2^4}$  auffassen, die Nachrichten der Länge 4 auf Codewörter der Länge 15 abbildet. Über die Distanz des entstandenen Codes können wir ebenfalls eine belastbare Aussage machen. Greifen wir nämlich vier beliebige Spalten aus der Generatormatrix heraus, so erhalten wir eine  $4 \times 4$ -Matrix der Form

$$\begin{pmatrix} \lambda_1 & \lambda_2 & \lambda_3 & \lambda_4 \\ \lambda_1^2 & \lambda_2^2 & \lambda_3^2 & \lambda_4^2 \\ \lambda_1^3 & \lambda_2^3 & \lambda_3^3 & \lambda_4^3 \\ \lambda_1^4 & \lambda_2^4 & \lambda_3^4 & \lambda_4^4 \end{pmatrix}$$

Kontrollmatrix in Binärform	Reduzierte Kontrollmatrix	Übergang in den Orthogonalraum
$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$

**Abb. 6.49:** Aus der Kontrollmatrix eines linearen Codes lässt sich die Generatormatrix durch den Übergang in den Orthogonalraum gewinnen. Als Ergebnis erhalten wir in diesem Beispiel einen 2-fehlerkorrigierenden Code. Die Codewörter sind eine Untermenge des [15,11]-Hamming-Codes.

Das Ergebnis ist eine Matrix vom Vandermonde-Typ, in der die Körperelemente  $\lambda_1, \dots, \lambda_4$  alle ungleich 0 und paarweise verschieden sind. Für solche Matrizen attestiert Satz 6.9, dass die Spalten linear unabhängig sind, und dies wiederum bedeutet nach Satz 6.7, dass die Code-Distanz mindestens 5 betragen muss.

Übersetzen wir die Körperelemente in ihre Binärdarstellung zurück, so erhalten wir die Kontrollmatrix, die in Abbildung 6.49 in der linken Spalte zu sehen ist. In dieser Form enthält die Matrix noch ein gehöriges Maß an Redundanz in Form linear abhängiger Zeilen. Bringen wir die Matrix in ihre reduzierte Form und eliminieren dadurch alle linearen Abhängigkeiten, so entsteht die Kontrollmatrix, die in Abbildung 6.49 in der mittleren Spalte abgedruckt ist.

Gehen wir jetzt noch, wie in Abbildung 6.49 geschehen, in den Orthogonalraum über, so entsteht die Generatormatrix des Codes, nach dem wir gesucht haben. Es ist eine ausgedünnte Variante des [15,11]-Hamming-Codes, dessen Codewörter einen Mindestabstand von 5 aufweisen.

Codewörter des ausgedünnten Hamming-Codes							
0	0000000000000000	32	101011011000000	64	111111100000000	96	010100111000000
1	011001010000001	33	110010001000001	65	100110110000001	97	001101101000001
2	101100010000010	34	000111001000010	66	0100111100000010	98	111000101000010
3	110101000000011	35	011110011000011	67	001010100000011	99	100001111000011
4	110100000100100	36	011111011100100	68	001011100100100	100	100000111100100
5	101101010100101	37	000110001100101	69	010010110100101	101	111001101100101
6	011000010100110	38	110011001100110	70	100111110100110	102	001100101100110
7	000001000100111	39	101010011100111	71	111110100100111	103	010101111100111
8	111001000101000	40	010010011101000	72	000110100101000	104	101101111101000
9	100000010101001	41	001011001101001	73	011111110101001	105	110100101101001
10	010101010101010	42	111110001101010	74	101010110101010	106	0000001101101010
11	001100000101011	43	100111011101011	75	110011100101011	107	011000111101011
12	001101000001100	44	100110011001100	76	110010100001100	108	011001111001100
13	010100010001101	45	111111001001101	77	101011110001101	109	0000000101001101
14	100001010001110	46	001010001001110	78	011110110001110	110	110101101001110
15	111000000001111	47	010011011001111	79	000111100001111	111	101100111001111
16	010011000110000	48	111000011110000	80	101100100110000	112	000111111100000
17	001010010110001	49	100001001110001	81	110101110110001	113	011110101110001
18	111111010110010	50	010100001110010	82	0000000110110010	114	101011101110010
19	100110000110011	51	001101011110011	83	011001100110011	115	110010111110011
20	100111000010100	52	001100011010100	84	011000100010100	116	110011111010100
21	111110010010101	53	010101001010101	85	0000001110010101	117	101010101010101
22	001011010010110	54	1000000001010110	86	110100110010110	118	011111010101110
23	010010000010111	55	111001011010111	87	101101100010111	119	000110111010111
24	101010000011000	56	0000001011011000	88	010101100011000	120	111110111011000
25	110011010011001	57	0110000001011001	89	001100110011001	121	100111101011001
26	000110010011010	58	101101001011010	90	111001110011010	122	010010101011010
27	011111000011011	59	110100011011011	91	100000100011011	123	001011111011011
28	011110000111100	60	110101011111100	92	1000001100111100	124	001010111111100
29	000111010111101	61	101100000111101	93	111000110111101	125	010011101111101
30	110010010111110	62	011001001111110	94	001101110111110	126	100110101111110
31	101011000111111	63	000000011111111	95	010100100111111	127	111111111111111

**Tab. 6.4:** Die systematische Ausdünnung der Hamming-Codewörter bringt einen 2-fehlerkorrigierenden Code hervor.

In Tabelle 6.4 können Sie sich selbst von dem Ergebnis überzeugen: Wenn Sie zwei beliebige Codewörter auswählen, werden Sie feststellen, dass diese an mindestens 5 Bitpositionen einen unterschiedlichen Wert aufweisen. Daraus folgt, nach Satz 6.3, dass der erzeugte Code 2-fehlerkorrigierend ist.

Auf die gleiche Weise können wir einen Code erzeugen, der 3 Fehler korrigieren kann. Hierzu müssen wir die Ausdünnung so weit fortsetzen, dass der Mindestabstand zweier Codewörter 7 beträgt. Wir erreichen dies, indem wir die Kontrollmatrix um zwei weitere Zeilen ergänzen:

$$\left( \begin{array}{cccccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 1 & 4 & 5 & 3 & 2 & 7 & 6 & 12 & 13 & 8 & 9 & 15 & 14 & 11 & 10 \\ 1 & 8 & 15 & 12 & 10 & 1 & 1 & 10 & 15 & 15 & 12 & 8 & 10 & 8 & 12 \\ 1 & 3 & 2 & 5 & 4 & 6 & 7 & 15 & 14 & 12 & 13 & 10 & 11 & 9 & 8 \\ 1 & 6 & 6 & 7 & 7 & 6 & 1 & 7 & 1 & 6 & 1 & 6 & 7 & 1 & \\ 1 & 12 & 10 & 15 & 8 & 1 & 1 & 8 & 10 & 10 & 15 & 12 & 8 & 12 & 15 \end{array} \right)$$

Dies ist die Kontrollmatrix eines 3-fehlerkorrigierenden Codes. Dass die notwendige Distanz von 7 tatsächlich erreicht wird, lässt sich mit dem gleichen Argument beweisen wie oben. Greifen wir willkürlich 6 Spalten dieser Matrix heraus, so erhalten wir eine  $6 \times 6$ -Matrix vom Vandermonde-Typ, die alle Voraussetzungen von Satz 6.9 erfüllt. Damit sind die Spalten linear unabhängig, und die Code-Distanz muss, nach Satz 6.7, größer als 6 sein.

Um den Code über dem Körper  $\mathbb{F}_{2^4}$  in einen Binärkode zu wandeln, gehen wir genauso vor wie in unserem ersten Beispiel und ersetzen die Körperelemente aus  $\mathbb{F}_{2^4}$  ganz einfach in ihre 4-Bit-Spaltendarstellung zurück. Als Ergebnis erhalten wir die Binärmatrix, die in Abbildung 6.50 in der linken Spalte abgebildet ist.

Bringen wir die Matrix in ihre reduzierte Form, so verschwinden die linear abhängigen Zeilen. Das Ergebnis ist eine Matrix, die nur noch aus 10 Zeilen besteht und in Abbildung 6.50 (Mitte) zu sehen ist.

Um die Generatormatrix des Codes zu erhalten, können wir ebenfalls so vorgehen wie in unseren ersten Beispiel. Indem wir in den Orthogonalraum wechseln, erhalten wir als Ergebnis die Matrix, die in Abbildung 6.50 in der rechten Spalte dargestellt ist. Sie generiert einen Code mit 32 Codewörtern, die einen Mindestabstand von 7 zueinander aufweisen. Tabelle 6.5 zeigt, um welche Codewörter es sich im Einzelnen handelt.

Jetzt fehlt uns nur noch ein kleiner Baustein, um die Konstruktion der BCH-Codes zu vollenden. Um ihn zu verstehen, werfen wir einen erneuten Blick auf Satz 6.9. Dort ist explizit gefordert, dass die Elemente  $\lambda_1, \dots, \lambda_n$  einer Vandermonde-Matrix paarweise verschieden sein müssen, und in unserem Beispiel war dies auch der Fall. Bei den BCH-Codes werden die Elemente aber nicht, wie bei uns, in einer willkürlichen Reihenfolge aufgezählt, sondern systematisch, mithilfe eines *primitiven Elements*, erzeugt. Wir erinnern uns: Ein primitives Element  $\alpha$

**Abb. 6.50:** Erweitern wir die Kontrollmatrix um zwei weitere Zeilen, so entsteht ein 3-fehlerkorrigierender Code mit 32 Codewörtern.

hat die Eigenschaft, dass seine Potenzen  $\alpha^0, \alpha^1, \alpha^2$  nacheinander alle von 0 verschiedenen Körperelemente durchlaufen.

Für unseren Beispielkörper  $\mathbb{F}_{2^4}$  ist die 2 ein primitives Element. Schreiben wir die Potenzen  $2^i$  für  $i = 0$  bis  $i = 14$  von rechts nach links auf, so erhalten wir folgenden Vektor:

$$( \begin{array}{cccccccccccccc} 9 & 13 & 15 & 14 & 7 & 10 & 5 & 11 & 12 & 6 & 3 & 8 & 4 & 2 & 1 \end{array} )$$

Diesen Vektor können wir, genau wie oben, durch weitere Zeilen ergänzen. Fügen wir beispielsweise drei neue hinzu, so erhalten wir die

Codewörter des noch weiter ausgedünnten Hamming-Codes					
0	0000000000000000	8	111000011110000	16	111111100000000
1	001011001101001	9	110011010011001	17	110100101101001
2	010101010101010	10	101101001011010	18	101010110101010
3	011110011000011	11	100110000110011	19	100001111000011
4	100110011001100	12	011110000111100	20	011001111001100
5	101101010100101	13	010101001010101	21	010010110100101
6	110011001100110	14	001011010010110	22	001100101100110
7	111000000001111	15	000000011111111	23	000111100001111
					31 111111111111111

**Tab. 6.5:** Dünnen wir die Hamming-Codewörter weiter aus, so erreichen wir einen 3-fehlerkorrigierenden Code.

folgende Matrix:

$$\begin{pmatrix} 9 & 13 & 15 & 14 & 7 & 10 & 5 & 11 & 12 & 6 & 3 & 8 & 4 & 2 & 1 \\ 13 & 14 & 10 & 11 & 6 & 8 & 2 & 9 & 15 & 7 & 5 & 12 & 3 & 4 & 1 \\ 15 & 10 & 12 & 8 & 1 & 15 & 10 & 12 & 8 & 1 & 15 & 10 & 12 & 8 & 1 \\ 14 & 11 & 8 & 9 & 7 & 12 & 4 & 13 & 10 & 6 & 2 & 15 & 5 & 3 & 1 \end{pmatrix}$$

Genau wie im Falle des [15,11]-Hamming-Codes können wir daraus eine binäre Matrix ableiten, indem wir die Körperelemente als Dualzahlen aufschreiben. Das Ergebnis der Transformation ist in Abbildung 6.51 dargestellt. Sie sehen dort die Kontrollmatrix und die Generatormatrix des Codes, der in unserer Nomenklatur als  $\text{BCH}_{\mathbb{F},2}(15,2)$  bezeichnet wird. Was die in Klammern angegebenen Codeparameter im Detail bedeuten, reichen wir gleich in einer formalen Definition nach.

Tabelle 6.6 fasst die Codewörter von  $\text{BCH}_{\mathbb{F},2}(15,2)$  zusammen. Die Art und Weise, wie wir die Generatormatrix konstruiert haben, stellt sicher, dass sich die Codewörter von jenen des ausgedünnten Hamming-Codes nur in der Reihenfolge unterscheiden, in der die Bits innerhalb der Codewörter angeordnet sind. Kurzum:  $\text{BCH}_{\mathbb{F},2}(15,2)$  und der ausgedünnte Hamming-Code sind äquivalent. Dass der ausgedünnte Hamming-Code selbst kein BCH-Code ist, liegt also nur daran, dass die Spaltenvektoren in der Kontrollmatrix speziell angeordnet sein müssen. Es gibt in  $\mathbb{F}_{2^4}$  kein primitives Element, das die Elemente in der Reihenfolge erzeugt, wie sie für den [15,11]-Hamming-Code in seiner ursprünglichen Form benötigt wird.

Fügen wir der Kontrollmatrix von  $\text{BCH}_{\mathbb{F},2}(15,2)$  zwei weitere Zeilen hinzu, so gelangen wir zu  $\text{BCH}_{\mathbb{F},2}(15,3)$ . Die Kontroll- und die Generatormatrix dieses Codes sind in Abbildung 6.52 zu sehen, und Tabelle 6.7 listet die Codewörter auf.

Kontrollmatrix in Binärform	Reduzierte Kontrollmatrix	Übergang in den Orthogonalraum
$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$ <p style="text-align: center;">Generatormatrix in systematischer Form</p>

Abb. 6.51: Kontroll- und Generatormatrix von  $\text{BCH}_{\mathbb{F},2}(15,2)$

Mit der geleisteten Vorarbeit macht es jetzt keine Mühe mehr, die formale Definition des BCH-Codes zu verstehen. Wir brauchen unsere angestellten Überlegungen nur noch ein wenig zu verallgemeinern:



### Definition 6.10 (BCH-Code)

Es seien  $k$  und  $\delta$  zwei natürliche Zahlen,  $n = 2^k - 1$ ,  $\mathbb{F}$  ein endlicher Körper mit  $2^k$  Elementen und  $\alpha \in \mathbb{F}$  ein primitives Element. Dann ist die Binärdarstellung von

$$\begin{pmatrix} (\alpha^{n-1}) & \dots & (\alpha^3) & (\alpha^2) & (\alpha) & 1 \\ (\alpha^{n-1})^2 & \dots & (\alpha^3)^2 & (\alpha^2)^2 & (\alpha)^2 & 1 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ (\alpha^{n-1})^{2\delta} & \dots & (\alpha^3)^{2\delta} & (\alpha^2)^{2\delta} & (\alpha)^{2\delta} & 1 \end{pmatrix}$$

die Kontrollmatrix des Codes  $\text{BCH}_{\mathbb{F},\alpha}(n, \delta)$ .

Codewörter von $\text{BCH}_{\mathbb{F},2}(15,2)$							
0	0000000000000000	32	010000001110100	64	100000011101000	96	110000010011100
1	000000111010001	33	010000110100101	65	100000100111001	97	110000101001101
2	000001001110011	34	010001000000111	66	100001010011011	98	110001011101111
3	000001110100010	35	010001111010110	67	100001101001010	99	110001100111110
4	000010011100110	36	010010010010010	68	100010000001110	100	110010001111010
5	000010100110111	37	010010101000011	69	100010111011111	101	110010110101011
6	000011010010101	38	010011011100001	70	100011001111101	102	110011000001001
7	000011101000100	39	010011100110000	71	10001110101100	103	11001111011000
8	00010000011101	40	010100001101001	72	100100011110101	104	110100010000001
9	000100111001100	41	010100110111000	73	100100100100100	105	110100101010000
10	000101001101110	42	010101000011010	74	100101010000110	106	110101011110010
11	000101110111111	43	010101111001011	75	100101101010111	107	110101100100011
12	000110011111011	44	010110010001111	76	100110000010011	108	110110001100111
13	000110100101010	45	010110101011110	77	100110111000010	109	110110110110110
14	000111010001000	46	010111011111100	78	100111001100000	110	110111000010100
15	000111101011001	47	010111100101101	79	100111110110001	111	11011111000101
16	001000000111010	48	011000001001110	80	101000011010010	112	111000010100110
17	001000111101011	49	011000110011111	81	101000100000011	113	111000101110111
18	001001001001001	50	011001000111101	82	101001010100001	114	111001011010101
19	001001110011000	51	011001111101100	83	101001101110000	115	111001100000100
20	001010011011100	52	011010010101000	84	101010000110100	116	111010001000000
21	001010100001101	53	011010101111001	85	101010111100101	117	111010110010001
22	001011010101111	54	011011011011011	86	101011001000111	118	111011000110011
23	001011101111110	55	011011100001010	87	101011110010110	119	111011111100010
24	001100000100111	56	011100001010011	88	101100011001111	120	111100010111011
25	001100111110110	57	011100110000010	89	101100100011110	121	111100101101010
26	001101001010100	58	011101000100000	90	101101010111100	122	111101011001000
27	001101110000101	59	011101111110001	91	101101101101101	123	111101100011001
28	001110011000001	60	011110010110101	92	101110000101001	124	111110001011101
29	001110100010000	61	011110101100100	93	101110111111000	125	111110110001100
30	001111010110010	62	011111011000110	94	101111001011010	126	111111000101110
31	001111101100011	63	011111100010111	95	101111110001011	127	111111111111111

**Tab. 6.6:** Der BCH-Code und der ausgedünnte Hamming-Code sind äquivalent. Sie unterscheiden sich lediglich in der Anordnung der Bits innerhalb der Codewörter.

Wir wollen uns die einzelnen Elemente dieser Definition klarmachen. Die Konstruktion von  $\text{BCH}_{\mathbb{F},\alpha}(n, \delta)$  basiert auf einer Matrix mit  $2\delta$  Zeilen und  $2^k - 1$  Spalten, wobei die Matrixelemente aus einem endlichen Körper  $\mathbb{F}$  mit  $2^k$  Elementen stammen. Die Matrix enthält also

Kontrollmatrix in Binärform	Reduzierte Kontrollmatrix	Übergang in den Orthogonalraum
$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$ <p style="text-align: center;">Generatormatrix in systematischer Form</p>

Abb. 6.52: Kontroll- und Generatormatrix von  $\text{BCH}_{\mathbb{F},2}(15,3)$ 

genauso viele Spalten, wie es in  $\mathbb{F}_{2^k}$  von 0 verschiedene Elemente gibt. Da die Elemente in der ersten Zeile paarweise verschieden und ungleich 0 sein müssen, ist dies die maximal mögliche Spaltenanzahl. Damit ist auch die Aufgabe des primitiven Elements  $\alpha$  klar, das in dieser Definition vorkommt: Es wird dazu verwendet, die  $2^k - 1$  von 0 verschiedenen Körperelementen zu erzeugen.

Gehen der Körper  $\mathbb{F}$  und das primitive Element aus dem Kontext hervor, so schreiben wir im Folgenden kürzer  $\text{BCH}(n, \delta)$  anstatt  $\text{BCH}_{\mathbb{F}, \alpha}(n, \delta)$ .

Beachten Sie, dass BCH-Codes binäre Codes sind. Das bedeutet, dass nicht die in Definition 6.10 angegebene Matrix die Kontrollmatrix des BCH-Codes ist, sondern dessen Binärrepräsentation. Wir erhalten diese aus der angegebenen Matrix, indem wir die Elemente des Körpers  $\mathbb{F}$ ,

Codewörter von $\text{BCH}_{\mathbb{F},2}(15,3)$							
0	0000000000000000	8	010001111010110	16	100001010011011	24	110000101001101
1	000010100110111	9	010011011100001	17	100011110101100	25	110010001111010
2	000101001101110	10	010100110111000	18	100100011110101	26	110101100100011
3	000111101011001	11	010110010001111	19	100110111000010	27	110111000010100
4	001000111101011	12	011001000111101	20	101001101110000	28	111000010100110
5	001010011011100	13	011011100001010	21	101011001000111	29	111010110010001
6	001101110000101	14	011100001010011	22	101100100011110	30	111101011001000
7	001111010110010	15	011110101100100	23	101110000101001	31	111111111111111

**Tab. 6.7:**  $\text{BCH}_{\mathbb{F},2}(15,3)$  umfasst 32 Codewörter und ist 3-fehlererkennend. Er ist zu dem ausgedünnten Hamming-Code aus Tabelle 6.5 äquivalent.

genau wie in unserem Beispiel, spaltenweise durch ihre Binärrepräsentation ersetzen. Hierdurch bleibt die Anzahl der Spalten unverändert, die Anzahl der Zeilen vervielfacht sich dagegen um den Faktor  $k$ . Genau dies konnten wir auch in unserem Beispiel beobachten. Da wir dort in dem Erweiterungskörper  $\mathbb{F}_{2^4}$  gerechnet haben, besaß die binäre Kontrollmatrix viermal so viele Zeilen wie die entsprechende Matrix über  $\mathbb{F}_{2^4}$ .

## BCH-Codes und Polynome

Wir haben gesehen, dass die Konstruktion von BCH-Codes auf einem Übergang von dem Restklassenkörper  $\mathbb{Z}_2$  in einen seiner Erweiterungskörper  $\mathbb{F}_{2^k}$  besteht, und wir wissen aus Abschnitt 2.4, dass sich hinter den endlichen Körpern in Wirklichkeit Polynomringe verbergen. Damit hat uns der Übergang in den Erweiterungskörper indirekt in die Welt der Polynome geführt, auch wenn dies auf den ersten Blick gar nicht ersichtlich war.

In diesem Abschnitt wollen wir den Zusammenhang zwischen Polynomen auf der einen Seite und den BCH-Codes auf der anderen Seite ein wenig tiefer beleuchten. Dabei werden wir die Erkenntnis gewinnen, dass beide viel engmaschiger verbunden sind, als es unsere bisher angestellten Überlegungen vermuten lassen.

Im Folgenden werden wir Codewörter, wie wir es aus Abschnitt 2.4 bereits gewohnt sind, mit Polynomen identifizieren. Konkret entspricht

das Codewort

$$\mathbf{v} = v_0 v_1 v_2 \dots v_{n-1} \quad (6.31)$$

dem Polynom

$$v(x) = v_0 x^{n-1} + v_1 x^{n-2} + \dots + v_{n-2} x + v_{n-1} = \sum_{j=0}^{n-1} v_j x^{n-1-j}$$

Aus den oben angestellten Überlegungen wissen wir, dass die Multiplikation der Kontrollmatrix mit einem transponierten Codewortvektor den Nullvektor ergibt:

$$H \cdot \mathbf{v}^T = \mathbf{0} \quad (6.32)$$

Im Falle des BCH-Codes ist  $H$  eine binäre Matrix. Das bedeutet, dass das Skalarprodukt von  $\mathbf{v}$  mit einer beliebigen Zeile von  $H$ , gerechnet in  $\mathbb{Z}_2$ , gleich 0 ist. Genauso gut können wir diesen Zusammenhang mithilfe der in Definition 6.10 angegebenen Matrix formulieren. Die Elemente dieser Matrix stammen aus dem Erweiterungskörper  $\mathbb{F}$  und wir können aus (6.32) den folgenden Schluss ziehen:

$$\begin{aligned} v_0(\alpha^{n-1}) &+ \dots + v_{n-3}(\alpha^2) + v_{n-2}(\alpha) + v_{n-1} = 0 \\ v_0(\alpha^{n-1})^2 &+ \dots + v_{n-3}(\alpha^2)^2 + v_{n-2}(\alpha)^2 + v_{n-1} = 0 \\ v_0(\alpha^{n-1})^3 &+ \dots + v_{n-3}(\alpha^2)^3 + v_{n-2}(\alpha)^3 + v_{n-1} = 0 \\ &\vdots \\ v_0(\alpha^{n-1})^{2\delta} &+ \dots + v_{n-3}(\alpha^2)^{2\delta} + v_{n-2}(\alpha)^{2\delta} + v_{n-1} = 0 \end{aligned}$$

Dies können wir kürzer auch so aufschreiben:

$$\sum_{j=0}^{n-1} v_j (\alpha^{n-1-j})^i = 0 \quad (1 \leq i \leq 2\delta) \quad (6.33)$$

Wegen

$$\sum_{j=0}^{n-1} v_j (\alpha^{n-1-j})^i = \sum_{j=0}^{n-1} v_j (\alpha^i)^{n-1-j}$$

ist (6.33) das Gleiche wie

$$v(\alpha^i) = 0 \quad (1 \leq i \leq 2\delta)$$

Mit anderen Worten:  $v$  ist genau dann ein BCH-Codewort, wenn das Polynom  $v(x)$  an den Stützstellen  $\alpha, \alpha^2, \dots, \alpha^{2\delta}$  eine Nullstelle aufweist.

Dies führt uns direkt zu einer alternativen Definition der BCH-Codes:

**Definition 6.11 (BCH-Code)**

Es seien  $k$  und  $\delta$  zwei natürliche Zahlen,  $n = 2^k - 1$ ,  $\mathbb{F}$  ein endlicher Körper mit  $2^k$  Elementen und  $\alpha \in \mathbb{F}$  ein primitives Element. Dann ist

$$\text{BCH}_{\mathbb{F}, \alpha}(n, \delta) := \{(v_0, \dots, v_{n-1}) \in \mathbb{Z}_2^n \mid v(\alpha) = \dots = v(\alpha^{2\delta}) = 0\}$$

Hierin steht  $v(x)$  für das Polynom  $v_0x^{n-1} + \dots + v_{n-1} \in \mathbb{F}[x]$ .



Pierre de Fermat  
(1607 – 1665)

Achten Sie darauf, diese Definition richtig zu lesen! Auf der einen Seite sind BCH-Codes binäre Codes, d.h., die Codewörter  $(v_0, \dots, v_{n-1})$  setzen sich aus Elementen der Menge  $\mathbb{Z}_2$  zusammen. Auf der anderen Seite ist das Polynom  $p(x)$ , das die Codewörter charakterisiert, ein Polynom über dem Erweiterungskörper. Bisher hatten wir ein Codewort  $v \in \mathbb{Z}_2^n$  immer mit einem Polynom aus der Menge  $\mathbb{Z}_2[x]$  assoziiert.

**Minimalpolynome**

In den folgenden Überlegungen spielen bestimmte Polynome eine Rolle, die in der Algebra als *Minimalpolynom* bezeichnet werden. Um zu verstehen, was sich hinter diesem Begriff verbirgt, betrachten wir den endlichen Körper  $\mathbb{F}_{2^4}$ , den wir in unserem Beispiel verwendet haben, sowie folgendes Polynom:

$$x^{15} - 1$$

Dieses Polynom lässt sich auf unterschiedliche Weisen faktorisieren, je nachdem, ob wir seine Koeffizienten als Elemente von  $\mathbb{Z}_2$  oder als Elemente des Erweiterungskörpers  $\mathbb{F}_{2^4}$  betrachten. Im zweiten Fall können wir auf ein berühmtes Ergebnis des französischen Mathematikers Pierre de Fermat zurückgreifen (Abbildung 6.53). Es besagt, dass in einem endlichen Körper  $\mathbb{F}$  mit  $n$  Elementen die folgende Zerlegung gilt:

$$x^{n-1} - 1 = \prod_{\substack{\alpha \in \mathbb{F} \\ \alpha \neq 0}} (x - \alpha) \quad (6.34)$$

Das bedeutet, dass die Nullstellen des Polynoms  $x^{n-1} - 1$  genau die von 0 verschiedenen Körperelemente sind. Bezogen auf unseren Beispieldkörper  $\mathbb{F}_{2^4}$  erhalten wir daraus die folgende Zerlegung:

$$x^{15} - 1 = (x - 1)(x - 2)(x - 3) \cdot \dots \cdot (x - 15) \quad (6.35)$$

**Abb. 6.53:** Dem französischen Mathematiker Pierre de Fermat verdanken wir zahlreiche Beiträge zur Zahlentheorie. Zu großer Bekanntheit gelangte er durch seine 1637 aufgestellte Behauptung, die Gleichung

$$a^n + b^n = c^n$$

habe für  $n > 2$  keine Lösungen in den positiven ganzen Zahlen. Im Laufe der Zeit avancierte seine Vermutung zu einer der großen offenen Fragen der Mathematik, die sich über dreihundert Jahre lang allen Versuchen entzog, sie zu beweisen. Erst im Jahr 1995 konnte der Brit Andrew Wiles einen lückenlosen Beweis für den *großen Fermat'schen Satz* vorbringen und damit ein bewegtes Kapitel der Mathematikgeschichte schließen [83, 106].

Neben dem großen Fermat'schen Satz existiert der sogenannte *kleine Fermat'sche Satz*. Dieser besagt, dass jede ganze Zahl  $a$  und jede Primzahl  $p$  die Beziehung

$$a^p \equiv a \pmod{p}$$

erfüllt. Der kleine Fermat'sche Satz ist eng mit unserer Gleichung (6.34) verwandt und lässt sich auch mit ähnlichen Mitteln beweisen.

$x$	$m_1(x)$	$m_2(x)$	$m_3(x)$	$m_4(x)$	$m_5(x)$
1	0	1	1	1	1
2	3	7	12	0	10
3	2	7	10	0	12
4	5	6	15	0	8
5	4	6	8	0	15
6	7	0	7	1	6
7	6	0	6	1	7
8	9	5	0	6	4
9	8	5	4	6	0
10	11	3	0	7	2
11	10	3	2	7	0
12	13	2	0	7	3
13	12	2	3	7	0
14	15	4	5	6	0
15	14	4	0	6	5

**Tab. 6.8:** Jedes von 0 verschiedene Körperelement  $\alpha$  ist eine Nullstelle von genau einem der Polynome  $m_1(x)$  bis  $m_5(x)$ . Dasjenige Polynom  $m_i(x)$  mit  $m_i(\alpha) = 0$  ist das Minimalpolynom von  $\alpha$ .

Auf der anderen Seite können wir  $x^{15} - 1$  auch über dem Körper  $\mathbb{Z}_2$  faktorisieren. Wir erhalten dann

$$x^{15} - 1 = m_1 \cdot m_2 \cdot m_3 \cdot m_4 \cdot m_5 \quad \text{mit} \quad (6.36)$$

$$m_1 = (x+1)$$

$$m_2 = (x^2+x+1)$$

$$m_3 = (x^4+x^3+x^2+x+1)$$

$$m_4 = (x^4+x+1)$$

$$m_5 = (x^4+x^3+1)$$

Die Beziehungen (6.35) und (6.36) sind eng miteinander verflochten. Es gilt, dass jede Nullstelle von (6.35), also jedes von 0 verschiedene Element  $\alpha \in \mathbb{F}_{2^4}$ , auch eine Nullstelle von genau einem der 5 Faktorpolynome  $m_1$  bis  $m_5$  sein muss. Das entsprechende Polynom ist das *Minimalpolynom* von  $\alpha$ .

Tabelle 6.8 gibt für den Körper  $\mathbb{F}_{2^4}$  Auskunft darüber, wie die Nullstellen verteilt sind. Wir können dort für jedes Körperelement ablesen, welches der fünf Polynome  $m_1, \dots, m_5$  das Minimalpolynom ist. Beispielsweise ist  $m_1$  das Minimalpolynom des Einselementes 1,  $m_2$  das Minimalpolynom der Elemente 6 und 7 und  $m_3$  das Minimalpolynom der Elemente 8, 10, 12 und 15. Die verbleibenden beiden Polynome  $m_4$  und  $m_5$  sind die Minimalpolynome der Elemente 2, 3, 4 und 5 bzw. 9, 11, 13 und 14.

Minimalpolynome tragen ihren Namen zu Recht. Formal lässt sich das Minimalpolynom eines Elements  $\alpha$  als das kleinstmögliche, von null verschiedene Polynom aus der Menge  $\mathbb{Z}_2[x]$  definieren, das an der Stelle  $\alpha$  eine Nullstelle hat.

Hieraus folgt eine wichtige Teilbarkeitsbeziehung, die wir weiter unten benötigen werden.

### Satz 6.10

Ist  $m(x) \in \mathbb{Z}_2[x]$  das Minimalpolynom von  $\alpha \in \mathbb{F}_{2^k}$  und  $p(x) \in \mathbb{Z}_2[x]$  ein Polynom mit  $p(\alpha) = 0$ , so ist  $p(x)$  ohne Rest durch  $m(x)$  teilbar.

*Beweis:* Aus dem Abschnitt über die Polynomdivision wissen wir, dass sich  $p(x)$  in der Form  $p(x) = q(x)m(x) + r(x)$  mit  $\deg r(x) < \deg m(x)$  darstellen lässt. Aus  $p(\alpha) = 0$  folgt  $q(\alpha)m(\alpha) + r(\alpha) = 0$ , und wegen

$m(\alpha) = 0$  ist dann auch  $r(\alpha) = 0$ . Da der Grad von  $r(x)$  kleiner ist als der Grad von  $m(x)$ , folgt aus der Minimalität von  $m(x)$ , dass  $r(x)$  das Nullpolynom sein muss;  $p(x)$  ist also ohne Rest durch  $m(x)$  teilbar.  $\square$

Als Nächstes wollen wir klären, warum Minimalpolynome für uns so wichtig sind. Hierzu werfen wir erneut einen Blick auf Definition 6.11. Dort haben wir festgelegt, dass  $v$  genau dann ein Codewort von  $\text{BCH}_{\mathbb{F}, \alpha}(n, \delta)$  ist, wenn das Polynom  $v(x)$  an den Stützstellen  $\alpha, \alpha^2, \dots, \alpha^{2\delta}$  eine Nullstelle aufweist.

Betrachten wir beispielsweise den endlichen Körpers  $\mathbb{F}_{2^4}$ , wählen für  $\alpha$  das primitive Element 2 und setzen  $\delta = 2$ , so sind  $\alpha, \dots, \alpha^{2\delta}$  die Elemente

$$2, 4, 8, 3$$

Für diese 4 Zahlen sind  $m_3$  und  $m_4$  die Minimalpolynome (vgl. Tabelle 6.8). Bilden wir das Produkt dieser Polynome, so erhalten wir

$$g(x) = (x^4 + x^3 + x^2 + x + 1)(x^4 + x + 1) = x^8 + x^7 + x^6 + x^4 + 1$$

Was können wir über dieses Polynom aussagen? Um der Bedeutung von  $g(x)$  auf die Schliche zu kommen, nehmen wir an,  $v$  sei ein Vektor, von dem wir nicht wissen, ob er ein Codewort ist oder nicht. Wir unterscheiden zwei Fälle:

■ 1. Fall:  $v(x)$  ist ein Vielfaches von  $g(x)$  mit  $\deg v(x) < n$

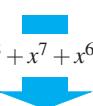
Da  $g(x)$  das Polynom  $v(x)$  ohne Rest teilt, lässt sich  $v(x)$  in der Form  $g(x)h(x)$  darstellen. Da die Minimalpolynome von  $\alpha, \alpha^2, \dots, \alpha^{2\delta}$  alle Faktoren von  $g(x)$  sind, ist  $g(\alpha^i)$  für alle  $i$  mit  $1 \leq i \leq 2\delta$  gleich 0. Dann ist aber auch  $v(\alpha^i)$  für alle  $i$  mit  $1 \leq i \leq 2\delta$  gleich 0, und das bedeutet, dass  $v(x)$  die Polynomdarstellung eines Codeworts ist.

■ 2. Fall:  $v(x)$  ist die Polynomdarstellung eines Codeworts

In diesem Fall ist  $v(\alpha) = v(\alpha^2) = \dots = v(\alpha^{2\delta}) = 0$ . Das bedeutet nach Satz 6.10, dass das Minimalpolynom von  $\alpha^i$  für  $1 \leq i < 2\delta$  ein Teiler von  $v(x)$  ist. Minimalpolynome sind irreduzibel und paarweise verschieden, sodass auch das Produkt  $g(x)$  das Polynom  $v(x)$  ohne Rest teilt.

Beide Fälle zusammen liefern uns als Ergebnis den folgenden Satz:

1 0 0 0 0 0 0	$x^{14} \bmod g(x)$
0 1 0 0 0 0 0	$x^{13} \bmod g(x)$
0 0 1 0 0 0 0	$x^{12} \bmod g(x)$
0 0 0 1 0 0 0	$x^{11} \bmod g(x)$
0 0 0 0 1 0 0	$x^{10} \bmod g(x)$
0 0 0 0 0 1 0	$x^9 \bmod g(x)$
0 0 0 0 0 0 1	$x^8 \bmod g(x)$

$$g(x) = x^8 + x^7 + x^6 + x^4 + 1$$


$p(x)$	$p(x) \bmod g(x)$
$x^{14}$	$x^7 + x^6 + x^5 + x^3$
$x^{13}$	$x^6 + x^5 + x^4 + x^2$
$x^{12}$	$x^5 + x^4 + x^3 + x$
$x^{11}$	$x^4 + x^3 + x^2 + 1$
$x^{10}$	$x^7 + x^6 + x^5 + x^2 + x$
$x^9$	$x^6 + x^5 + x^4 + x + 1$
$x^8$	$x^7 + x^6 + x^4 + 1$



1 0 0 0 0 0 0 0 1 1 1 0 1 0 0 0
0 1 0 0 0 0 0 0 1 1 1 0 1 0 0 0
0 0 1 0 0 0 0 0 0 1 1 1 0 1 0 0
0 0 0 1 0 0 0 0 0 0 1 1 1 0 1 0 1
0 0 0 0 1 0 0 1 1 1 0 0 1 1 0 0
0 0 0 0 0 1 0 0 1 1 1 0 0 0 1 1
0 0 0 0 0 0 1 1 1 0 1 0 0 0 1 1

**Abb. 6.54:** Aus dem Generatorpolynom lässt sich ohne Umwege die Generatormatrix in systematischer Form erzeugen, hier demonstriert am Beispiel von  $\text{BCH}(15,2)$ .

### Satz 6.11

Sei  $g(x)$  das von doppelten Faktoren bereinigte Produkt der Minimalpolynome von  $\alpha, \alpha^2, \dots, \alpha^{2\delta}$  und  $v \in \mathbb{Z}_2^n$ . Dann gilt:

$$v \in \text{BCH}_{\mathbb{F}, \alpha}(n, \delta) \Leftrightarrow g(x) \mid v(x)$$

Dieser Satz deckt die wahre Bedeutung des Polynoms  $g(x)$  auf: Es ist das Generatorpolynom, das den BCH-Code erzeugt.

Wäre uns vorab bekannt gewesen, dass sich die BCH-Codes mithilfe von Polynomen generieren lassen, so hätten wir das Generatorpolynom auch direkt aus Tabelle 6.6 ablesen können. Wie wir das Generatorpolynom finden, wissen wir aus Abschnitt 6.4.3. Wir müssen die Codewörter lediglich als Binärzahlen interpretieren und die Codetabelle nach dem kleinsten Wert durchsuchen. Das Codewort, das wir auf diese Weise finden, codiert die Koeffizienten des Generatorpolynoms. Führen wir die Suche auf den Codewörtern in Tabelle 6.6 aus, so ist das Ergebnis folgendes Codewort:

$$000000111010001$$

Übersetzen wir die Bitsequenz in ihre Polynomdarstellung zurück, so entsteht das folgende Generatorpolynom:

$$g(x) = x^8 + x^7 + x^6 + x^4 + 1 \quad (6.37)$$

Wiederholen wir die Suche in der Codetabelle 6.7, so finden wir dort das nachstehende Codewort als kleinstes Element:

$$000010100110111$$

Nach allem, was wir bisher wissen, muss demnach

$$x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1 \quad (6.38)$$

das Generatorpolynom des dort abgebildeten Codes sein. Wir wollen versuchen, das Generatorpolynom ohne Kenntnis der Codetabelle zu ermitteln, indem wir es als Produkt von Minimalpolynomen berechnen.

Um herauszubekommen, welche Minimalpolynome in das Produkt aufgenommen werden müssen, bestimmen wir zunächst die Elemente  $\alpha, \dots, \alpha^{2\delta}$ . Für den Code aus Tabelle 6.7 ist  $\delta = 3$ , sodass wir die folgenden 6 Elemente erhalten:

$$2, 4, 8, 3, 6, 12$$

Die Minimalpolynome der beiden zusätzlichen Elemente 6 und 12 sind  $m_2$  und  $m_3$ . Das bedeutet, dass wir das Generatorpolynom als Produkt von drei Minimalpolynomen erhalten, den Polynomen  $m_2$ ,  $m_3$  und  $m_4$ :

$$g(x) = (x^2 + x + 1)(x^4 + x^3 + x^2 + x + 1)(x^4 + x + 1)$$

Multiplizieren wir die Produkte aus, so erhalten wir mit

$$g(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1 \quad (6.39)$$

exakt das Polynom (6.38), das wir anhand der Codetabelle vorhergesagt haben.

Wenn Sie einen Blick zurück auf Seite 386 werfen, werden Sie sich sicher daran erinnern, dass die Generatormatrix mit wenigen Rechenschritten aus dem Generatorpolynom abgeleitet werden kann. Ist  $r$  der Grad des Generatorpolynoms, so müssen wir lediglich die Polynome  $x^{n-1}, \dots, x^r$  durch das Generatorpolynom dividieren und die Reste rechts neben der Einheitsmatrix notieren. Führen wir diese Rechnung, wie es in den Abbildungen 6.54 und 6.55 gezeigt ist, für unsere beiden Beispieldenom (6.37) bzw. (6.39) aus, so erhalten wir die gleichen Generatormatrizen, die wir in den Abbildungen 6.51 und 6.52 aus der Kontrollmatrix gewonnen haben. Alle Puzzlestücke passen perfekt zusammen!

Zum Schluss wollen wir nochmals einen Blick auf Gleichung (6.36) werfen. In  $\mathbb{Z}_2$  ist  $x^{15} - 1$  das Gleiche wie  $x^{15} + 1$  und daraus folgt aus Satz 6.5, dass der entsprechende BCH-Code zyklisch ist. Anhand der Codetabellen 6.6 und 6.7 lässt sich diese Eigenschaft konkret überprüfen. Greifen wir willkürlich ein Codewort heraus und rotieren es um eine beliebige Anzahl an Positionen, so entsteht eine Bitsequenz, die ebenfalls wieder ein Codewort ist.

Sind vielleicht nicht nur unsere Beispiele, sondern alle BCH-Codes zyklisch? Die Antwort lautet Ja, und mit unserem erworbenen Wissen können wir dies auch formal begründen. Verantwortlich hierfür ist die Tatsache, dass das Generatorpolynom aus dem Produkt von Minimalpolynomen gebildet wird, und das Produkt aller Minimalpolynome gleich  $x^n + 1$  ist. Folglich ist auch das Generatorpolynom immer ein Teiler von  $x^n + 1$ . Damit sind wir in der Lage, diesen Abschnitt mit einem bedeutsamen Satz zu beenden:

### Satz 6.12

Alle BCH-Codes sind zyklisch.

1 0 0 0 0	$x^{14} \bmod g(x)$
0 1 0 0 0	$x^{13} \bmod g(x)$
0 0 1 0 0	$x^{12} \bmod g(x)$
0 0 0 1 0	$x^{11} \bmod g(x)$
0 0 0 0 1	$x^{10} \bmod g(x)$

$$g(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$$


$p(x)$	$p(x) \bmod g(x)$
$x^{14}$	$x^9 + x^7 + x^4 + x^3 + x + 1$
$x^{13}$	$x^9 + x^8 + x^7 + x^6 + x^4 + x^2 + x$
$x^{12}$	$x^8 + x^7 + x^6 + x^5 + x^3 + x + 1$
$x^{11}$	$x^9 + x^6 + x^5 + x^3 + x^2 + x$
$x^{10}$	$x^8 + x^5 + x^4 + x^2 + x + 1$



1 0 0 0 0 1 0 1 0 0 1 1 0 1 1
0 1 0 0 0 1 1 1 1 0 1 0 1 1 0
0 0 1 0 0 0 1 1 1 1 0 1 0 1 1
0 0 0 1 0 1 0 0 1 1 0 1 1 1 0
0 0 0 0 1 0 1 0 0 1 1 0 1 1 1

**Abb. 6.55:** Berechnung der Generatormatrix von  $\text{BCH}(15,3)$

### 6.4.5 Reed-Solomon-Codes

„Consider the polynomial  $P(x)$  of degree  $m - 1$

$$P(x) = a_0 + a_1x + \dots + a_{m-1}x^{m-1}$$

where  $a_i \in K$  and  $m < 2^n$ . Code  $E$  is the mapping of the  $m$ -tuple  $(a_0, a_1, \dots, a_{m-1})$  into the  $2^n$ -tuple  $(P(0), P(\alpha), P(\alpha^2), \dots, P(1))$ ; this  $m$ -tuple might be some encoded message and the corresponding  $2^n$ -tuple is to be transmitted.“

I. S. Reed, G. Solomon, 1960 [75]

Im Jahr 1960 publizierten Irving S. Reed und Gustave Solomon eine Arbeit mit dem Titel „Polynomial Codes Over Certain Finite Fields“ [75], die wir rückblickend als ein Meilenstein der Codierungstheorie ansehen dürfen. Im Kern basieren die Codes, die wir heute – ihren Entdeckern zu Ehren – als *Reed-Solomon-Codes* bezeichnen, auf einer verblüffend einfachen Idee: Sie nutzen aus, dass ein Polynom vom Grad  $k - 1$  durch  $k$  Stützstellen eindeutig bestimmt ist. Das bedeutet, dass jeweils  $k$  Nachrichtensymbole dazu verwendet werden können, um ein Polynom  $p(x)$  mit  $\deg p(x) < k$  zu fixieren. Wird das Polynom durch die Hinzunahme weiterer Stützpunkte überapproximiert, so kann es der Empfänger selbst dann noch rekonstruieren, wenn einige davon während der Übertragung verfälscht wurden oder verloren gegangen sind. Sobald das Polynom korrekt wiederhergestellt ist, kann der Empfänger daraus die Nachricht extrahieren.

Wir wollen die einzelnen Schritte der Reed-Solomon-Codierung genauer ansehen und mit der Herleitung eines bekannten Ergebnisses aus der Theorie der Polynome beginnen. Hierzu sei  $\mathbb{F}$  ein Körper,  $x_0, \dots, x_{k-1}$  sowie  $y_0, \dots, y_{k-1}$  seien beliebige Elemente aus  $\mathbb{F}$ , und  $p(x) \in \mathbb{F}[x]$  sei ein Polynom mit  $\deg p(x) < k$ ;  $p(x)$  hat also die folgende Form:

$$p(x) = a_0x^{k-1} + \dots + a_{k-3}x^2 + a_{k-2}x + a_{k-1}$$

Wir wollen die Frage beantworten, unter welchen Bedingungen wir die Koeffizienten  $a_0, \dots, a_{k-1}$  so wählen können, dass  $y_i$  dem Funktionswert an der Stützstelle  $x_i$  entspricht, dass also Folgendes gilt:

$$p(x_0) = y_0$$

$$p(x_1) = y_1$$

...

$$p(x_{k-1}) = y_{k-1}$$

Etwas ausführlicher aufgeschrieben lauten diese Gleichungen so:

$$\begin{aligned} a_0x_0^{k-1} + \dots + a_{k-3}x_0^2 + a_{k-2}x_0 + a_{k-1} &= y_0 \\ a_0x_1^{k-1} + \dots + a_{k-3}x_1^2 + a_{k-2}x_1 + a_{k-1} &= y_1 \\ &\dots \\ a_0x_{k-1}^{k-1} + \dots + a_{k-3}x_{k-1}^2 + a_{k-2}x_{k-1} + a_{k-1} &= y_{k-1} \end{aligned}$$

Dies wiederum ist dasselbe wie

$$\begin{pmatrix} x_0^{k-1} & \dots & x_0^2 & x_0 & 1 \\ x_1^{k-1} & \dots & x_1^2 & x_1 & 1 \\ \vdots & \ddots & \vdots & \vdots & \\ x_{k-1}^{k-1} & \dots & x_{k-1}^2 & x_{k-1} & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{k-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{k-1} \end{pmatrix}$$

Auf der linken Seite steht eine um  $90^\circ$  nach rechts rotierte Vandermonde-Matrix, wie sie in der Randspalte auf Seite 397 abgedruckt ist.

Aus der Determinantenformel folgt, dass die Zeilen der Matrix genau dann linear unabhängig sind, wenn die Stützstellen  $x_0, \dots, x_{k-1}$  paarweise verschieden sind; in diesem, und nur in diesem Fall ist das Gleichungssystem eindeutig lösbar. In Satzform lautet unser Ergebnis so:

### Satz 6.13

Es sei  $\mathbb{F}$  ein Körper und  $x_0, \dots, x_{k-1}, y_0, \dots, y_{k-1} \in \mathbb{F}$ . Sind die Stützstellen  $x_0, \dots, x_{k-1}$  paarweise verschieden, dann existiert ein eindeutig bestimmtes Polynom  $p(x) \in \mathbb{F}[x]$  mit  $\deg p(x) < k$  und

$$p(x_i) = y_i \quad (0 \leq i < k)$$

Abbildung 6.56 demonstriert die Polynomapproximation anhand dreier Beispiele. Im ersten Beispiel erhalten wir als Ergebnis ein eindeutig bestimmtes Polynom vom Grad 1, im zweiten ein Polynom vom Grad 2 und im dritten ein Polynom vom Grad 3.

#### 6.4.5.1 Codierung

##### Interpolationscodierung

Die Tatsache, dass  $k$  Stützstellen eindeutig ein Polynom  $p(x)$  mit  $\deg p(x) < k$  bestimmen, können wir in der folgenden Weise ausnutzen:

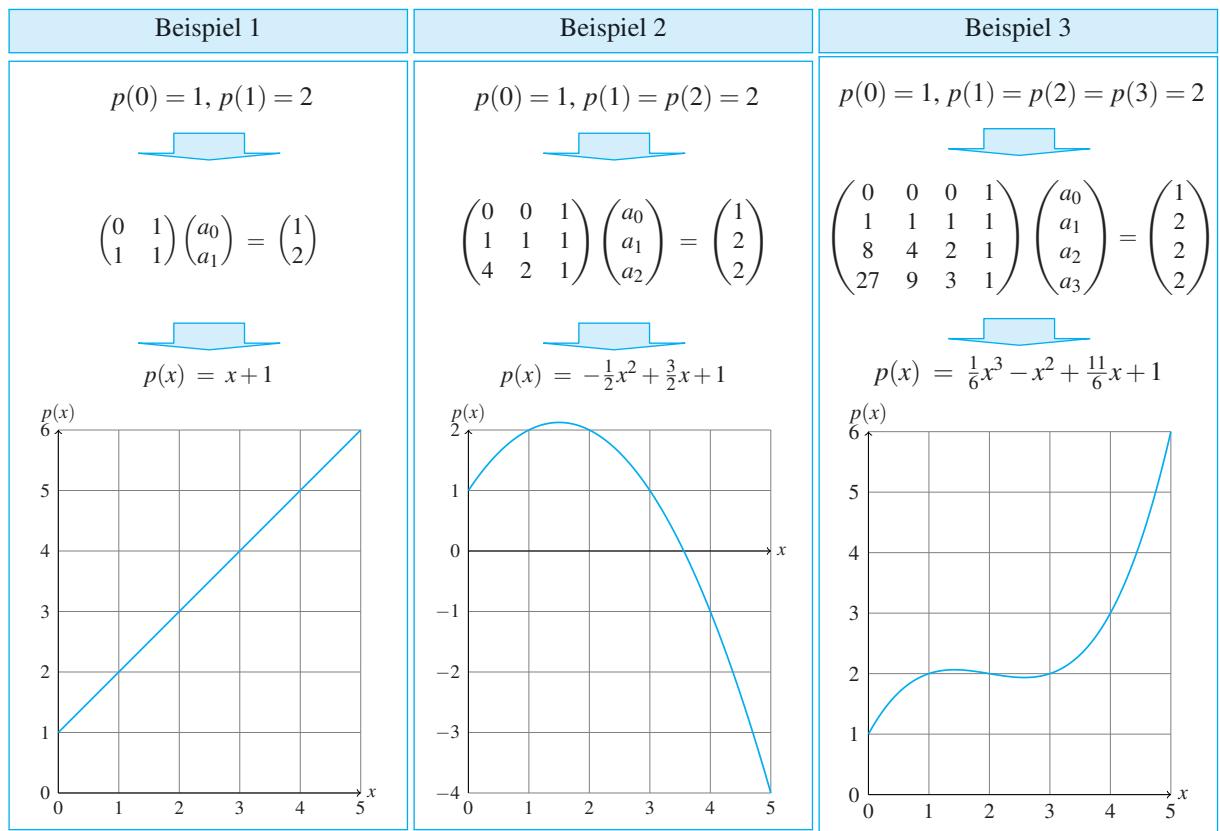


Abb. 6.56: Polynomapproximation in  $\mathbb{Q}[x]$

Um beispielsweise die Nachricht

$$\mathbf{u} = (1 \quad 2)$$

zu übertragen, bestimmen wir zunächst ein Polynom  $p(x)$  mit dem Grad 1, das an den Stützstellen 0 und 1 die zu übertragenden Werte annimmt; es soll also gelten:

$$\begin{aligned} p(0) &= 1 \\ p(1) &= 2 \end{aligned}$$

Nach Satz 6.13 existiert ein eindeutig bestimmtes Polynom mit dieser Eigenschaft. Wählen wir die Koeffizienten aus der Menge  $\mathbb{Q}$  der rationalen Zahlen, so finden wir das Polynom unter den Beispielen in Abbildung 6.56 wieder. Es ist das Polynom  $p(x) = x + 1$ , das in der linken Spalte abgebildet ist.

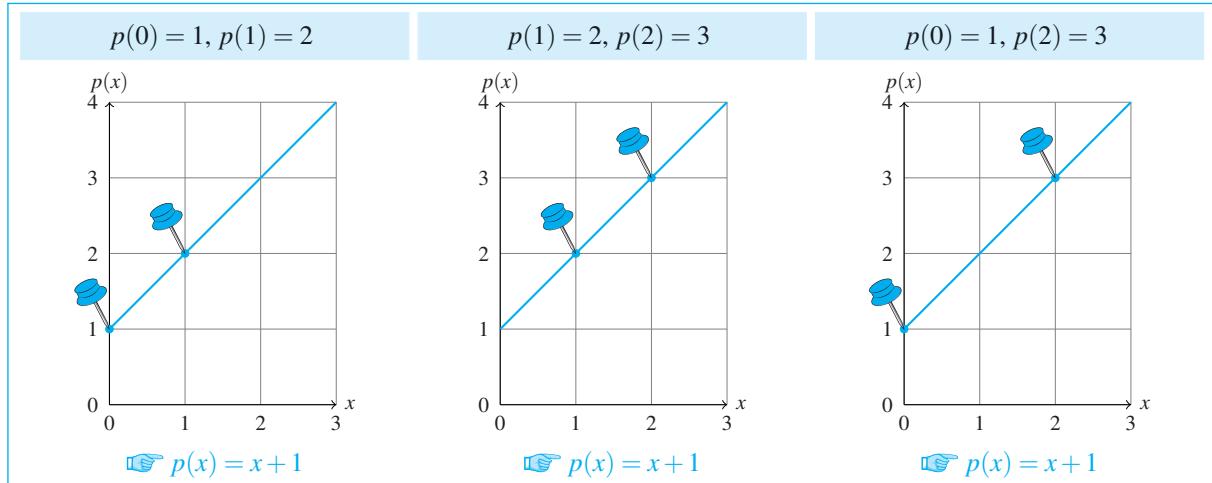


Abb. 6.57: Zwei der drei Stützpunkte reichen aus, um die Geradengleichung eindeutig zu rekonstruieren.

Jetzt bestimmen wir eine weitere Stützstelle und überapproximieren hierdurch das Polynom. Wegen  $p(2) = 3$  wird die Nachricht  $u$  auf das folgende Codewort abgebildet:

$$v = c(u) = (1 \ 2 \ 3)$$

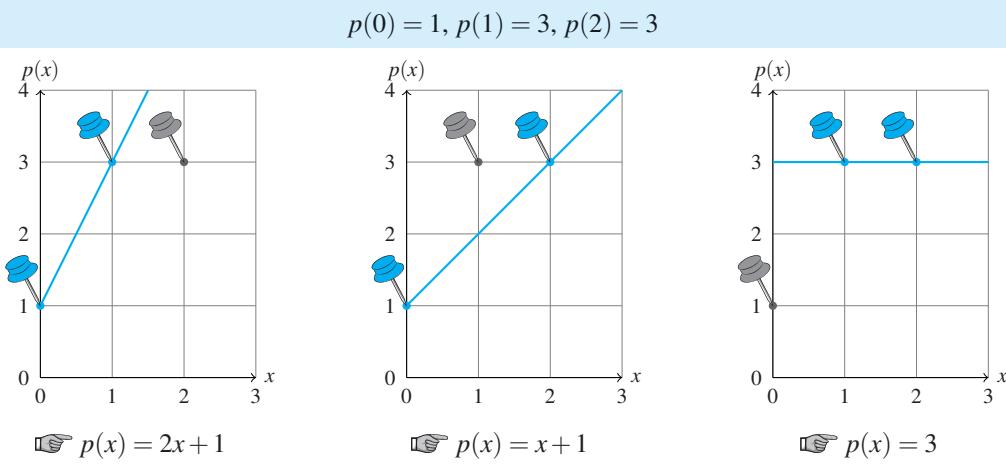
Als Nächstes wollen wir untersuchen, wie sich der Code im Falle eines Übertragungsfehlers verhält, und dabei akribisch zwischen zwei verschiedenen Fehlerszenarien unterscheiden: dem Verlust von Datenpaketen und der Verfälschung von Datenpaketen.

#### ■ Verlust von Datenpaketen

In unserem Beispiel haben wir die Gerade  $x + 1$  durch drei Stützpunkte beschrieben. Kennt der Empfänger zwei davon, so kann er die Geradengleichung rekonstruieren (Abbildung 6.57). Mit der Gleichung in Händen lässt sich das verloren gegangene Datenpaket problemlos ausrechnen, indem die Funktion  $x + 1$  an der betreffenden Stützstelle ausgewertet wird.

#### ■ Verfälschung von Datenpaketen

Im Gegensatz zu einem Verlust eines Datenpakets bedeutet eine Verfälschung, dass wir nicht wissen, ob ein empfangenes Datenpaket korrekt ist oder nicht. Unter der Annahme, dass höchstens ein Datenpaket verfälscht wurde, erreichen den Empfänger in unserem Beispiel drei Werte, von denen mindestens zwei auf der Geraden  $x + 1$  liegen.



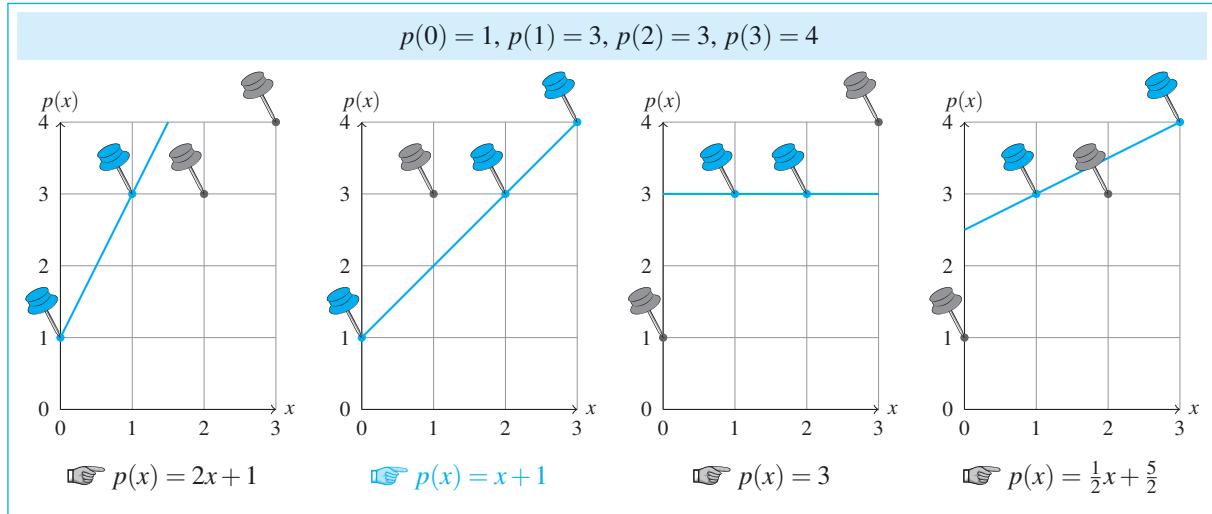
**Abb. 6.58:** Der Empfänger kann in diesem Beispiel lediglich feststellen, dass ein Übertragungsfehler aufgetreten ist. Bestimmen kann er das defekte Datenpaket nicht.

Wir nehmen an, in der Nachricht  $(1 \ 2 \ 3)$  wird das zweite Datenpaket falsch übertragen und anstelle des korrekten Werts 2 der Wert 3 empfangen. Abbildung 6.58 zeigt, dass der Empfänger in diesem Fall zweifelsfrei erkennen kann, dass die drei Punkte keine gemeinsame Gerade bilden, es gibt aber mehrere Möglichkeiten, jeweils zwei Punkte mit einer Linie zu verbinden. Der Empfänger kann somit nicht entscheiden, welches der drei Datenpakete fehlerhaft ist. Kurzum: Der Code ist 1-fehlererkennend, aber nicht 1-fehlerkorrigierend.

Um einen fehlerkorrigierenden Code zu erhalten, reicht das Hinzufügen einer einzelnen Stützstelle offenbar nicht aus. Wir wollen deshalb untersuchen, ob sich dies durch die Hinzunahme einer weiteren, in unserem Beispiel der Stützstelle  $x = 3$ , verändert. In diesem Fall weisen die Codewörter die Länge 4 auf, und unsere Beispielnachricht wird durch das folgende Codewort beschrieben:

$$c(\mathbf{u}) = (p(0) \ p(1) \ p(2) \ p(3)) = (1 \ 2 \ 3 \ 4)$$

Abbildung 6.59 zeigt, dass wir in diesem Fall die Geradengleichung tatsächlich rekonstruieren können: Von den vier Möglichkeiten, die Punkte mit einer Geraden zu verbinden, gibt es genau eine, die drei der vier Punkte auf der Geraden vereint. Der konstruierte Code ist jetzt 1-fehlerkorrigierend.



**Abb. 6.59:** Unter den vier Möglichkeiten, die empfangenen Stützpunkte mit einer Geraden zu verbinden, gibt es genau eine, die drei Punkte auf der Geraden vereint.

Zusammenfassend haben unsere Überlegungen gezeigt, dass ein einzelner Prüfwert den Verlust eines Datenpakets kompensieren kann. Ferner reicht ein einzelner Prüfwert aus, um ein verfälschtes Datenpaket zu erkennen. Um ein verfälschtes Datenpaket zu korrigieren, werden dagegen zwei Prüfwerte benötigt. Verallgemeinert führt die Argumentation zu diesem Ergebnis:

### Satz 6.14

Mit  $r$  zusätzlichen Prüfwerten lassen sich

- $r$  Übertragungsverluste kompensieren oder
- $r$  Übertragungsfehler erkennen oder
- $\lfloor \frac{r}{2} \rfloor$  Übertragungsfehler korrigieren.

Auch wenn die Idee bestechend ist, durch die Überapproximation von Polynomen Redundanz zu generieren, hat sie in der bisher präsentierten Form einen Haken. Dieser wird deutlich, wenn wir eine längere Nachricht übertragen wollen, wie z. B. diese hier:

$$\mathbf{u} = (4 \quad 2 \quad 10 \quad 0 \quad 4 \quad 3 \quad 7 \quad 9 \quad 12) \quad (6.40)$$

Interpolieren wir aus diesen Werten ein Polynom  $p(x) \in \mathbb{Q}[x]$  mit  $\deg p(x) < 9$ , so erhalten wir

$$p(x) = \frac{43}{3360}x^8 - \frac{2123}{5040}x^7 + \frac{4109}{720}x^6 - \frac{7379}{180}x^5 + \frac{240703}{1440}x^4 \\ - \frac{274991}{720}x^3 + \frac{139742}{315}x^2 - \frac{20491}{105}x + 4$$

Wollen wir den Empfänger in die Lage versetzen, 3 Übertragungsfehler zu korrigieren, so müssen wir das Polynom an 6 zusätzlichen Stützstellen auswerten. Das Ergebnis ist das folgende Codewort:

$$c(\mathbf{u}) = (4 \ 2 \ 10 \ 0 \ 4 \ 3 \ 7 \ 9 \ 12 \ 844 \ 6992 \ 33202 \ 117108 \ 340669 \ 863709)$$

Jetzt wird klar, welche Nachteile die Rechnung in  $\mathbb{Q}[x]$  mit sich bringt. Die Stützwerte werden so groß, dass die Codierung in dieser Form nicht praxistauglich ist.

### Rechnen in endlichen Körpern

Tatsächlich liegt die Lösung für dieses Problem sehr nahe. Wir brauchen nämlich gar nicht auf Polynome mit rationalen Koeffizienten zurückzugreifen, sondern können genauso gut mit Polynomen aus der Menge  $\mathbb{F}[x]$  rechnen, wobei  $\mathbb{F}$  ein endlicher Körper ist. Für unser Beispiel können wir beispielsweise den Körper  $\mathbb{F}_{2^4}$  verwenden, den wir mittlerweile gut kennen. Dieser umfasst alle Zahlen, die in unserer Beispieldaten (6.40) vorkommen, und enthält genug Elemente, damit das Polynom an 15 unterschiedlichen Stellen ausgewertet werden kann.

Um die Nachricht (6.40) zu codieren, benötigen wir das eindeutig bestimmte Polynom  $p(x) \in \mathbb{F}_{2^4}[x]$  mit  $\deg p(x) < 9$  und

$$\begin{aligned} p(0) &= 4 & p(1) &= 2 & p(2) &= 10 \\ p(3) &= 0 & p(4) &= 4 & p(5) &= 3 \\ p(6) &= 7 & p(7) &= 9 & p(8) &= 12 \end{aligned}$$

Das Polynom, das diese Bedingungen erfüllt, ist

$$p(x) = 15x^8 + 5x^7 + 2x^6 + 3x^5 + 4x^4 + 10x^3 + 5x^2 + 6x + 4$$

Damit sind wir in der Lage, das Codewort auszurechnen. Es ist

$$\begin{aligned} c(\mathbf{u}) &= (p(0) \ \dots \ p(8) \ p(9) \ \dots \ p(14)) \\ &= (4 \ 2 \ 10 \ 0 \ 4 \ 3 \ 7 \ 9 \ 12 \ 5 \ 10 \ 13 \ 2 \ 0 \ 8) \end{aligned} \quad (6.41)$$

Wir sind jetzt gleichsam in der Lage, die Menge der Codewörter zu charakterisieren. Ein Vektor  $(v_0 \dots v_{14})$  aus der Menge  $\mathbb{F}_{2^4}^{15}$  ist genau

dann ein Codewort, wenn ein Polynom  $p(x) \in \mathbb{F}_{2^4}[x]$  mit einem Grad kleiner als 9 existiert mit

$$p(0) = v_0, p(1) = v_1, \dots, p(14) = v_{14}$$

Mit den angestellten Überlegungen bereitet die formale Definition des Reed-Solomon-Codes jetzt keine Schwierigkeiten mehr:



### Definition 6.12 (Reed-Solomon-Code, allgemeine Form)

Es seien  $\mathbb{F}$  ein endlicher Körper und  $\alpha_0, \dots, \alpha_{n-1}$  paarweise verschiedene Elemente aus  $\mathbb{F}$ . Wir setzen:

$$\begin{aligned} \text{RS}_{\mathbb{F}, (\alpha_0, \dots, \alpha_{n-1})}(n, k) := & \{(p(\alpha_0) \ p(\alpha_1) \ \dots \ p(\alpha_{n-1})) \in \mathbb{F}^n \mid \\ & p(x) \in \mathbb{F}[x] \text{ mit } \deg p(x) < k\} \end{aligned}$$

In unserer obigen Beispielrechnung haben wir jeweils 9 Datenpakete aus der Menge  $\mathbb{F}_{2^4}$  auf 15 Datenpakete erweitert. Formal entspricht dies dem Code  $\text{RS}_{\mathbb{F}_{2^4}, (\alpha_0, \dots, \alpha_{14})}(15, 9)$  mit den folgenden Stützstellen:

$$\alpha_0, \dots, \alpha_{14} = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14$$

Welche Stützstellen ausgewählt und in welcher Reihenfolge sie verwendet werden, spielt für die Codierung im Grunde genommen keine Rolle; es ist ausschließlich von Bedeutung, dass die Stützstellen paarweise verschieden sind. Die Codierung funktioniert deshalb genauso gut, wenn sich der Sender und der Empfänger auf die folgenden Stützstellen einigen:

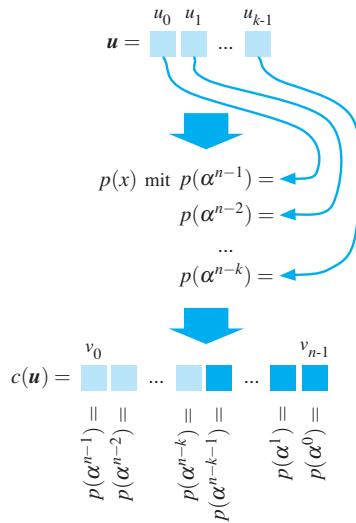
$$\alpha_0, \dots, \alpha_{14} = 9, 13, 15, 14, 7, 10, 5, 11, 12, 6, 3, 8, 4, 2, 1 \quad (6.42)$$

Gegenüber den alten Stützstellen haben die neuen den Vorteil, dass wir sie systematisch mithilfe eines primitiven Elements  $\alpha$  erzeugen können. Wir erinnern uns: Ein solches Element hat die Eigenschaft, dass die Potenzen  $\alpha^0, \alpha^1, \dots$  nacheinander alle von 0 verschiedenen Körperelemente durchlaufen. Ein primitives Element, das die Elemente in der Reihenfolge erzeugt, wie sie in (6.42) angegeben ist, kennen wir aus Abschnitt 6.4.4. Dort wurde gezeigt, dass das Element  $\alpha = 2$  die Körperelemente in genau dieser Reihenfolge erzeugt, wenn wir die Potenzen  $\alpha^0$  bis  $\alpha^{14}$  in umgekehrter Reihenfolge notieren. Es ist

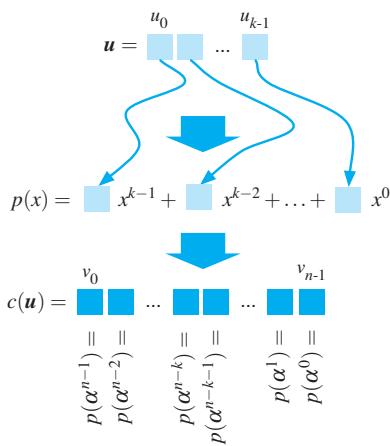
$$2^{14}, \dots, 2^0 = 9, 13, 15, 14, 7, 10, 5, 11, 12, 6, 3, 8, 4, 2, 1$$

Beschränken wir uns darauf, die Stützstellen mithilfe eines primitiven Elements zu erzeugen, dann können wir die Definition von Reed-Solomon-Codes in einer vereinfachten Form angeben:

## Interpolationscodierung



## Koeffizientencodierung



**Abb. 6.60:** Die Interpolationscodierung und die Koeffizientencodierung im Vergleich. Die Menge der Codewörter ist in beiden Fällen gleich, lediglich die konkrete Zuordnung zwischen den Nachrichten und den Codewörtern ist eine andere.



## Definition 6.13 (Reed-Solomon-Code)

Es seien  $\mathbb{F}$  ein endlicher Körper mit mindestens  $n+1$  Elementen und  $\alpha \in \mathbb{F}$  ein primitives Element. Wir setzen:

$$\begin{aligned} \text{RS}_{\mathbb{F}, \alpha}(n, k) := & \{ (p(\alpha^{n-1}), p(\alpha^{n-2}), \dots, p(\alpha^0)) \in \mathbb{F}^n \mid \\ & p(x) \in \mathbb{F}[x] \text{ mit } \deg p(x) < k \} \end{aligned}$$

Wir wollen die Codierung unserer Beispielnachricht wiederholen und dieses Mal den Code  $\text{RS}_{\mathbb{F}_{2^4}, 2}(15, 9)$  verwenden. Um das Codewort zu berechnen, müssen wir ein Polynom interpolieren, das die folgenden Bedingungen erfüllt:

$$\begin{array}{lll} p(9) = 4 & p(13) = 2 & p(15) = 10 \\ p(14) = 0 & p(7) = 4 & p(10) = 3 \\ p(5) = 7 & p(11) = 9 & p(12) = 12 \end{array}$$

Fündig werden wir bei diesem Polynom:

$$p(x) = 2x^8 + 13x^7 + 4x^6 + 10x^5 + 0x^4 + 8x^3 + 4x^2 + 5x + 11$$

Werten wir  $p(x)$  an den vereinbarten Stützstellen aus, so erhalten wir das Codewort. Es ist

$$\begin{aligned} c(\mathbf{u}) &= (p(2^{14}), \dots, p(2^6), \textcolor{blue}{p(2^5)}, \dots, \textcolor{blue}{p(2^0)}) \\ &= (4 \ 2 \ 10 \ 0 \ 4 \ 3 \ 7 \ 9 \ 12 \ \textcolor{blue}{0} \ 3 \ \textcolor{blue}{7} \ 11 \ 14 \ 3) \end{aligned}$$

## Koeffizientencodierung

In den Definitionen 6.12 und 6.13 haben wir festgelegt, wie die Codewörter des Reed-Solomon-Codes aussehen, aber nicht geregelt, auf welches Codewort eine vorgelegte Nachricht konkret abgebildet wird. In unserer Beispielrechnung haben wir eine mögliche Art der Codierung bereits vorweggenommen, allerdings ist diese nicht die einzige mögliche. Anstatt die Nachrichtenpakete als die Funktionswerte eines Polynoms zu interpretieren, können wir diese nämlich genauso gut als die Koeffizienten eines Polynoms betrachten (Abbildung 6.60).

Um eine Nachricht

$$\mathbf{u} = (u_0 \ \dots \ u_{k-1})$$

auf diese Weise zu codieren, wird sie ganz einfach als ein Polynom

$$u(x) = u_0x^{k-1} + u_1x^{k-2} + \dots + u_{k-3}x^2 + u_{k-2}x + u_{k-1}$$

aufgefasst und an  $n$  Stützstellen ausgewertet. Das gesendete Codewort ist dann

$$c(\mathbf{u}) = (p(\alpha_0) \quad p(\alpha_1) \quad p(\alpha_2) \quad \dots \quad p(\alpha_{n-1}))$$

wobei  $\alpha_0$  bis  $\alpha_{n-1}$  paarweise verschiedene Körperelemente sind. Auch hier können wir die Stützstellen über ein primitives Element  $\alpha$  erzeugen und das Codewort dann beispielsweise so wählen:

$$c(\mathbf{u}) = (p(\alpha^{n-1}) \quad p(\alpha^{n-2}) \quad p(\alpha^{n-3}) \quad \dots \quad p(\alpha^0))$$

Unsere Beispielnachricht

$$\mathbf{u} = (4 \quad 2 \quad 10 \quad 0 \quad 4 \quad 3 \quad 7 \quad 9 \quad 12)$$

entspricht dann dem Polynom

$$u(x) = 4x^8 + 2x^7 + 10x^6 + 4x^4 + 3x^3 + 7x^2 + 9x + 12,$$

und das Codewort  $c(\mathbf{u})$  lautet, wenn wir die das primitive Element  $\alpha = 2$  wählen, wie folgt:

$$c(\mathbf{u}) = (3 \quad 1 \quad 14 \quad 14 \quad 4 \quad 1 \quad 11 \quad 11 \quad 14 \quad 8 \quad 4 \quad 4 \quad 2 \quad 6 \quad 9)$$

Damit haben wir zwei prinzipielle Möglichkeiten kennengelernt, Nachrichten auf Reed-Solomon-Codewörter abzubilden. Es ist wichtig zu erkennen, dass die Menge der Codewörter beide Male dieselbe ist, lediglich die Zuordnung, welche Nachricht auf welches Codewort abgebildet wird, ist eine andere.

Durch die geänderte Zuordnungsvorschrift verliert die Codierung die Eigenschaft, systematisch zu sein, dafür erspart das Vorgehen die zeitaufwendige Interpolation des Polynoms. In Abschnitt 6.4.5.3 werden wir zeigen, dass der vermeintliche Geschwindigkeitsvorteil aber nur auf den ersten Blick existiert. Als Nebenprodukt der dort angestellten Überlegungen werden wir zu der Erkenntnis gelangen, dass sich die Interpolationscodierung genauso schnell ausführen lässt wie die Koeffizientencodierung.

### 6.4.5.2 Decodierung

Nachdem wir uns ausführlich mit den verschiedenen Möglichkeiten befasst haben, eine Nachricht auf ein Reed-Solomon-Codewort abzubilden, wollen wir uns der Frage zuwenden, wie sich codierte Nachrichten auf der Empfängerseite überprüfen und gegebenenfalls korrigieren lassen. Die am nächsten liegende Methode ist sicherlich diese hier: Haben



In diesem Abschnitt haben wir gesehen, dass es mehrere naheliegende Zuordnungen zwischen Nachrichten auf der einen Seite und den Reed-Solomon-Codewörtern auf der anderen Seite gibt. An dieser Stelle ist es angebracht, einen erneuten Blick auf das Eingangszitat auf Seite 414 zu werfen. Aus ihm geht hervor, dass Reed und Solomon in ihrer Publikation aus dem Jahr 1960 eine Nachricht nach dem Verfahren codieren, das wir hier als Koeffizientencodierung bezeichnet haben. Das Eingangszitat gibt auch Auskunft über die restlichen Codeparameter. Reed und Solomon definierten ihren Code über einem endlichen Körper  $K$  mit  $2^n$  Elementen und verwendeten als Stützstellen die Elemente  $0, \alpha, \alpha^2, \dots, \alpha^{2^n-1}$ . In unserer Notation ist der historische Reed-Solomon-Code der Code

$$\text{RS}_{\mathbb{F}_{2^n}, 0, \alpha, \alpha^2, \dots, \alpha^{2^n-1}}(2^n, m)$$

Beachten Sie, dass die historische Definition des Reed-Solomon-Codes nicht vollständig kompatibel zu jenem Code ist, den wir in Definition 6.13 als den Reed-Solomon-Code  $\text{RS}_{\mathbb{F}, \alpha}(n, k)$  bezeichnet haben. Zwar werden die Stützstellen beide Male durch ein primitives Element  $\alpha$  erzeugt, allerdings gibt es in zwei Punkten Unterschiede. Zum einen ist die Reihenfolge, in der die Stützstellen ausgewertet werden, bei uns eine andere. Zum anderen verwenden Reed und Solomon die Stützstelle  $0$ , die bei uns ausgespart wird. Dass wir von der historischen Definition abweichen, hat einen Grund, der später klar werden wird. In der Form, wie die Reed-Solomon-Codes hier definiert sind, weisen sie eine enge Verwandtschaft mit den BCH-Codes aus Abschnitt 6.4.4 auf. Dieser Zusammenhang gehört zu den faszinierendsten Ergebnissen der Codierungstheorie; Reed und Solomon war er im Jahr 1960 aber noch nicht bekannt.



Elwyn Ralph Berlekamp [70]  
(geb. 6. 9. 1940)

**Abb. 6.61:** In der Codierungstheorie treffen wir an vielen Stellen auf den Namen Elwyn Berlekamp. Neben dem Berlekamp-Welch-Algorithmus ist der amerikanische Mathematiker vor allem durch den *Berlekamp-Massey-Algorithmus* bekannt. Entstanden ist dieser aus einem Algorithmus, der von Berlekamp Ende der 60er-Jahre für die Decodierung von BCH-Codes eronnen und von James Lee Massey weiterentwickelt wurde. Neben seinen Arbeiten zur algebraischen Codierungstheorie hat Berlekamp auch im Bereich der Spieltheorie wichtige Beiträge verfasst. Seit 1971 ist er Professor an der University of California, Berkeley, wo er noch heute lehrt und forscht.

wir, wie in unserem Beispiel, 9 Datenpakete um 6 Prüfpakete erweitert, so wissen wir, dass 9 korrekte Stützwerte das verwendete Polynom eindeutig bestimmen. Da wir nicht wissen können, welche Datenpakete korrekt und welche falsch sind, könnten wir der Idee verfallen, für jeweils 9 Datenpakete ein Polynom zu interpolieren und zu testen, wie viele der empfangenen Datenpakete Stützwerte dieses Polynoms sind. Probiert der Empfänger sämtliche Möglichkeiten durch und wählt am Ende dasjenige Polynom mit den meisten Treffern aus, so erhalten wir das, was wir auf Seite 362 als *Maximum-Likelihood-Decoder* bezeichnet haben.

Auch wenn dieses Verfahren zweifellos funktioniert, braucht es kaum eine tiefergehende Analyse, um seine praktische Untauglichkeit festzustellen. Die Anzahl an Kombinationen, die wir berücksichtigen müssen, ist bei längeren Codewörtern so groß, dass wir die Terminierung dieses Algorithmus in den meisten Fällen nicht mehr erleben würden.

Glücklicherweise reichen ein paar mathematische Kunstgriffe aus, um Reed-Solomon-codierte Nachrichten effizient zu decodieren. Eine besonders clevere Methode wurde im Jahr 1986 zum Patent angemeldet. Sie stammt von den amerikanischen Mathematikern Elwyn Ralph Berlekamp (Abbildung 6.61) und Lloyd Richard Welch.

### Berlekamp-Welch-Algorithmus

Um die Darstellung übersichtlich zu halten, werden wir den Berlekamp-Welch-Algorithmus an einem konkreten Beispiel demonstrieren. Alle der nachfolgend angestellten Überlegungen lassen sich aber problemlos verallgemeinern.

Wir verwenden den Code  $\text{RS}_{\mathbb{Z}_7,3}(6,2)$ , für den Folgendes gilt:

- Der endliche Körper umfasst 7 Elemente.  $\text{RS}_{\mathbb{Z}_7,3}(6,2)$
- Das primitive Element  $\alpha$  ist die Zahl 3.  $\text{RS}_{\mathbb{Z}_7,3}(6,2)$
- Nachrichten der Länge 2 werden ...  $\text{RS}_{\mathbb{Z}_7,3}(6,2)$
- ... auf Codewörter der Länge 6 abgebildet.  $\text{RS}_{\mathbb{Z}_7,3}(6,2)$

Als Beispiel betrachten wir die folgende Nachricht:

$$\mathbf{u} = (2 \quad 3)$$

Verwendet der Sender das Prinzip der Koeffizientencodierung, so formt er daraus das Polynom

$$p(x) = 2x + 3$$

und überträgt das Codewort

$$\begin{aligned} \mathbf{v} = c(\mathbf{u}) &= (p(\alpha^5) \quad p(\alpha^4) \quad p(\alpha^3) \quad p(\alpha^2) \quad p(\alpha^1) \quad p(\alpha^0)) \\ &= (p(5) \quad p(4) \quad p(6) \quad p(2) \quad p(3) \quad p(1)) \\ &= (6 \quad 4 \quad 1 \quad 0 \quad 2 \quad 5) \end{aligned}$$

Wir nehmen an, das erste und das vorletzte Datenpaket wurden auf der Übertragungsstrecke so verfälscht, dass der folgende Vektor den Empfänger erreicht:

$$\mathbf{w} = (5 \quad 4 \quad 1 \quad 0 \quad 1 \quad 5)$$

Die empfangenen Werte können wir mit einem Polynom  $r(x)$  identifizieren, das die Beziehung

$$\mathbf{w} = (r(\alpha^5) \quad r(\alpha^4) \quad r(\alpha^3) \quad r(\alpha^2) \quad r(\alpha^1) \quad r(\alpha^0))$$

erfüllt und mit  $p(x)$  folgendermaßen in Beziehung steht:

$$\begin{aligned} p(\alpha^5) &\neq r(\alpha^5) \quad \text{👉 Übertragungsfehler} \\ p(\alpha^4) &= r(\alpha^4) \\ p(\alpha^3) &= r(\alpha^3) \\ p(\alpha^2) &= r(\alpha^2) \\ p(\alpha^1) &\neq r(\alpha^1) \quad \text{👉 Übertragungsfehler} \\ p(\alpha^0) &= r(\alpha^0) \end{aligned}$$

Im nächsten Schritt kommt der Hauptakteur des Berlekamp-Welch-Algorithmus ins Spiel: das *Fehlerpolynom*  $e(x)$ . Per Definition erfüllt  $e(x)$  die folgende Eigenschaft:

$e(\alpha^i) \neq 0$  falls der Stützwert  $p(\alpha^i)$  richtig übertragen wurde

$e(\alpha^i) = 0$  falls der Stützwert  $p(\alpha^i)$  falsch übertragen wurde

In unserem Fehlerszenario werden die Werte an den Positionen  $\alpha^5$  und  $\alpha^1$  falsch übertragen, was durch das folgende Polynom beschrieben wird:

$$e(x) = (x - \alpha^5)(x - \alpha^1) = (x - 5)(x - 3)$$

Auch wenn der Empfänger das Polynom  $e(x)$  noch nicht kennen kann, weiß er sehr wohl über dessen allgemeine Form Bescheid. Unserem Beispiel liegt eine *2-Fehlerannahme* zugrunde, d. h., wir gehen davon

aus, dass höchstens 2 Datenpakete während der Übertragung verfälscht wurden. Damit weiß der Empfänger, dass sich  $e(x)$  in der Form

$$e(x) = (x - \alpha^i)(x - \alpha^j)$$

aufschreiben lässt, die sich in normalisierter Darstellung folgendermaßen liest:

$$e(x) = x^2 + e_1 x + e_0$$

In unserem Beispiel gilt

$$e(x) = x^2 + 6x + 1,$$

was dem Empfänger natürlich noch nicht bekannt ist.

Die Definition von  $e(x)$  ist trickreicher, als es der erste Blick vermuten lässt. Sie verleiht dem Fehlerpolynom die Eigenschaft, die Polynome  $p(x)$  und  $r(x)$  an den Fehlerstellen anzugeleichen. Es gilt:

$$p(\alpha^5)e(\alpha^5) = r(\alpha^5)e(\alpha^5) \quad \text{wegen } e(\alpha^5) = 0 \quad (6.43)$$

$$p(\alpha^4)e(\alpha^4) = r(\alpha^4)e(\alpha^4) \quad (6.44)$$

$$p(\alpha^3)e(\alpha^3) = r(\alpha^3)e(\alpha^3) \quad (6.45)$$

$$p(\alpha^2)e(\alpha^2) = r(\alpha^2)e(\alpha^2) \quad (6.46)$$

$$p(\alpha^1)e(\alpha^1) = r(\alpha^1)e(\alpha^1) \quad \text{wegen } e(\alpha^1) = 0 \quad (6.47)$$

$$p(\alpha^0)e(\alpha^0) = r(\alpha^0)e(\alpha^0) \quad (6.48)$$

Was weiß der Empfänger über das Produktpolynom  $p(x)e(x)$ , das auf den linken Seiten dieser Gleichungen auftritt? Der Empfänger kennt zwar weder  $p(x)$  noch  $e(x)$  im Detail, allerdings weiß er über den Grad der Polynome Bescheid. In unserem Beispiel ist  $\deg e(x) = 2$  und  $\deg p(x) = 1$ , sodass der Grad des Polynoms  $p(x)e(x)$  gleich 3 sein muss. In normalisierter Form hat  $p(x)e(x)$  demnach die folgende Gestalt:

$$p(x)e(x) = a_3 x^3 + a_2 x^2 + a_1 x + a_0$$

Für unser Beispiel gilt

$$p(x)e(x) = (2x + 3)(x - 5)(x - 3) = 2x^3 + x^2 + 6x + 3,$$

aber auch das kann der Empfänger natürlich noch nicht wissen.

Schreiben wir die Gleichungen (6.43) bis (6.48) aus, so erhalten wir

$$a_3 x^3 + a_2 x^2 + a_1 x + a_0 = r(x)(x^2 + e_1 x + e_0) \quad (\text{für } x \in \{\alpha^5, \dots, \alpha^0\})$$

und dies können wir weiter umformen zu

$$a_3x^3 + a_2x^2 + a_1x + a_0 - r(x)e_1x - r(x)e_0 = r(x)x^2 \quad (\text{für } x \in \{\alpha^5, \dots, \alpha^0\})$$

Wegen  $\alpha = 3$  ist dies ausgeschrieben das Gleiche wie:

$$\begin{aligned} a_3 \cdot 6 + a_2 \cdot 4 + a_1 \cdot 5 + a_0 - r(5) \cdot e_1 \cdot 5 - r(5) \cdot e_0 &= r(5) \cdot 4 \\ a_3 \cdot 1 + a_2 \cdot 2 + a_1 \cdot 4 + a_0 - r(4) \cdot e_1 \cdot 4 - r(4) \cdot e_0 &= r(4) \cdot 2 \\ a_3 \cdot 6 + a_2 \cdot 1 + a_1 \cdot 6 + a_0 - r(6) \cdot e_1 \cdot 6 - r(6) \cdot e_0 &= r(6) \cdot 1 \\ a_3 \cdot 1 + a_2 \cdot 4 + a_1 \cdot 2 + a_0 - r(2) \cdot e_1 \cdot 2 - r(2) \cdot e_0 &= r(2) \cdot 4 \\ a_3 \cdot 6 + a_2 \cdot 2 + a_1 \cdot 3 + a_0 - r(3) \cdot e_1 \cdot 3 - r(3) \cdot e_0 &= r(3) \cdot 2 \\ a_3 \cdot 1 + a_2 \cdot 1 + a_1 \cdot 1 + a_0 - r(1) \cdot e_1 \cdot 1 - r(1) \cdot e_0 &= r(1) \cdot 1 \end{aligned}$$

Hierin sind  $r(5), r(4), r(6), r(2), r(3)$  und  $r(1)$  die empfangenen Datenpakete. Setzen wir hierfür die konkreten Werte ein, so erhalten wir das folgende Ergebnis:

$$\begin{aligned} a_3 \cdot 6 + a_2 \cdot 4 + a_1 \cdot 5 + a_0 - 5 \cdot e_1 \cdot 5 - 5 \cdot e_0 &= 5 \cdot 4 \\ a_3 \cdot 1 + a_2 \cdot 2 + a_1 \cdot 4 + a_0 - 4 \cdot e_1 \cdot 4 - 4 \cdot e_0 &= 4 \cdot 2 \\ a_3 \cdot 6 + a_2 \cdot 1 + a_1 \cdot 6 + a_0 - 1 \cdot e_1 \cdot 6 - 1 \cdot e_0 &= 1 \cdot 1 \\ a_3 \cdot 1 + a_2 \cdot 4 + a_1 \cdot 2 + a_0 - 0 \cdot e_1 \cdot 2 - 0 \cdot e_0 &= 0 \cdot 4 \\ a_3 \cdot 6 + a_2 \cdot 2 + a_1 \cdot 3 + a_0 - 1 \cdot e_1 \cdot 3 - 1 \cdot e_0 &= 1 \cdot 2 \\ a_3 \cdot 1 + a_2 \cdot 1 + a_1 \cdot 1 + a_0 - 5 \cdot e_1 \cdot 1 - 5 \cdot e_0 &= 5 \cdot 1 \end{aligned}$$

Was wir hier für uns haben, ist ein lineares Gleichungssystem mit 6 Gleichungen und 6 Unbekannten:

$$\begin{aligned} 6a_3 + 4a_2 + 5a_1 + 1a_0 + 3e_1 + 2e_0 &= 6 \\ 1a_3 + 2a_2 + 4a_1 + 1a_0 + 5e_1 + 3e_0 &= 1 \\ 6a_3 + 1a_2 + 6a_1 + 1a_0 + 1e_1 + 6e_0 &= 1 \\ 1a_3 + 4a_2 + 2a_1 + 1a_0 + 0e_1 + 0e_0 &= 0 \\ 6a_3 + 2a_2 + 3a_1 + 1a_0 + 4e_1 + 6e_0 &= 2 \\ 1a_3 + 1a_2 + 1a_1 + 1a_0 + 2e_1 + 2e_0 &= 5 \end{aligned}$$

Abbildung 6.62 (oben) zeigt, wie sich dieses Gleichungssystems lösen lässt. Für die Koeffizienten des Polynoms  $g(x)e(x)$  erhalten wir  $a_3 = 2$ ,  $a_2 = 1$ ,  $a_1 = 6$ ,  $a_0 = 3$ , und für die Koeffizienten des Fehlerpolynoms  $e(x)$  erhalten wir  $e_1 = 6$  und  $e_0 = 1$ .

Ab jetzt weiß der Empfänger das Folgende:

$$\begin{aligned} p(x)e(x) &= 2x^3 + x^2 + 6x + 3 \\ e(x) &= x^2 + 6x + 1 \end{aligned}$$

$$\left( \begin{array}{cccccc} 6 & 4 & 5 & 1 & 3 & 2 \\ 1 & 2 & 4 & 1 & 5 & 3 \\ 6 & 1 & 6 & 1 & 1 & 6 \\ 1 & 4 & 2 & 1 & 0 & 0 \\ 6 & 2 & 3 & 1 & 4 & 6 \\ 1 & 1 & 1 & 1 & 2 & 2 \end{array} \right) \left( \begin{array}{c} a_3 \\ a_2 \\ a_1 \\ a_0 \\ e_1 \\ e_0 \end{array} \right) = \left( \begin{array}{c} 6 \\ 1 \\ 1 \\ 0 \\ 2 \\ 5 \end{array} \right)$$

$$\left( \begin{array}{cccccc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array} \right) \left( \begin{array}{c} a_3 \\ a_2 \\ a_1 \\ a_0 \\ e_1 \\ e_0 \end{array} \right) = \left( \begin{array}{c} 2 \\ 1 \\ 6 \\ 3 \\ 6 \\ 1 \end{array} \right)$$

$$\begin{aligned} p(x) \cdot e(x) &= 2x^3 + x^2 + 6x + 3 \\ e(x) &= x^2 + 6x + 1 \end{aligned}$$

$$\begin{array}{r} 2x^3 + x^2 + 6x + 3 = (x^2 + 6x + 1) \cdot (2x + 3) \\ -2x^3 - 5x^2 - 2x \\ \hline 3x^2 + 4x + 3 \\ -3x^2 - 4x - 3 \\ \hline 0 \end{array}$$

$$p(x) = 2x + 3$$

**Abb. 6.62:** Rekonstruktion der Originalnachricht mithilfe des Berlekamp-Welch-Algorithmus

Jetzt ist die Lösung ganz nahe. Um das Polynom  $p(x)$ , mit dem der Sender die Nachricht codiert hat, zu rekonstruieren, muss der Empfänger nur noch  $p(x)e(x)$  durch  $e(x)$  dividieren. Die Rechnung in Abbildung 6.62 (unten) liefert das Ergebnis

$$p(x) = 2x + 3,$$

und die Koeffizienten dieses Polynoms sind die ursprünglich gesendete Nachricht

$$\mathbf{u} = (2 \quad 3)$$

Damit haben wir es geschafft, die Nachricht erfolgreich zu rekonstruieren.

#### 6.4.5.3 Reed-Solomon-Codes unter der Lupe

Nachdem wir die grundlegenden Eigenschaften von Reed-Solomon-Codes kennengelernt haben, wollen wir ein wenig tiefer schürfen und eine Reihe von Querbezügen zu den Codeklassen herstellen, die wir bereits kennen. Am Ende dieses Abschnitts wird die Erkenntnis stehen, dass Reed-Solomon-Codes, so andersartig sie auf den ersten Blick wirken mögen, alles andere als isolierte Gebilde im Codierungsuniversum sind.

Zunächst werden wir zeigen, dass die Reed-Solomon-Codes zu der großen Gruppe der linearen Codes gehören, und als Nebenprodukt eine Berechnungsmethode erarbeiten, mit der die Interpolationscodierung sehr schnell ausgeführt werden kann. Im Anschluss daran werden wir darlegen, dass Reed-Solomon-Codes eng mit einer Klasse von Codes verwandt sind, mit der wir schon gut vertraut sind. Die Rede ist von den BCH-Codes, die wir ausgiebig in Abschnitt 6.4.4 besprochen haben. Zum Schluss werden wir herausarbeiten, dass Reed-Solomon-Codes zyklisch sind, wenn für die Länge der Codewörter ganz bestimmte Werte gewählt werden. Damit ist genug verraten, legen wir los!

Um zu zeigen, dass Reed-Solomon-Codes linear sind, müssen wir uns lediglich an die Vorschrift erinnern, die wir weiter oben als Koeffizientencodierung bezeichnet haben. Nach dieser Vorschrift wird eine Nachricht

$$\mathbf{u} = (u_0, \dots, u_{k-1})$$

codiert, indem die Symbole  $u_0, \dots, u_{k-1} \in \mathbb{F}$  als die Koeffizienten des Polynoms

$$u(x) = u_0x^{k-1} + \dots + u_{k-3}x^2 + u_{k-2}x + u_{k-1}$$

aufgefasst werden, und  $u(x)$  anschließend an  $n$  paarweise verschiedenen Stützstellen  $\alpha_0, \dots, \alpha_{n-1}$  ausgewertet wird. Als Ergebnis erhalten wir das Codewort

$$\mathbf{v} = (v_0 \dots v_{n-1}) = (p(\alpha_0) \quad p(\alpha_1) \quad p(\alpha_2) \quad \dots \quad p(\alpha_{n-1}))$$

Genauso gut können wir  $\mathbf{v}$  über eine Matrixmultiplikation berechnen.

Es ist

$$\mathbf{v} = (v_0 \dots v_{n-1}) = (u_0 \dots u_{k-1}) \cdot G$$

mit

$$G = \begin{pmatrix} (\alpha_0)^{k-1} & \dots & (\alpha_{n-3})^{k-1} & (\alpha_{n-2})^{k-1} & (\alpha_{n-1})^{k-1} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ (\alpha_0)^2 & \dots & (\alpha_{n-3})^2 & (\alpha_{n-2})^2 & (\alpha_{n-1})^2 \\ (\alpha_0) & \dots & (\alpha_{n-3}) & (\alpha_{n-2}) & (\alpha_{n-1}) \\ 1 & \dots & 1 & 1 & 1 \end{pmatrix}$$

Das bedeutet, dass  $G$  die Generatormatrix von  $\text{RS}(n, k)$  ist, und die Tatsache, dass sich Reed-Solomon-Codes mithilfe einer Generatormatrix erzeugen lassen, beweist deren Linearität:

### Satz 6.15

Reed-Solomon-Codes sind linear.

Erzeugen wir die Stützstellen mithilfe eines primitiven Elements, so wie es in Definition 6.13 gefordert ist, dann nimmt die Matrix die folgende Gestalt an:

$$G = \begin{pmatrix} (\alpha^{n-1})^{k-1} & \dots & (\alpha^2)^{k-1} & (\alpha)^{k-1} & 1 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ (\alpha^{n-1})^2 & \dots & (\alpha^2)^2 & (\alpha)^2 & 1 \\ (\alpha^{n-1}) & \dots & (\alpha^2) & (\alpha) & 1 \\ 1 & \dots & 1 & 1 & 1 \end{pmatrix} \quad (6.49)$$

Abbildung 6.63 zeigt, wie die Generatormatrix für unsere Beispielcodierung aussieht, mit der wir auf Seite 420 die Funktionsweise der Reed-Solomon-Codierung verdeutlicht haben. Diese Codierung bildet jeweils 9 Elemente aus  $\mathbb{F}_{2^4}$  auf Codewörter der Länge 15 ab, es ist also  $k = 9$  und  $n = 15$ . Als primitives Element wird  $\alpha = 2$  verwendet, und die Stützstellen sind  $\alpha^{14}, \dots, \alpha^0$ . Die Generatormatrix lässt sich durch elementare Zeilenumformungen in eine systematische Form bringen, die ebenfalls in Abbildung 6.63 zu sehen ist.

Generatormatrix in der Form (6.49)	Generatormatrix in systematischer Form
$\begin{pmatrix} 11 & 9 & 12 & 13 & 6 & 15 & 3 & 14 & 8 & 7 & 4 & 10 & 2 & 5 & 1 \\ 5 & 2 & 10 & 4 & 7 & 8 & 14 & 3 & 15 & 6 & 13 & 12 & 9 & 11 & 1 \\ 10 & 8 & 15 & 12 & 1 & 10 & 8 & 15 & 12 & 1 & 10 & 8 & 15 & 12 & 1 \\ 7 & 6 & 1 & 7 & 6 & 1 & 7 & 6 & 1 & 7 & 6 & 1 & 7 & 6 & 1 \\ 14 & 11 & 8 & 9 & 7 & 12 & 4 & 13 & 10 & 6 & 2 & 15 & 5 & 3 & 1 \\ 15 & 10 & 12 & 8 & 1 & 15 & 10 & 12 & 8 & 1 & 15 & 10 & 12 & 8 & 1 \\ 13 & 14 & 10 & 11 & 6 & 8 & 2 & 9 & 15 & 7 & 5 & 12 & 3 & 4 & 1 \\ 9 & 13 & 15 & 14 & 7 & 10 & 5 & 11 & 12 & 6 & 3 & 8 & 4 & 2 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 & 3 & 5 & 13 & 1 & 8 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 15 & 1 & 13 & 7 & 5 & 13 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 11 & 11 & 13 & 3 & 10 & 7 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 2 & 3 & 8 & 4 & 7 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 3 & 10 & 10 & 6 & 15 & 9 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 5 & 11 & 1 & 5 & 15 & 11 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 2 & 11 & 10 & 7 & 14 & 8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 15 & 9 & 5 & 8 & 15 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 7 & 9 & 3 & 12 & 10 & 12 \end{pmatrix}$

Abb. 6.63: Generatormatrix von RS(15,9), links in der Gestalt von (6.49) und rechts in systematischer Form

Tatsächlich lösen wir mit dieser Matrix unser Versprechen ein, das wir weiter oben gegeben haben. Dort haben wir angedeutet, dass sich die Interpolationscodierung genauso effizient berechnen lässt wie die Koeffizientencodierung. Die soeben hergeleitete Matrix zeigt, dass dies wirklich der Fall ist. Wir können das Codewort ganz einfach durch die Multiplikation mit der Generatormatrix in systematischer Form erzeugen und hierdurch auf die zeitintensive Polynominterpolation verzichten.

Ab jetzt betrachten wir nur noch Reed-Solomon-Codes  $\text{RS}_{\mathbb{F},\alpha}(n,k)$  über einem endlichen Körper  $\mathbb{F}$ , der genau  $n+1$  Elemente enthält; die Anzahl der Körperelemente ist also genau um eins größer als die Codewortlänge. Für einen solchen Code stellen wir jetzt die folgende Matrix auf:

$$H = \begin{pmatrix} (\alpha^{n-1})^{n-k} & \dots & (\alpha^2)^{n-k} & (\alpha)^{n-k} & 1 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ (\alpha^{n-1})^2 & \dots & (\alpha^2)^2 & (\alpha)^2 & 1 \\ (\alpha^{n-1}) & \dots & (\alpha^2) & (\alpha) & 1 \end{pmatrix}$$

Nehmen wir jeweils eine Zeile aus  $G$  und eine Zeile aus  $H$  heraus und bilden das Produkt, so erhalten wir

$$\sum_{i=0}^{n-1} (\alpha^i)^p (\alpha^i)^q \quad (6.50)$$

mit  $0 \leq p \leq k-1$  und  $1 \leq q \leq n-k$ . Die Summe (6.50) ist das Gleiche wie

$$\sum_{i=0}^{n-1} (\alpha^i)^{p+q} \quad (6.51)$$

Da der Körper  $\mathbb{F}$  genau  $n+1$  Elemente enthält, durchläuft  $\alpha^i$  nacheinander alle von 0 verschiedenen Elemente. Das heißt, dass wir (6.51) auch so aufschreiben können:

$$\sum_{\substack{\alpha \in \mathbb{F} \\ \alpha \neq 0}} \alpha^{p+q} \quad (6.52)$$

Für die Summe  $p+q$  gilt:

$$1 \leq p+q \leq n-1$$

Damit folgt aus Satz 2.9, den wir auf Seite 122 bewiesen haben, dass die Summe (6.52) gleich 0 ist. Alles in allem gilt damit für (6.50) die folgende Beziehung:

$$\sum_{i=0}^{n-1} (\alpha^i)^p (\alpha^i)^q = 0$$

Das heißt, dass die Zeilenvektoren aus  $G$  und die Zeilenvektoren aus  $H$  orthogonal aufeinander stehen. Da die Zeilenvektoren in  $G$  und  $H$  linear unabhängig sind, spannen sie Vektorräume der Dimension  $k$  bzw.  $n-k$  auf. Das wiederum bedeutet, dass  $H$  die Generatormatrix des Orthogonalraums von  $\text{RS}(n,k)$  und damit gleichzeitig die Kontrollmatrix von  $\text{RS}(n,k)$  ist.

Die Kenntnis der Kontrollmatrix eröffnet uns eine neue Möglichkeit, die Reed-Solomon-Codewörter zu charakterisieren. Ist  $(v_0 \dots v_{n-1})$  ein Codewort, so wissen wir, dass die Multiplikation mit der Kontrollmatrix den Nullvektor ergeben muss:

$$Hv^T = \mathbf{0}$$

Dies ist ausgeschrieben das Gleiche wie

$$\begin{aligned} v_0(\alpha^{n-1})^{n-k} + \dots + v_{n-3}(\alpha^2)^{n-k} + v_{n-2}(\alpha)^{n-k} + v_{n-1} &= 0 \\ \dots \\ v_0(\alpha^{n-1})^2 + \dots + v_3(\alpha^2)^2 + v_2(\alpha)^2 + v_{n-1} &= 0 \\ v_0(\alpha^{n-1}) + \dots + v_3(\alpha^2) + v_2(\alpha) + v_{n-1} &= 0 \end{aligned}$$

und dies lässt sich folgendermaßen umformen:

$$\begin{aligned} v_0(\alpha^{n-k})^{n-1} + \dots + v_{n-3}(\alpha^{n-k})^2 + v_{n-2}(\alpha^{n-k}) + v_{n-1} &= 0 \\ \dots \\ v_0(\alpha^2)^{n-1} + \dots + v_3(\alpha^2)^2 + v_2(\alpha^2) + v_{n-1} &= 0 \\ v_0(\alpha)^{n-1} + \dots + v_3(\alpha)^2 + v_2(\alpha) + v_{n-1} &= 0 \end{aligned}$$

Vereinbaren wir, wie gewöhnlich,

$$v(x) = v_0x^{n-1} + \dots + v_{n-3}x^2 + v_{n-2}x + v_{n-1},$$

so können wir die aufgestellten Gleichungen deutlich kürzer notieren.  
Sie sind das Gleiche wie:

$$v(\alpha^i) = 0 \quad (1 \leq i \leq n-k)$$

Damit sind wir in der Lage, eine alternative Definition der Reed-Solomon-Codes anzugeben, die sich auf die Nullstellen des Codewortpolynoms stützt:



#### Definition 6.14 (Reed-Solomon-Code)

Es seien  $\mathbb{F}$  ein endlicher Körper mit  $n+1$  Elementen und  $\alpha \in \mathbb{F}$  ein primitives Element. Wir setzen:

$$\text{RS}_{\mathbb{F}, \alpha}(n, k) := \{(v_0, \dots, v_{n-1}) \in \mathbb{F}^n \mid v(\alpha) = \dots = v(\alpha^{n-k}) = 0\}$$

Hierin steht  $v(x)$  für das Polynom  $v_0x^{n-1} + \dots + v_{n-1} \in \mathbb{F}[x]$ .

Ein Blick auf Seite 409 zeigt, dass diese Definition eine verblüffende Ähnlichkeit mit der Definition der BCH-Codes aufweist. Wir wollen noch ein weniger konkreter werden und den Reed-Solomon-Code RS(15,9) mit dem BCH-Code BCH(15,3) vergleichen. Beide weisen eine Codewortlänge von 15 Bit auf und erfüllen die folgenden Beziehungen:

$$\text{RS}(15,9) : v(\alpha) = \dots = v(\alpha^{n-k}) = 0 \text{ mit } n = 15 \text{ und } k = 9 \quad (6.53)$$

$$\text{BCH}(15,3) : v(\alpha) = \dots = v(\alpha^{2\delta}) = 0 \text{ mit } \delta = 3 \quad (6.54)$$

Wegen  $2\delta = 6$  und  $n - k = 6$  sind (6.54) und (6.53) beide äquivalent zu

$$v(\alpha) = \dots = v(\alpha^6) = 0$$

BCH(15,3) und RS(15,9) unterscheiden sich somit nur in den Symbolen, aus denen die Codewörter bestehen. Während BCH-Codes binäre Codes sind und die Codewörter deshalb nur die Elemente 0 und 1 enthalten, setzen sich die Reed-Solomon-Codewörter aus beliebigen Körperelementen zusammen. Folgerichtig ist die Menge  $\text{BCH}(15,3)$  eine Untermenge von  $\text{RS}(15,9)$ : Sie enthält genau diejenigen Reed-Solomon-Codewörter, in denen ausschließlich die Symbole 0 und 1 vorkommen. Verallgemeinert können wir den Zusammenhang, der zwischen den BCH-Codes auf der einen Seite und den Reed-Solomon-Codes auf der anderen Seite besteht, so aufschreiben:



```
gcc rs_filter.c
./a.out
```

0. 0000000000000000	9. 010011011100001	18. 100100011110101	27. 110111000010100
1. 000010100110111	10. 010100110111000	19. 100110111000010	28. 111000010100110
2. 000101001101110	11. 010110010001111	20. 101001101110000	29. 111010110010001
3. 000111101011001	12. 011001000111101	21. 101011001000111	30. 111101011001000
4. 001000111101011	13. 011011100001010	22. 101100100011110	31. 111111111111111
5. 001010011011100	14. 011100001010011	23. 101110000101001	
6. 001101110000101	15. 011110101100100	24. 110000101001101	
7. 001111010110010	16. 100001010011011	25. 110010001111010	
8. 010001111010110	17. 100011110101100	26. 110101100100011	

Abb. 6.64: Listen wir genau diejenigen Codewörter von RS(15,9) auf, die einem Binärvektor entsprechen, so erhalten wir den Code BCH(15,3).

### Satz 6.16

Es seien  $k$  eine natürliche Zahl,  $n = 2^k - 1$ ,  $\mathbb{F}$  ein endlicher Körper mit  $2^k$  Elementen und  $\alpha \in \mathbb{F}$  ein primitives Element. Dann ist

$$\text{BCH}_{\mathbb{F},\alpha}(n,\delta) = \text{RS}_{\mathbb{F},\alpha}(n,n-2\delta) \cap \mathbb{Z}_2^n$$

Abbildung 6.64 zeigt die Ausgabe eines einfachen Computerprogramms, das aus den Codewörtern von RS(15,9) diejenigen ausfiltert, die ein Symbol aus der Menge  $\mathbb{F} \setminus \{0,1\}$  enthalten. Die Ausgabe bestätigt unsere theoretischen Überlegungen: Es verbleiben genau die Codewörter aus Tabelle 6.7 übrig.

Wenn Sie Abschnitt 6.4.4 aufmerksam gelesen haben, so erinnern Sie sich sicher an eine elementare Eigenschaft, die alle BCH-Codes teilen: BCH-Codes sind zyklisch. Gilt dies vielleicht auch für die so eng verwandten Reed-Solomon-Codes? Die Antwort lautet Ja, wenn wir uns, wie oben vereinbart, auf die Codes  $\text{RS}_{\mathbb{F},\alpha}(n,k)$  beschränken, die über einem endlichen Körper mit genau  $n+1$  Elementen definiert sind.

Dass in diesem Fall ein zyklischer Code entsteht, lässt sich leicht einsehen. Ist

$$v = v_0 v_1 v_2 \dots v_{n-1}$$

ein Codewort, so existiert ein Polynom

$$p(x) = p_{n-1}x^{n-1} + \dots + p_1x + p_0$$

mit

$$p(\alpha^{n-1}) = v_0, p(\alpha^{n-2}) = v_1, \dots, p(\alpha^0) = v_{n-1}$$

Aus  $p(x)$  konstruieren wir jetzt das Polynom

$$q(x) := (p_{n-1}\alpha^{n-1})x^{n-1} + \dots + (p_1\alpha)x + p_0$$

Zwischen  $p(x)$  und  $q(x)$  besteht für  $i > 0$  der folgende Zusammenhang:

$$\begin{aligned} p(\alpha^i) &= p_{n-1}(\alpha^i)^{n-1} + \dots + p_1(\alpha^i) + p_0 \\ &= p_{n-1}(\alpha^{i-1})^{n-1}\alpha^{n-1} + \dots + p_1(\alpha^{i-1})\alpha + p_0 \\ &= (p_{n-1}\alpha^{n-1})(\alpha^{i-1})^{n-1} + \dots + (p_1\alpha)(\alpha^{i-1}) + p_0 \\ &= q(\alpha^{i-1}) \end{aligned}$$

Es gilt also:

$$\begin{array}{ccccccccc} & v_0 & & v_1 & & v_{n-3} & & v_{n-2} & & v_{n-1} \\ & \parallel & & \parallel & & \parallel & & \parallel & & \parallel \\ p(\alpha^n) & p(\alpha^{n-1}) & p(\alpha^{n-2}) & \dots & p(\alpha^2) & p(\alpha^1) & & p(\alpha^0) \\ \parallel & \parallel & \parallel & & \parallel & \parallel & & \parallel \\ q(\alpha^{n-1}) & q(\alpha^{n-2}) & q(\alpha^{n-3}) & \dots & q(\alpha^1) & q(\alpha^0) & & & \\ \parallel & \parallel & \parallel & & \parallel & \parallel & & & \\ ? & v_0 & v_1 & \dots & v_{n-3} & v_{n-2} & & & \end{array}$$

Jetzt fehlt nur noch ein winziges Puzzlestück: Wir müssen klären, mit welchem Symbol das durch  $q(x)$  beschriebene Codewort beginnt. An dieser Stelle kommt die weiter oben vereinbarte Voraussetzung ins Spiel, dass  $\mathbb{F}$  genau  $n+1$  Symbole enthält. Hieraus folgt, dass jedes primitive Element, und damit auch unser Element  $\alpha$ , die Beziehung

$$\alpha^n = 1$$

erfüllt. Somit ist

$$q(\alpha^{n-1}) = p(\alpha^n) = p(1) = p(\alpha^0) = v_{n-1},$$

und wir haben die Bedeutung von  $q(x)$  schwarz auf weiß vor Augen. Das Polynom beschreibt die rotierte Symbolfrequenz

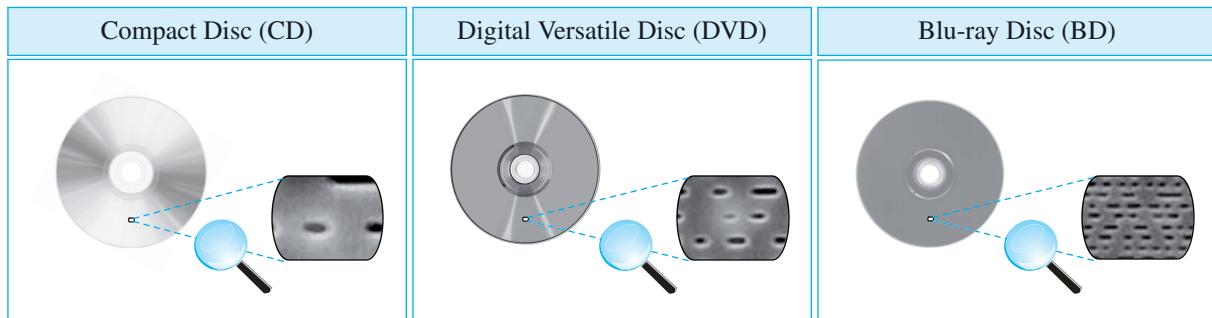
$$v \gg 1 = v_{n-1}v_0 \dots v_{n-3}v_{n-2},$$

die damit ebenfalls ein Codewort ist. Jetzt sind wir am Ziel: Wir haben den Reed-Solomon-Code als zyklisch identifiziert:



### Satz 6.17

Ist  $\mathbb{F}$  ein Körper mit  $n+1$  Elementen, dann ist  $\text{RS}_{\mathbb{F}, \alpha}(n, k)$  zyklisch.



**Abb. 6.65:** Auf der physikalischen Ebene unterscheiden sich CDs, DVDs und BDs hauptsächlich in der Strukturdichte. Bei allen drei sorgen Reed-Solomon-Codes dafür, dass die gespeicherten Daten selbst dann korrekt eingelesen werden können, wenn die Oberfläche durch Staubpartikel verschmutzt ist oder durch Kratzer beschädigt wurde.

#### 6.4.5.4 Cross-interleaved Reed-Solomon Code

An dieser Stelle unternehmen wir einen Ausflug in die Medientechnik, wo Reed-Solomon-Codes heute einen festen Platz eingenommen haben. Zum Einsatz kommen diese Codes bei nahezu allen optischen Datenspeichern, angefangen von der altbekannten *Compact Disc* (CD) über die *Digital Versatile Disc* (DVD) bis hin zur *Blu-ray Disc* (BD) (Abbildung 6.65). Schon bei der Audio-CD, der CD-DA, entschieden sich die Entwickler, die gespeicherten Daten mit einer Kaskade verschiedener Codierungen abzusichern, in deren Mitte zwei Reed-Solomon-Encoder arbeiten und geschickt miteinander interagieren. Zusammen ergeben die Codierungen den sogenannten *Cross-interleaved Reed-Solomon Code*, kurz CIRC, den wir uns nun genauer ansehen wollen.

Ausgangspunkt für die Codierung sind die digitalisierten Rohdaten einer Audioaufnahme, wie sie von einem *Analog-Digital-Wandler* geliefert werden (Abbildung 6.66). Anders als die DVD oder die BD, die Audio- und Videodaten in unterschiedlichen Formaten speichern können, lässt die CD-DA ausschließlich die Stereoaufzeichnung zu; wir haben es also stets mit zwei Audioströmen zu tun.

Zu Beginn werden jeweils 12 Audiosamples, von denen 6 dem linken und 6 dem rechten Stereokanal zugeordnet sind, zu einem Block zusammengefasst und gemeinsam verarbeitet. Konkret sieht ein solcher Block so aus:

$$L_0, R_0, L_1, R_1, L_2, R_2, L_3, R_3, L_4, R_4, L_5, R_5 \quad (6.55)$$

Auf einer Audio-CD werden die Samples mit einer Auflösung von 16 Bit gespeichert, d. h., jeder Wert in (6.55) wird durch 2 Byte dargestellt.

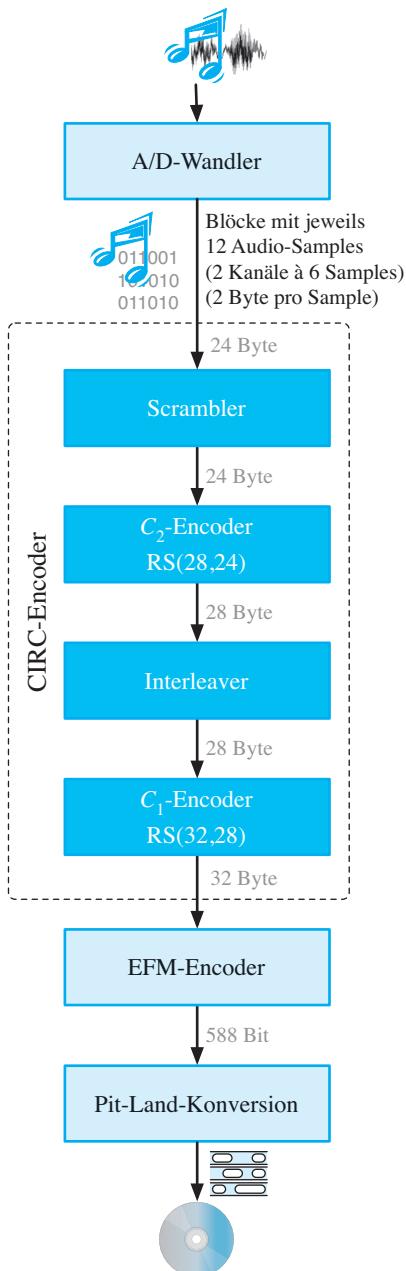


Abb. 6.66: Die verschiedenen Encoder-Stufen der Audio-CD

Im Folgenden steht  $L_i.A$  und für das *High Byte* und  $L_i.B$  für das *Low Byte* von  $L_i$ . Die gleiche Notation benutzen wir für  $R_i$ .

Die 24 Byte eines Blocks bilden zusammen einen *F1-Frame*. Dieser ist die Eingabe für den CIRC-Encoder, dessen detaillierter Aufbau in Abbildung 6.67 zu sehen ist (vgl. [22]). Insgesamt durchläuft jeder F1-Frame die folgenden Stufen:

#### ■ Scrambler

Im ersten Schritt werden die Daten räumlich und zeitlich gespreizt. Die räumliche Spreizung entsteht durch eine einfache Umverdrahtung, und die zeitliche Spreizung wird durch zusätzlich eingefügte Verzögerungsglieder erreicht. Ein Blick auf die Schemazeichnung in Abbildung 6.67 zeigt, dass die Samples  $(L_0, R_0)$ ,  $(L_2, R_2)$  und  $(L_4, R_4)$  um genau zwei Zeiteinheiten verzögert werden.

Die Umverteilung der Daten hat den Zweck, zusammengehörende Audio-Samples auf der CD-Oberfläche möglichst weit voneinander entfernt zu speichern. Dies macht die Codierung robuster gegen *Burstfehler*, bei denen eine lange Kette physikalisch nebeneinander liegender Bits beschädigt wird. Burstfehler sind bei optischen Speichermedien allgegenwärtig; sie werden bereits durch kleine Kratzer auf dem Speichermedium verursacht und treten entsprechend häufig auf.

#### ■ C<sub>2</sub>-Encoder

Der  $C_2$ -Code ist ein  $RS_{\mathbb{F}_{2^8}}(28,24)$ -Code. Es handelt sich also um einen Reed-Solomon-Code über einem Körper mit 256 Elementen und der Codewortlänge 28. Der  $C_2$ -Encoder nimmt jeweils Blöcke von 24 Byte entgegen und bildet diese auf Blöcke von jeweils 28 Byte ab. Da die Ausgabe um 4 Byte länger ist als die Eingabe, ist der  $C_2$ -Code 2-fehlerkorrigierend.

Aus der Schemazeichnung in Abbildung 6.67 geht hervor, dass die beiden Reed-Solomon-Decoder lediglich 4 Prüfbytes hinzufügen und den Rest des Datenstroms unverändert lassen. Das bedeutet in der Nomenklatur der Codierungstheorie, dass zwei *systematische* Reed-Solomon-Codes verwendet werden. Beachten Sie bei dieser Gelegenheit darauf, die Schemazeichnung richtig zu interpretieren. Auf jeder eingezeichneten Leitung wird ein Element aus dem Körper  $\mathbb{F}_{2^8}$  transportiert; jede eingezeichnete Verbindung steht also stellvertretend für 8 separate Leitungen.

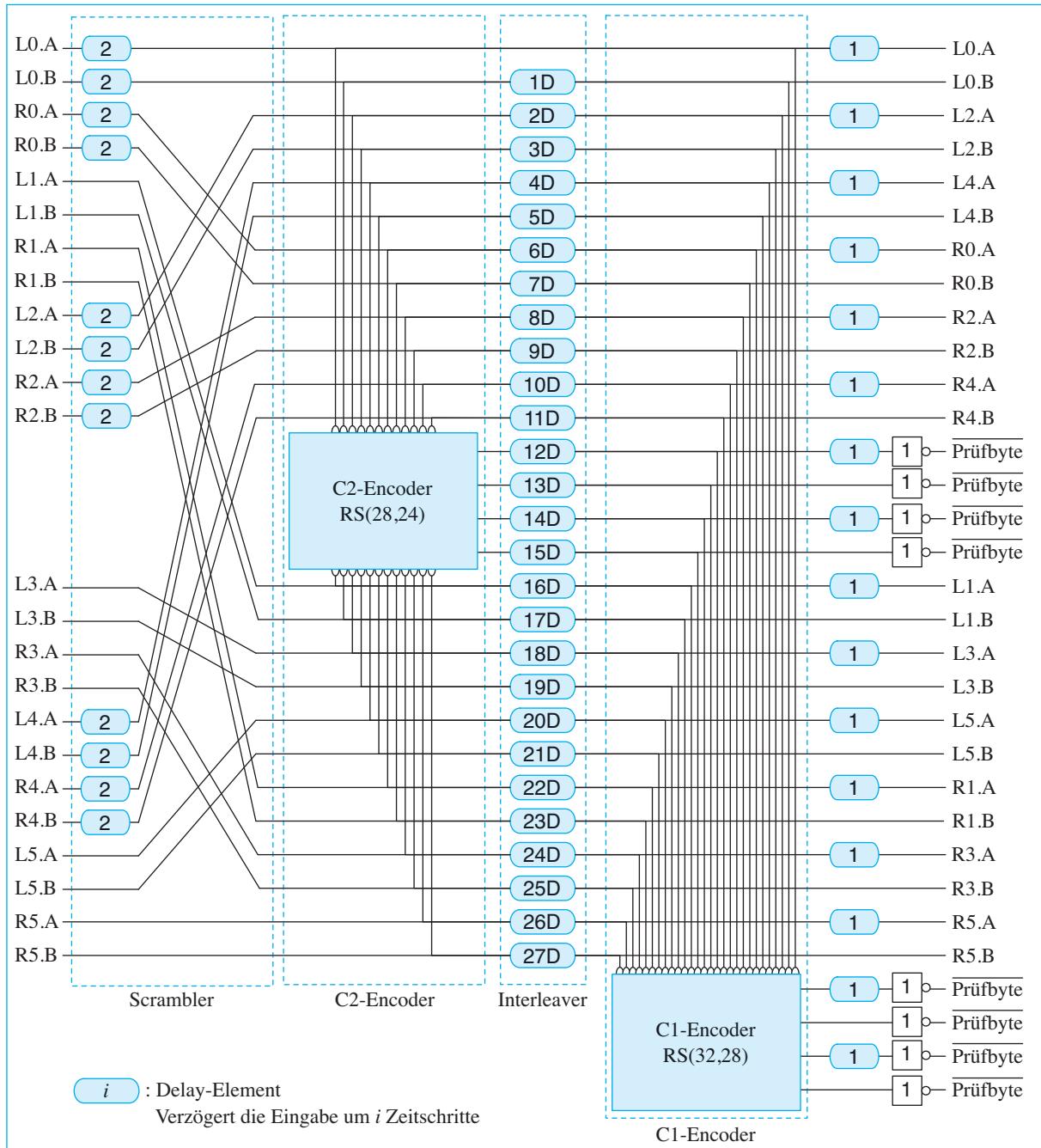
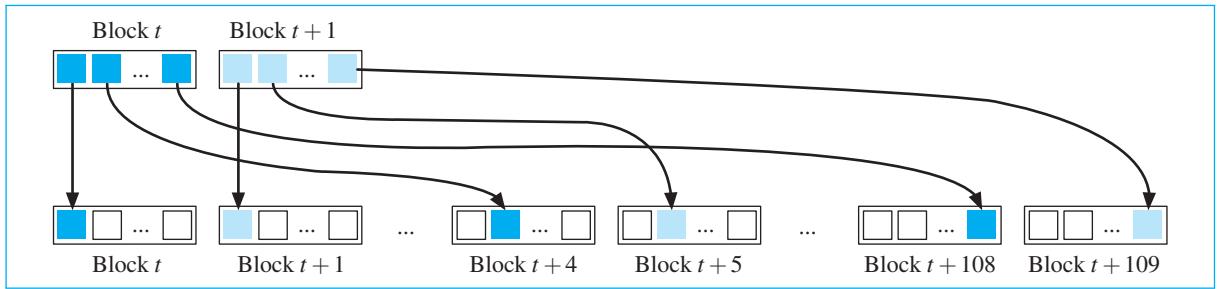


Abb. 6.67: Schematischer Aufbau eines CIRC-Encoders



**Abb. 6.68:** Der Interleaver sorgt dafür, dass die Daten eines Blocks auf verschiedene Blöcke verteilt werden und damit auf der CD nicht physikalisch nebeneinander stehen.

#### ■ Interleaver

Die 28 Byte des  $C_2$ -Encoders werden anschließend durch den *Interleaver* geleitet, der für die eigentliche zeitliche Spreizung der Daten sorgt. In Abbildung 6.67 sind die Verzögerungszeiten als Vielfache eines Faktors  $D$  angegeben. Bei der Audio-CD wurde der Wert  $D = 4$  gewählt, sodass die folgende Umverteilung entsteht (Abbildung 6.68):

- Byte 1 von Block  $t$  wird Byte 1 von Block  $t$  zugeordnet,
- Byte 2 von Block  $t$  wird Byte 2 von Block  $t + 4$  zugeordnet,
- ...
- Byte 28 von Block  $t$  wird Byte 28 von Block  $t + 108$  zugeordnet.

Im Ergebnis sorgt der Interleaver dafür, dass die gleichzeitig produzierten Ausgabewerte des  $C_2$ -Encoders jeweils vier Blöcke voneinander entfernt sind.

#### ■ $C_1$ -Encoder

Der  $C_1$ -Code ist ein  $\text{RS}_{\mathbb{F}_{2^8}}(32,28)$ -Code. Genau wie  $C_2$  ist auch  $C_1$  ein Reed-Solomon-Code über einem Körper mit 256 Elementen. Unterschiedlich sind die Codewortlängen. Der  $C_1$ -Encoder nimmt die 28 Byte des Interleavers entgegen und bildet diese auf Blöcke von jeweils 32 Byte ab. Genau wie  $C_2$  ist also auch  $C_1$  ein 2-fehlerkorrigierender Code.

Dem  $C_1$ -Encoder ist ein zweiter Interleaver nachgeschaltet, der alle High Bytes um eine Zeiteinheit verzögert und hierdurch für eine weitere zeitliche Spreizung der Daten sorgt. Zusätzlich werden durch mehrere nachgeschaltete Inverter alle Prüfbytes invertiert, um zu verhindern, dass ein gültiges Codewort entsteht, wenn alle Signalleitungen gleich 0 sind.

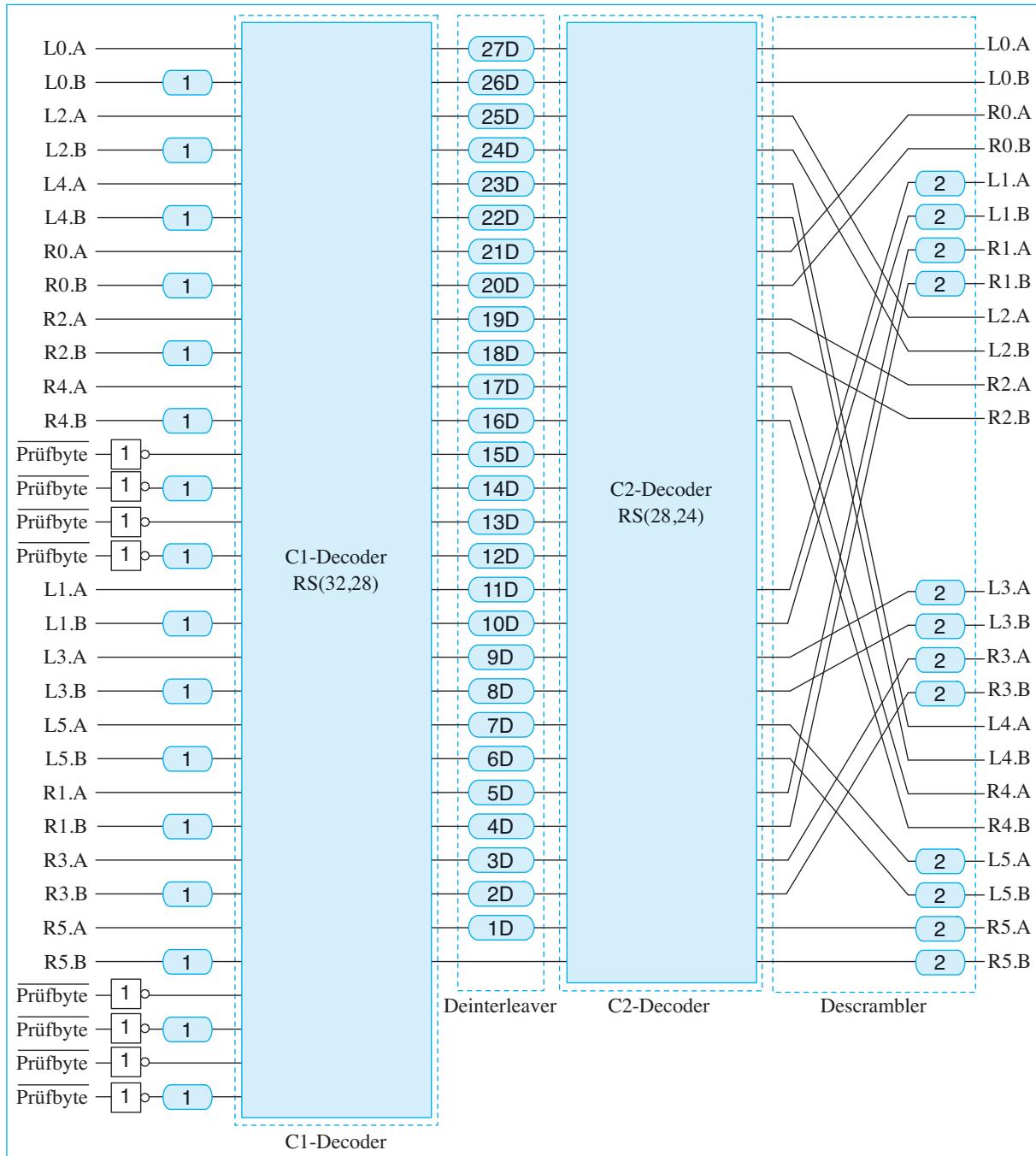
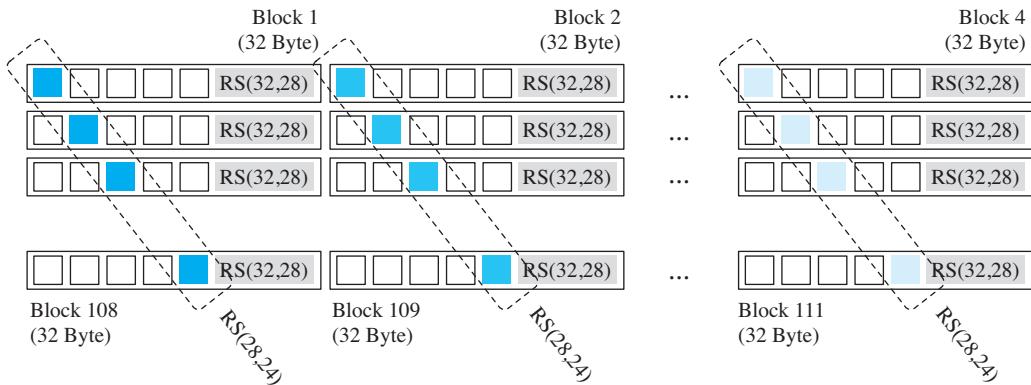
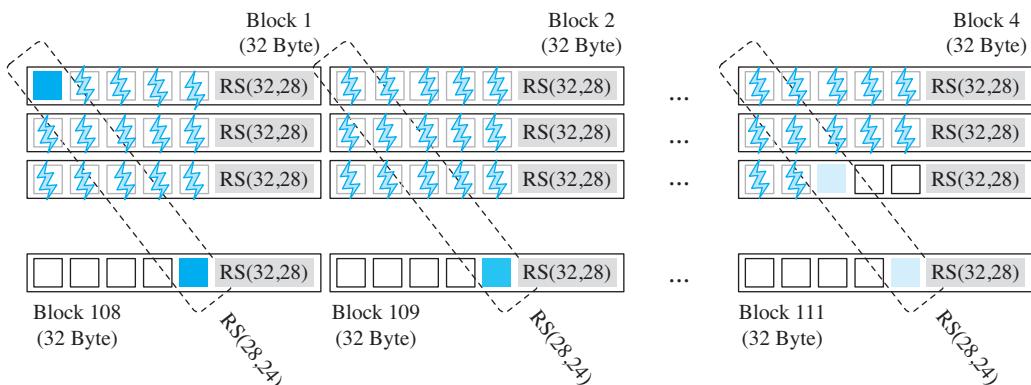


Abb. 6.69: Schematischer Aufbau eines CIRC-Decoders



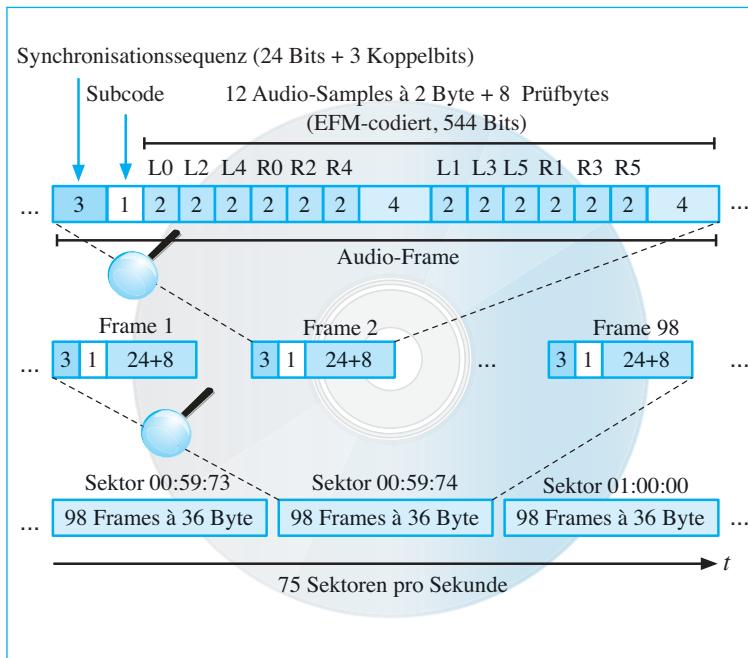
**Abb. 6.70:** Das geschickte Zusammenspiel zwischen den beiden Reed-Solomon-Codes und dem Interleaver ist der Grund, dass CDs auch bei tieferen Kratzern lesbar bleiben. Durch den Interleaver werden die konsekutiv beschädigten Datenpakete auf verschiedene Blöcke verteilt und lassen sich dort separat korrigieren.



**Abb. 6.71:** Durch die zweidimensionale Absicherung der Datenpakete lassen sich auch Burstfehler korrigieren. In diesem Beispiel enthalten die  $C_2$ -Codewörter immer noch genug Information, um eine fehlerfreie Rekonstruktion zu erreichen.

Abbildung 6.69 zeigt, dass die Decodierung eines Datenstroms mit einer Schaltung durchgeführt werden kann, die ganz ähnlich aufgebaut ist.

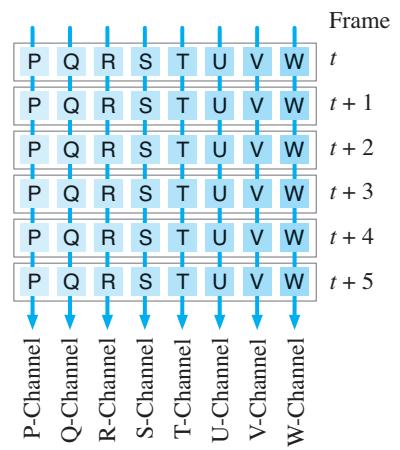
Um einen Eindruck von der Leistungsfähigkeit dieses Decoders zu bekommen, werfen wir einen Blick auf Abbildung 6.70. Dort ist darge-



**Abb. 6.72:** Frame-Struktur einer Audio-CD. Ein Audio-Frame besteht aus 588 Bits. Die ersten 24 bilden eine spezielle Synchronisationssequenz, die mit 3 zusätzlichen Koppelbits abgeschlossen wird. Die restlichen 561 Bits codieren 33 Bytes im EFM-Format. Das erste Byte ist der Subcode; jedes Bit gehört zu einem der 8 Subchannels, in denen Metainformationen wie das Inhaltsverzeichnis oder der Anfang eines Tracks codiert sind. Die restlichen 32 Bytes setzen sich aus 12 Audio-Samples zu jeweils 2 Byte und 8 Reed-Solomon-Prüfbytes zusammen. 4 der 8 Prüfbytes werden durch den  $C_1$ -Encoder erzeugt und die anderen vier durch den  $C_2$ -Encoder. 98 Frames bilden einen logischen Sektor. Pro Sekunde werden 75 Sektoren abgespielt, was für jeden Stereokanal 44100 Audio-Samples pro Sekunde bedeutet.

stellt, wie die Daten, die der  $C_2$ -Encoder liefert, nach der Bearbeitung durch den  $C_1$ -Encoder aussehen. In Kombination mit dem Interleaver sorgen die beiden Reed-Solomon-Encoder dafür, dass wir uns die Datenbytes zweidimensional angeordnet vorstellen dürfen. Horizontal sind die Datenbytes über den  $C_1$ -Code abgesichert. Zusätzlich sorgt der  $C_2$ -Code für eine Absicherung auf den von links oben nach rechts unten verlaufenden Diagonalen.

Tritt beim Einlesen eines Blocks ein vereinzelter Lesefehler auf, so kann dieser in vielen Fällen direkt durch den  $C_1$ -Decoder entfernt werden. Wurde eine lange Kette nebeneinanderliegender Daten durch einen Burstfehler beschädigt, wie es in Abbildung 6.71 exemplarisch dargestellt ist, so kann der  $C_1$ -Decoder nichts mehr ausrichten. Da die defekten Bits, durch den Interleaver verursacht, nicht zu den gleichen logischen Blöcken gehören, können diese Fehler in den meisten Fällen durch den  $C_2$ -Decoder entfernt werden. In der Schemazeichnung in Abbildung 6.69 ist dieses Verhalten ebenfalls gut zu erkennen. Tritt ein Burstfehler auf, so liegen an den Ausgängen des  $C_1$ -Decoders überall falsche Werte an. Durch den nachgeschalteten Deinterleaver erreichen die defekten Datenpakete den  $C_2$ -Decoder aber zeitversetzt und können dort nacheinander einzeln korrigiert werden.



**Abb. 6.73:** Jedes Subcodebyte enthält 8 Bits, die verschiedenen Subchannels zugeordnet sind. Verwendet werden diese Datenkanäle für die Codierung von Metainformationen.

	$u$	$c(u)$		$u$	$c(u)$	
0	00000000	01001000100000		231	11100111	00100100010010
1	00000001	10000100000000		232	11101000	10000100000010
2	00000010	10010000100000		233	11101001	10000100000100
3	00000011	10001000100000		234	11101010	00001001001001
4	00000100	01000100000000		235	11101011	00001001000010
5	00000101	00000100010000		236	11101100	01000100000100
6	00000110	00010000100000		237	11101101	00000100000100
7	00000111	00100100000000		238	11101110	00010000100010
8	00001000	01001001000000		239	11101111	00100100000100
9	00001001	10000001000000		240	11110000	00000100100010
10	0001010	10010001000000		241	11110001	10000010010010
11	0001011	10001001000000		242	11110010	10010010010010
12	0001100	01000001000000		243	11110011	00001000100010
13	0001101	00000001000000		244	11110100	01000010010010
14	0001110	00010001000000		245	11110101	00000010010010
15	0001111	00100001000000		246	11110110	00010010010010
16	0010000	10000000100000		247	11110111	00100010010010
17	0010001	10000010000000		248	11111000	01001000010010
18	0010010	10010010000000		249	11111001	10000000010010
19	0010011	00100000100000		250	11111010	10010000010010
20	0010100	01000010000000		251	11111011	10001000010010
21	0010101	00000010000000		252	11111100	01000000010010
22	0010110	00010010000000		253	11111101	00001000010010
23	0010111	00100010000000		254	11111110	00010000010010
...						
				255	11111111	00100000010010

**Tab. 6.9:** Ausschnitt aus der 256 Einträge umfassenden EFM-Tabelle

### Logischer Aufbau einer CD

Die 32 Bytes, die der CIRC-Encoder produziert, werden auf einer CD in einem Audio-Frame gespeichert. Wie ein solcher Frame aufgebaut ist, zeigt Abbildung 6.72. Zunächst wird den 32 Bytes des CIRC-Encoders, die zusammen einen *F2-Frame* bilden, ein *Subcodebyte* vorangestellt. Das Ergebnis ist eine 33 Byte lange Sequenz, die als *F3-Frame* bezeichnet wird. Jedes Subcodebit gehört zu einem von 8 *Subchannels*, die Metainformation codieren (Abbildung 6.73). Beispielsweise wird

der P-Channel dazu benutzt, um den Anfang eines Tracks zu markieren, und der Q-Channel wird für die Codierung des Inhaltsverzeichnisses verwendet.

### EFM-Codierung

Die 33 Bytes eines F3-Frames werden mit einem *EFM-Codierer* weiter verarbeitet. Aus der Abkürzung EFM, die ausgeschrieben *Eight-to-Fourteen-Modulation* bedeutet, können wir die grundlegende Funktionsweise des Codierers ableiten: Er nimmt 8 Datenbits entgegen und bildet diese auf Codewörter der Länge 14 ab. Tabelle 6.9 zeigt einen Ausschnitt aus der insgesamt 256 Einträge umfassenden Tabelle und macht deutlich, dass die EFM-Codewörter hauptsächlich aus Nullen bestehen. Insgesamt sind die Codewörter so strukturiert, dass die folgenden beiden Bedingungen erfüllt sind:

- Zwischen zwei Einsen liegen mindestens zwei Nullen. (EFM1)
- Spätestens nach zehn Nullen folgt eine Eins. (EFM2)

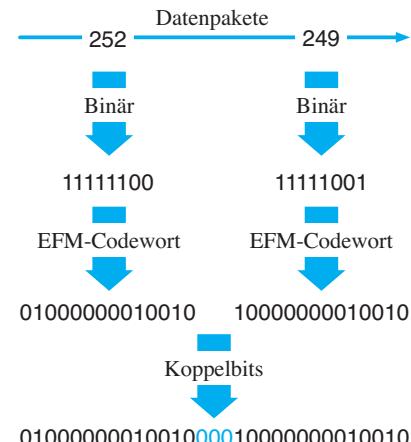
Auf einer CD werden zwei EFM-Codewörter niemals direkt hintereinander gespeichert, sondern durch 3 *Koppelbits* getrennt. Da die Koppelbitsequenzen so gewählt werden müssen, dass die Bedingungen (EFM1) und (EFM2) auch codewortübergreifend erfüllt sind, kommen nur vier mögliche Sequenzen in Frage:

000,001,010,100

Abbildung 6.74 demonstriert die EFM-Codierung an einem konkreten Beispiel. Zunächst werden die Eingabewerte in ihre Binärdarstellung und anschließend in die entsprechenden EFM-Codewörter übersetzt. Danach werden die EFM-Codewörter durch drei Koppelbits miteinander verbunden. Beachten Sie, dass in unserem Beispiel die Koppelbitsequenz 000 die einzige mögliche ist; alle anderen würden ein Bitmuster produzieren, in denen zwei Einsen zu nahe beieinander liegen.

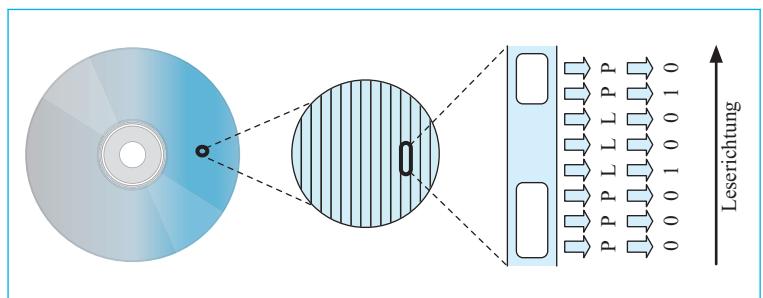
Durch die EFM-Codierung wachsen die 264 Bits eines F3-Frames auf 561 Bits an. Insgesamt besteht ein Audio-Frame aus 588 Bits, von denen die letzten 561 Bits jene sind, die der EFM-Codierer erzeugt. Die ersten 24 Bits sind eine spezielle Synchronisationssequenz, die den EFM-codierten Daten vorangestellt wird und durch zusätzliche 3 Koppelbits von diesen getrennt ist. Diese Bitsequenz lautet folgendermaßen:

$$1000000000010000000000010 \quad (6.56)$$



**Abb. 6.74:** Bei der EFM-Codierung werden 8 Bits auf 14 Bits erweitert und durch 3 zusätzliche Koppelbits getrennt.

**Abb. 6.75:** Pit-Land-Codierung des EFM-codierten Datenstroms. Bei der Abtastung der CD-Oberfläche registriert der Laser eine Abfolge von Vertiefungen (*Pits*, P) und Erhöhungen (*Lands*, L). Eine binäre 1 wird codiert, indem von P nach L oder von L nach P gewechselt wird. Folgt dagegen auf ein P ein weiteres P oder auf ein L ein weiteres L, so entspricht dies einer binären 0.



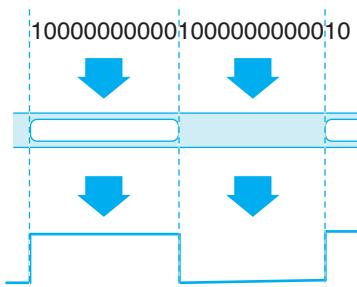
Die Synchronisationssequenz übernimmt gleich mehrere Aufgaben. Zum einen wird sie benötigt, um den Anfang eines Frames eindeutig zu markieren. Zum anderen wird sie von der Steuerung des Servomotors ausgewertet, um die CD mit der korrekten Geschwindigkeit rotieren zu lassen.

Auf der logischen Ebene werden die einzelnen Frames, wie es weiter oben, in Abbildung 6.72, bereits zu sehen war, zu größeren Einheiten zusammengefasst. Jeweils 98 Frames bilden einen *Sektor*, die kleinste Adressierungseinheit einer Audio-CD. In einer Sekunde spielt ein Wiedergabegerät 75 Sektoren ab, und daraus können wir sofort auf die Abtastrate einer CD rückschließen. In jedem Sektor sind

$$98 \times 6 = 588$$

Stereo-Audio-Samples gespeichert, sodass in einer Sekunde  $75 \times 588 = 44100$  Samples abgespielt werden. Dies ergibt die bekannte Abtastrate einer Audio-CD von

$$1,44 \text{ kHz}.$$



**Abb. 6.76:** Jeder Audio-Frame beginnt mit einer 24 Bit langen Synchronisationssequenz, die bei der Abtastung ein Rechtecksignal erzeugt.

Physikalisch werden die Daten auf einer CD gespeichert, indem bei der Produktion mikroskopisch kleine Vertiefungen in das Trägermaterial gepresst werden. Die Vertiefungen werden als *Pits* (P) bezeichnet und die Zwischenräume als *Lands* (L). Pits besitzen die Eigenschaft, den einfallenden Laserstrahl zu streuen, während Lands den Lichtstrahl gut reflektieren. Beim Lesen des Datenträgers wird der P-L-Datenstrom zunächst in einen binären Datenstrom aus Nullen und Einsen übersetzt. Jeder Übergang von P nach L oder von L nach P entspricht einer Eins, ansonsten wird eine Null codiert (Abbildung 6.75).

Damit ist auch klar, was sich hinter der Synchronisationssequenz (6.56) in Wahrheit verbirgt. Physikalisch wird sie durch zwei gleich lange Folgen von Pits und Lands codiert, die bei der Abtastung ein gleichförmiges Rechtecksignal erzeugen (Abbildung 6.76).

## 6.4.6 Hadamard-Codes

Die Codes, für die wir uns in diesem Abschnitt interessieren, basieren auf den Eigenschaften spezieller Matrizen, die in der Literatur als *Hadamard-Matrizen* bezeichnet werden. Ausführlich untersucht wurden diese erstmals im Jahr 1867, in einer Arbeit des englischen Mathematikers James Joseph Sylvester [90] (Abbildung 6.77). Im Jahr 1893 nahm sich der französische Mathematiker Jacques Hadamard den Matrizen an, die Sylvester in seiner Originalarbeit noch etwas holprig als *anallagmatic pavements* bezeichnete. Eine von ihm entdeckte Determinantenbeziehung machte die Matrizen bekannt und verknüpfte seinen Namen schließlich so eng mit ihnen, dass in der modernen Literatur heute fast durchweg von *Hadamard-Matrizen* gesprochen wird [38].

Die folgende Definition klärt, um welche speziellen mathematischen Objekte es sich dabei handelt:



### Definition 6.15 (Hadamard-Matrix)

Eine  $n \times n$ -Matrix  $H$ , in der nur die Elemente  $-1$  und  $1$  vorkommen, heißt *Hadamard-Matrix der Ordnung  $n$* , wenn sie die Beziehung

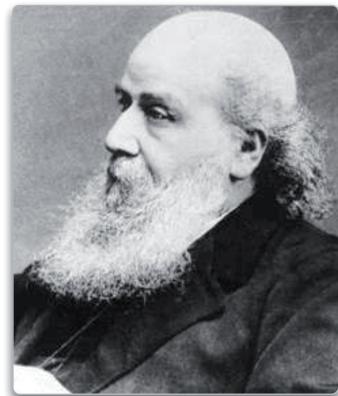
$$HH^T = nE$$

erfüllt. Hierin steht  $E$  für die  $n \times n$ -Einheitsmatrix.

Die Eigenschaft  $HH^T = nE$  bedeutet das Folgende: Wählen wir zwei unterschiedliche Zeilen oder zwei unterschiedliche Spalten einer Hadamard-Matrix aus, so ist deren Skalarprodukt stets 0. Mit anderen Worten: Alle Zeilen einer Hadamard-Matrix sind paarweise orthogonal, und das Gleiche gilt auch für deren Spalten. Diese Eigenschaft macht deutlich, dass Hadamard-Matrizen eine enge Beziehung zu jenen Objekten pflegen, die in der Mathematik als *orthogonale Matrizen* bezeichnet werden. Beide unterscheiden sich nur darin, dass die Multiplikation einer orthogonalen Matrix  $M$  mit  $M^T$  immer die Einheitsmatrix  $E$  ergibt, und die Multiplikation einer Hadamard-Matrix  $H$  mit  $H^T$  das Ergebnis  $nE$  liefert, wobei  $n$  die Ordnung von  $H$  ist. Wir finden auf der Hauptdiagonalen der Produktmatrix also nicht das Element 1, sondern das Element  $n$  wieder.

Die Matrix

$$\begin{pmatrix} -1 & -1 \\ 1 & -1 \end{pmatrix}$$



James Joseph Sylvester  
(1814 – 1897)



Jacques Salomon Hadamard  
(1865 – 1963)

**Abb. 6.77:** James Joseph Sylvester ist der Schöpfer und Jacques Salomon Hadamard der Namensgeber einer Klasse von Matrizen, die in der Codierungstheorie eine prominente Rolle spielen.

ist eine Hadamard-Matrix der Ordnung 2 und

$$\begin{pmatrix} 1 & -1 & -1 & -1 \\ 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & 1 & -1 \end{pmatrix}$$

eine Hadamard-Matrix der Ordnung 4. Dass beide die in Definition 6.15 geforderte Eigenschaft erfüllen, können wir durch einfaches Nachrechnen überprüfen. Für die erste Matrix erhalten wir:

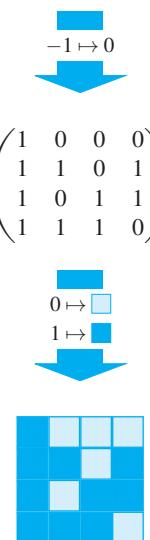
$$\begin{pmatrix} -1 & -1 \\ 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} -1 & 1 \\ -1 & -1 \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$$

Für die zweite Matrix gilt:

$$\begin{pmatrix} 1 & -1 & -1 & -1 \\ 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & 1 & 1 & -1 \end{pmatrix} = \begin{pmatrix} 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix}$$

Da in einer Hadamard-Matrix lediglich die Elemente  $-1$  und  $1$  vorkommen, können zwei Zeilenvektoren in Matrizen der Ordnung  $n > 1$  nur dann orthogonal zueinander sein, wenn  $n$  eine gerade Zahl ist. Im Übungsteil werden wir diese Beziehung noch enger fassen. Dort werden Sie zeigen, dass die Ordnung einer Hadamard-Matrix gleich 1, gleich 2 oder durch 4 teilbar ist.

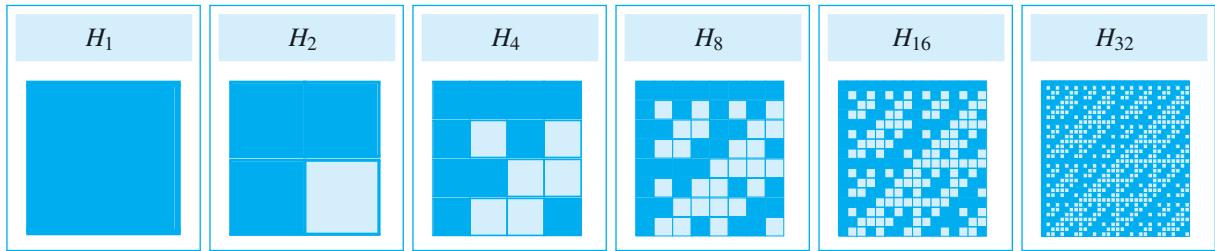
Überdies machen unsere Überlegungen klar, dass wir die geforderte Eigenschaft  $HH^T = nE$  durch die folgende, äquivalente Charakterisierung ersetzen können:



Eine  $n \times n$ -Matrix  $H$ , in der nur die Elemente  $-1$  und  $1$  vorkommen, ist eine *Hadamard-Matrix*, wenn zwei Zeilen an jeweils  $\frac{n}{2}$  Positionen übereinstimmen.

Diese Form der Formulierung macht klar, dass wir auch dann von Hadamard-Matrizen sprechen können, wenn in einer Matrix anstelle der Elemente  $-1$  und  $1$  zwei ganz andere Symbole vorkommen. Diesen Umstand machen wir uns zu Nutze und ersetzen die Zahl  $-1$  durch die Zahl  $0$ . Auf diese Weise wird aus einer Hadamard-Matrix eine Matrix über dem Körper  $\mathbb{Z}_2$ , mit dem wir bestens vertraut sind. Abbildung 6.78 demonstriert die Konvertierung an einem konkreten Beispiel und führt zugleich eine grafische Notation ein, die im Zusammenhang

**Abb. 6.78:** Jede Hadamard-Matrix lässt sich eins zu eins in eine Matrix über dem Körper  $\mathbb{Z}_2$  übersetzen. Um die geometrische Struktur hervorzuheben, werden die Nullen und Einsen unten durch unterschiedlich eingefärbte Quadrate dargestellt.



**Abb. 6.79:** Rekursive Konstruktion von Hadamard-Matrizen nach der Vorschrift von Joseph Sylvester aus dem Jahr 1867

mit Hadamard-Matrizen gerne verwendet wird. In dieser wird das Element 0 ganz einfach durch ein hell gefärbtes Quadrat dargestellt und das Element 1 durch ein dunkel gefärbtes.

Hadamard-Matrizen lassen sich sehr einfach konstruieren, wenn ihre Ordnung eine Zweierpotenz ist. In diesem Fall können wir auf eine rekursive Konstruktionsvorschrift zurückgreifen, die aus der Feder von James Joseph Sylvester stammt:

$$H_1 := (1) \quad (6.57)$$

$$H_{2n} := \begin{pmatrix} H_n & H_n \\ H_n & -H_n \end{pmatrix} \quad (6.58)$$

Abbildung 6.79 zeigt die ersten 6 Matrizen, die dieses Rekursionsschema hervorbringt.

Die Eigenschaft zweier beliebig herausgegriffener Zeilen, sich genau an der Hälfte der Positionen zu unterscheiden, macht die Hadamard-Matrizen für die Codierungstheorie interessant und führt uns auf direktem Weg zur Definition der Hadamard-Codes: Ein Hadamard-Code entsteht, indem aus einer Hadamard-Matrix  $H$  der Ordnung  $2^k$  eine neue Matrix

$$C = \begin{pmatrix} H \\ -H \end{pmatrix}$$

konstruiert wird und die Zeilen von  $C$  als Codewörter der Länge  $2^k$  interpretiert werden. Da wir sowohl  $H$  als auch  $-H$  verwenden, stehen  $2 \cdot 2^k = 2^{k+1}$  Codewörter zur Verfügung, und das bedeutet, dass wir aus jeder Hadamard-Matrix der Ordnung  $2^k$  eine Codierung der Form

$$c : \{0,1\}^{k+1} \rightarrow \{0,1\}^{2^k}$$

ableiten können. Die Zuordnung einer Nachricht  $u$  zu einem Codewort gelingt am einfachsten, wenn wir die Nachricht als die binäre Darstellung der Matrixzeile interpretieren, aus der das Codewort ausgelesen

Hadamard-Code der Ordnung 2		Hadamard-Code der Ordnung 4		Hadamard-Code der Ordnung 8	
$u$	$c(u)$	$u$	$c(u)$	$u$	$c(u)$
00	11	000	1111	0000	11111111
01	10	001	1010	0001	10101010
10	00	010	1100	0010	11001100
11	01	011	1001	0011	10011001
		100	0000	0100	11110000
		101	0101	0101	10100101
		110	0011	0110	11000011
		111	0110	0111	10010110
				1000	00000000
				1001	01010101
				1010	00110011
				1011	01100110
				1100	00001111
				1101	01011010
				1110	00111100
				1111	01101001

Abb. 6.80: Hadamard-Codes der Ordnungen 2, 4 und 8

werden kann. Der Wert  $u = 0$  entspricht dabei der ersten Zeile, der Wert  $u = 1$  der zweiten und so fort.

Abbildung 6.80 zeigt die Codierungen, die wir auf diese Weise aus den weiter oben berechneten Matrizen  $H_2$ ,  $H_4$  und  $H_8$  ableiten können.

Während der Hadamard-Code  $H_2$  keine praktische Bedeutung hat, treffen wir im Falle von  $H_4$  auf einen Code, den wir gut kennen. Es handelt sich um den geraden Paritätscode, der genau diejenigen Codewörter umfasst, die eine gerade Anzahl Einsen aufweisen. Schon bald werden wir zeigen, dass sich auch hinter  $H_8$  ein bekannter Code verbirgt.

Weiter oben haben wir dargelegt, dass sich zwei Zeilen einer Hadamard-Matrix an genau  $\frac{n}{2}$  Positionen unterscheiden. Diese Distanzüberlegung können wir problemlos auf Hadamard-Codes der Ordnung  $n$  übertragen. Vergleichen wir zwei verschiedene Zeilen aus der Codematrix  $C$  miteinander, so unterscheiden sich diese entweder an  $\frac{n}{2}$  oder an  $n$  Positionen. Den ersten Fall dürfen wir ungeniert als den Normalfall bezeichnen, da der zweite nur dann eintritt, wenn wir eine Zeile aus  $H$  mit ihrer komplementären Zeile aus  $-H$  vergleichen.

Bezeichnen wir die  $i$ -te Zeile der Codematrix  $C$  mit  $r_i$ , dann können wir diesen Zusammenhang formal so aufschreiben.

$$\Delta(r_i, r_{i+k}) = \begin{cases} 0 & \text{falls } k = 0 \\ n & \text{falls } k = n \\ \frac{n}{2} & \text{sonst} \end{cases} \quad (6.59)$$

Die Code-Distanz eines Hadamard-Codes können wir sofort aus Gleichung (6.59) ablesen: Sie beträgt  $\frac{n}{2} = 2^{k-1}$ , und unter Berücksichtigung von Satz 6.3 folgt hieraus, dass  $\frac{n}{4} - 1 = 2^{k-2} - 1$  Übertragungsfehler korrigiert werden können. Werden bei der Übertragung  $\frac{n}{4}$  Bits verfälscht, so lässt sich der Fehler nur noch erkennen, da die empfangene Bitsequenz in diesem Fall von mehreren Codewörtern gleich weit entfernt ist.

Eine mögliche Art der Fehlerkorrektur ist in Abbildung 6.81 beschrieben. Dort wird die Zeile mit der höchsten Übereinstimmung ermittelt, indem die empfangene Bitsequenz von links mit der transponierten Hadamard-Matrix multipliziert wird. Unter der Annahme, dass maximal 1 Bit verfälscht wurde, muss der Ergebnisvektor in dem gezeigten Beispiel an einer eindeutig bestimmten Position  $i$  den Wert 6 oder den Wert -6 annehmen. Im ersten Fall entspricht das ursprünglich gesendete Codewort der  $i$ -ten Zeile von  $H$  und im zweiten Fall der  $i$ -ten Zeile von  $-H$ . Nachdem die Zeilennummer bestimmt ist, können wir die gesendete Nachricht rekonstruieren, indem wir den Wert von  $i$  bzw. den Wert von  $i + n$  ganz einfach in das Binärsystem übersetzen.

Werden zwei Bits verfälscht, so lässt sich der Fehler zwar noch erkennen, aber nicht mehr korrigieren. Warum dies so ist, macht das Beispiel in Abbildung 6.82 deutlich. In dem gezeigten Fehlerszenario gibt es



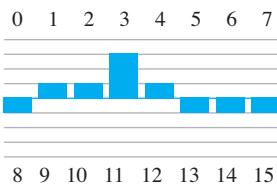
Im Übungsteil auf Seite 503 werden wir herleiten, dass die Ordnung  $m$  einer Hadamard-Matrix gleich 1, gleich 2 oder ein Vielfaches von 4 sein muss. Für  $m > 2$  ist die Teilbarkeit durch 4 demnach eine notwendige Eigenschaft. Dies wirft die Frage auf, ob sie auch hinreichend ist: Gibt es für jede durch 4 teilbare Zahl eine Hadamard-Matrix mit dieser Ordnung? Im Jahr 1933 hat der englische Mathematiker Raymond Paley in seiner bedeutenden Arbeit „On Orthogonal Matrices“ die Vermutung aufgestellt, dass dies tatsächlich der Fall ist. Paley schreibt:

*„It seems probable that, whenever  $m$  is divisible by 4, it is possible to construct an orthogonal matrix of order  $m$  composed of  $\pm 1$ , but the general theorem has every appearance of difficulty.“ [69]*

Paleys Vermutung ist das, was wir heute als die *Hadamard-Vermutung* bezeichnen. In der Vergangenheit wurden viele Versuche unternommen, sie zu beweisen, eine endgültige Klärung steht aber bis heute aus. Vollständig verifiziert ist die Aussage lediglich für Matrizen bis zur Ordnung 664. Die nächste durch 4 teilbare Zahl ist die Zahl 668. Bis heute ist offen, ob eine Hadamard-Matrix dieser Ordnung tatsächlich existiert.

$(u = 0011) \mapsto (v = 10011001) \mapsto (w = 10011000)$  (Übertragungsfehler an der letzten Position)

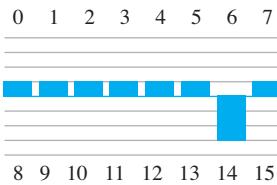
$$\begin{aligned} w \cdot H^T &= (1 \quad -1 \quad -1 \quad 1 \quad 1 \quad -1 \quad -1 \quad -1) \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix} \\ &= (-2 \quad 2 \quad 2 \quad 6 \quad 2 \quad -2 \quad -2 \quad 2) \end{aligned}$$



- ⇒ Gesendet wurde das Codewort an der Indexposition 3 (vierte Zeile von  $H$ )
- ⇒ Die korrigierte Nachricht ist das Binärmuster der Zahl 3
- ⇒ Ergebnis: 0011

$(u = 1110) \mapsto (v = 00111100) \mapsto (w = 10111100)$  (Übertragungsfehler an der ersten Position)

$$\begin{aligned} w \cdot H^T &= (1 \quad -1 \quad 1 \quad 1 \quad 1 \quad -1 \quad -1) \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix} \\ &= (2 \quad 2 \quad 2 \quad 2 \quad 2 \quad 2 \quad -6 \quad 2) \end{aligned}$$



- ⇒ Gesendet wurde das Codewort an der Indexposition 14 (siebte Zeile von  $-H$ )
- ⇒ Die korrigierte Nachricht ist das Binärmuster der Zahl 14
- ⇒ Ergebnis: 1110

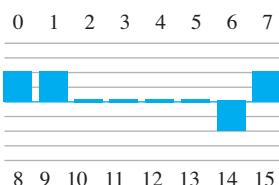
Abb. 6.81: Fehlerkorrektur am Beispiel des Hadamard-Codes achter Ordnung

vier Codewörter, die von der empfangenen Bitsequenz gleich weit entfernt sind. Entsprechend können wir an vier Stellen innerhalb des Histogramms einen Ausschlag mit einer gleich großen Amplitude beobachten. Die vier Positionen entsprechen vier verschiedenen Nachrichten,

$(\mathbf{u} = 1110) \mapsto (\mathbf{v} = 00111100) \mapsto (\mathbf{w} = 10111110)$  (Fehler an der ersten und der vorletzten Position)

$$\mathbf{w} \cdot \mathbf{H}^T = (1 \ -1 \ 1 \ 1 \ 1 \ 1 \ 1 \ -1) \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix}$$

$$= (4 \ 4 \ 0 \ 0 \ 0 \ -4 \ 4)$$



⇒ Es gibt vier gleich wahrscheinliche Fehlerszenarien  
⇒ Die Korrektur ist nicht möglich

**Abb. 6.82:** In diesem Beispiel sind zwei Übertragungsfehler aufgetreten. Der von uns benutzte Hadamard-Code kann die Übertragungsfehler nur noch erkennen, aber nicht mehr korrigieren.

die mit der gleichen Wahrscheinlichkeit gesendet wurden. Eine Korrektur des Übertragungsfehlers ist in diesem Fall nicht möglich.

Nachdem wir uns mit der Konstruktion von Hadamard-Codes vertraut gemacht haben, wollen wir einen Schritt zurücktreten und überlegen, in welcher Beziehung diese Codes zu den bisher betrachteten stehen. Bereits an mehreren Stellen haben wir die Erfahrung gemacht, dass viele Codes *linear* sind, die Codewörter also so gewählt sind, dass sie einen Untervektorraum bilden. Dies wirft die Frage auf, ob die Hadamard-Codes ebenfalls zu den linearen Codes gehören.

Eine einfache Überlegung zeigt, dass wir diese Frage verneinen müssen, wenn wir für die Codekonstruktion beliebige Hadamard-Matrizen zugelassen hätten. Da die Anzahl der Elemente eines Untervektorraums über  $\mathbb{Z}_2$  immer eine Zweierpotenz ist, kann eine Hadamard-Matrix, deren Ordnung nicht in der Form  $2^k$  angegeben werden kann, niemals einen linearen Code erzeugen. Anders stellt sich die Situation für die Hadamard-Matrizen dar, deren Ordnung, wie oben gefordert, eine Zweierpotenz ist. Diese Matrizen erzeugen tatsächlich lineare Codes, d. h., die Codewörter lassen sich mithilfe von Generatormatrizen erzeugen. Abbildung 6.83 zeigt, wie diese Matrizen für die Hadamard-Codes aussehen, die von den Matrizen  $H_2$ ,  $H_4$ ,  $H_8$  erzeugt werden.

$H_2$		$H_4$		$H_8$	
$u$	$c(u)$	$u$	$c(u)$	$u$	$c(u)$
00	00	000	0000	0000	00000000
01	01	001	0011	0001	00001111
10	11	010	0101	0010	00110011
11	10	011	0110	0011	00111100
		100	1111	0100	01010101
		101	1100	0101	01011010
		110	1010	0110	01100110
		111	1001	0111	01101001
				1000	11111111
				1001	11110000
				1010	11001100
				1011	11000011
				1100	10101010
				1101	10100101
				1110	10011001
				1111	10010110

Abb. 6.83: Hadamard-Codes, deren Ordnung eine Zweierpotenz ist, sind linear. Dargestellt sind die Generatormatrizen der Hadamard-Codes, die von den Matrizen  $H_2$ ,  $H_4$  und  $H_8$  erzeugt werden.

Vergleichen wir die Tabellen aus Abbildung 6.83 mit den Tabellen aus Abbildung 6.80, so fällt auf, dass dort zwar die gleichen Codes, d. h. die gleichen Mengen an Codewörtern, abgebildet sind, die Zuordnung zwischen den Nachrichten und den Codewörtern aber eine andere ist. Lassen Sie sich davon nicht irritieren! Immer dann, wenn wir von einem Code sprechen, spielt die konkrete Zuordnung keine Rolle. Formal ist ein Code eine Menge von Codewörtern und macht keine Aussage darüber, wie die konkrete Zuordnung zwischen den Nachrichten und den Codewörtern aussieht. Dagegen tragen *Codierungen* diese Information sehr wohl in sich. Mathematisch gesehen sind sie Abbildungen von der Menge der Nachrichten in die Menge der Codewörter.

An dieser Stelle können wir das weiter oben angesprochene Geheimnis um den Code  $H_8$  lüften und seine wahre Identität offenlegen. Was sich hinter diesem Code verbirgt, macht eine einfache Umformung deutlich. Indem wir in der Generatormatrix  $H_8$  die dritte mit der siebten Spalte vertauschen und danach die erste Zeile auf die dritte und die vierte Zeile addieren, erhalten wir die folgende Matrix:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Addieren wir jetzt die zweite auf die dritte Zeile und vertauschen beide, so ergibt sich diese Matrix:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Addieren wir jetzt noch die unteren drei Zeilen auf die erste, so entsteht eine Matrix, die wir bereits kennen:

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Diese Matrix ist die Generatormatrix des erweiterten Hamming-Codes. Mit anderen Worten: Der Hadamard-Code  $H_8$  und der erweiterte Hamming-Code sind äquivalent. Dass beide dennoch unterschiedliche Codewortmengen aufweisen, liegt daran, dass wir im ersten Umformungsschritt eine Vertauschung der Spalten vorgenommen haben und sich hierdurch die Positionen der Bits innerhalb der Codewörter ändern.

## Mariner 9

Ein prominenter Code wird durch die Matrix  $H_{32}$  erzeugt (Abbildung 6.84). Er wurde von der Raumsonde Mariner 9 verwendet, die am 30. Mai 1971 an Bord einer Atlas-Trägerrakete ihre mehrmonatige Reise zum Mars antrat (Abbildung 6.85).

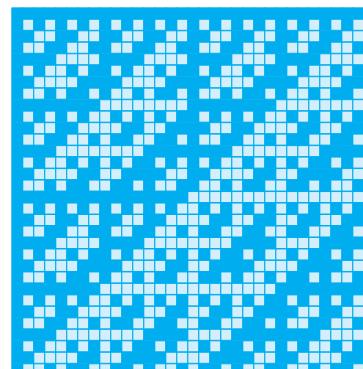
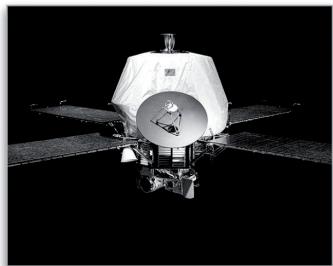


Abb. 6.84: Hadamard-Matrix  $H_{32}$



Raumsonde Mariner 9

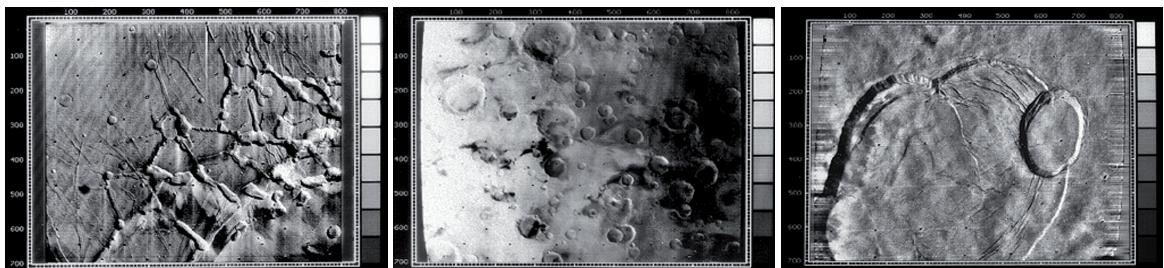


**Abb. 6.85:** Am 30. Mai 1971 wurde die Raumsonde Mariner 9 von der Cape Canaveral Air Force Station in Florida gestartet.

Am 14. November 1971 schwenkte Mariner 9 in den Mars-Orbit ein und war nach der fehlgeschlagenen Mariner-8-Mission die erste Raumsonde der Geschichte, der ein solches Manöver gelang. Mariner 9 sollte aus dem Orbit die Marslandschaft kartografieren, doch die anfängliche Euphorie schlug rasch in Ernüchterung um. Auf dem Mars tobte ein massiver Sandsturm, der keinerlei Bildaufnahmen ermöglichte. Erst im Januar 1972 hatten sich die Staubwolken so weit aufgelöst, dass die Sicht auf die Marsoberfläche frei wurde. Was Mariner 9 in den folgenden Monaten an die Erde übermittelte, waren faszinierende Bilder, die den Mars in einer vorher nicht bekannten Auflösung darstellten (Abbildung 6.86).

Am 27. Oktober 1972 wurde Mariner 9 abgeschaltet, als die Treibstoffvorräte aufgebraucht waren. Zurzeit befindet sich die Sonde immer noch in einer Umlaufbahn um den Mars und umrundet als künstlicher Trabant den roten Planeten. Aufgrund der kontinuierlichen Reibungsverluste wird Mariner 9 voraussichtlich im Jahr 2022 den Orbit verlassen und auf der Marsoberfläche zerschellen.

Übertragen wurden die Marsaufnahmen als Grauwertbilder, wobei für jedes Pixel 64 Helligkeitsstufen unterschieden wurden. Zur Codierung eines Pixels wurde der Hadamard-Code verwendet, der aus der Matrix  $H_{32}$  entsteht (Abbildung 6.87). Er besteht aus 64 Codewörtern der Länge 32, d. h., für jedes Pixel wurden von der Mariner-Sonde 32 Bits zur Erde gesendet. Das geringe Verhältnis von 6 Nachrichtenbits zu 32 Codewortbits wurde bewusst gewählt, um eine gute Fehlerkorrektur-eigenschaft zu erzielen. Da sich alle Zeilenvektoren der Codematrix in mindestens 16 Bitpositionen unterscheiden, konnten in jedem Datenpaket 7 Übertragungsfehler korrigiert werden.



**Abb. 6.86:** Im Jahr 1972 nahm die Raumsonde Mariner 9 Bilder von der Marsoberfläche in einer damals unerreichten Schärfe auf. Für die Codierung der Bilder wurde der Hadamard-Code eingesetzt, der durch die Matrix  $H_{32}$  erzeugt wird.

$u$	$c(u)$	$u$	$c(u)$		
0	000000	11111111111111111111111111111111	32	100000	00000000000000000000000000000000
1	000001	1010101010101010101010101010101010	33	100001	01010101010101010101010101010101
2	000010	11001100110011001100110011001100	34	100010	00110011001100110011001100110011
3	000011	10011001100110011001100110011001	35	100011	01100110011001100110011001100110
4	000100	11110000111100001111000011110000	36	100100	0000111000011110000111100001111
5	000101	10100101101001011010010110100101	37	100101	01011010010110100101101001011010
6	000110	11000011110000111100001111000011	38	100110	00111100001111000011110000111100
7	000111	10010110100101101001011010010110	39	100111	01101001011010010110100101101001
8	001000	11111111000000001111111100000000	40	101000	00000000111111110000000011111111
9	001001	10101010010101011010100101010101	41	101001	01010101101010101010101011010101
10	001010	11001100001100111100110000110011	42	101010	00110011110011000011001111001100
11	001011	10011001011001101001100101100110	43	101011	01100110100110010110011010011001
12	001100	11110000000011111111000000001111	44	101100	000011111110000000111111110000
13	001101	10100101011010101010101011010	45	101101	0101101010100101010110101010101
14	001110	11000011001111001100001100111100	46	101110	00111100110000110011110011000011
15	001111	10010110011010011001011001101001	47	101111	01101001100101100110100110010110
16	010000	11111111111111000000000000000000	48	110000	00000000000000001111111111111111
17	010001	101010101010100101010101010101	49	110001	01010101010101011010101010101010
18	010010	11001100110011000011001100110011	50	110010	00110011001100111100110011001100
19	010011	1001100110011001100110011001100110	51	110011	01100110011001101001100110011001
20	010100	11110000111100000000111100001111	52	110100	00001111000011111111000011110000
21	010101	101001011010010101101001011010	53	110101	01011010010110101010010110100101
22	010110	11000011110000110011110000111100	54	110110	00111100001111001100001111000011
23	010111	10010110100101100110100101101001	55	110111	01101001011010011001011010010110
24	011000	1111111100000000000000000011111111	56	111000	00000000111111111111111100000000
25	011001	10101010010101010101010101010101	57	111001	0101010110101010101010101010101
26	011010	11001100001100110011001111001100	58	111010	00110011110011001100110000110011
27	011011	10011001011001100110011010011001	59	111011	01100110100110011001100101100110
28	011100	111100000001111000011111110000	60	111100	00001111110000111100000001111
29	011101	10100101010100101101010101010101	61	111101	010110101010010110100101010101010
30	011110	11000011001111000011110011000011	62	111110	00111100110000111100001100111100
31	011111	10010110011010010110100110010110	63	111111	01101001100101101001011001011001

Abb. 6.87: Hadamard-Code der Mars-Raumsonde Mariner 9

### 6.4.7 Simplex-Codes

Hadamard-Codes weisen eine enge Verwandtschaft mit einer Codeklasse auf, der wir bereits auf Seite 193 begegnet sind: die Klasse der *Simplex-Codes*. Im Raum  $\mathbb{R}^n$  wird unter einem Simplex ein geometrisches Objekt verstanden, das  $n + 1$  Eckpunkte aufweist, die paarweise gleich weit voneinander entfernt sind. In  $\mathbb{R}$  ist ein Simplex eine Strecke, in  $\mathbb{R}^2$  ein gleichseitiges Dreieck und in  $\mathbb{R}^3$  ein regelmäßiger Tetraeder. Die Simplex-Codes verfolgen die gleiche Idee:



#### Definition 6.16 (Simplex-Code)

Ein linearer Code  $C \subseteq \mathbb{Z}_2^n$  heißt *Simplex-Code*, wenn er  $n + 1$  Elemente umfasst und alle Codewörter paarweise die gleiche Distanz aufweisen.

In Abschnitt 3.4.3 haben wir bereits eine wichtige Eigenschaft von Simplex-Codes angedeutet, die uns gleichsam einen Weg eröffnet, sie systematisch zu erzeugen: Jeder Simplex-Code ist zu einem Hamming-Code orthogonal und kann daher durch eine entsprechende Transformation aus der Generatormatrix des Hamming-Codes erzeugt werden. Abbildung 6.88 demonstriert dies an drei konkreten Beispielen. Als Ergebnis erhalten wir die in Abbildung 6.89 gezeigten Simplex-Codes, die jeweils eine andere Länge aufweisen.

Der Ausgangspunkt für die Transformation ist die Generatormatrix eines Hamming-Codes in systematischer Form. Wie groß die Matrix gewählt werden muss, hängt von der Länge des Simplex-Codes ab. Setzt sich die Nachricht  $u$  aus  $k$  Bits zusammen, so gibt es  $2^k$  Codewörter, und das bedeutet nach Definition 6.16, dass der Simplex-Code eine Untergruppe von  $\mathbb{Z}_2^{2^k-1}$  ist. Folglich bestehen die Codewörter aus  $2^k - 1$  Bits, was unmittelbar die Längen der erzeugten Beispielcodes erklärt. Für Nachrichten der Längen 2, 3 und 4 erhalten wir Simplex-Codes der Längen  $2^2 - 1 = 3$ ,  $2^3 - 1 = 7$  bzw.  $2^4 - 1 = 15$ . Damit die benötigten Größen entstehen, muss die Generatormatrix des Hamming-Codes so gewählt werden, dass sie  $2^k - 1$  Spalten und  $2^k - 1 - k$  Zeilen umfasst. Dies führt dazu, dass die Generatormatrix des Orthogonalraums  $k$  Zeilen und  $2^k - 1$  Spalten aufweist und damit genau den Code erzeugt, den wir suchen.

Eine zweite Möglichkeit, um zu den Simplex-Codes zu gelangen, haben wir uns mit den Untersuchungen in Abschnitt 6.4.6 eröffnet. Dort haben wir die Hadamard-Matrizen kennengelernt, die von Hause aus

[3,2]-Simplex-Code	[7,3]-Simplex-Code	[15,4]-Simplex-Code
$(1 \ 1 \ 1)$ 	$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$ 	$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$ 
Übergang in den Orthogonalraum	Übergang in den Orthogonalraum	Übergang in den Orthogonalraum
$\begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$

**Abb. 6.88:** Die Generatormatrizen des Simplex-Codes lassen sich aus den Generatormatrizen des systematischen Hamming-Codes durch den Übergang in den Orthogonalraum erzeugen.

eine ganz ähnliche Struktur aufweisen wie ein Simplex. Vergleichen wir zwei Zeilenvektoren einer Hadamard-Matrix der Ordnung  $n$  miteinander, so unterscheiden sich diese an genau  $\frac{n}{2}$  Positionen. Die Zeilen einer Hadamard-Matrix weisen also paarweise die gleiche Distanz auf, genau wie die Eckpunkte eines Simplex. Erzeugen wir die Hadamard-Matrix mit der Konstruktionsvorschrift von Sylvester, so ist ferner sichergestellt, dass jede Zeile der Matrix mit einer 1 beginnt. Entfernen wir dieses Bit, so können wir eine Hadamard-Matrix der Ordnung  $n$  in eine Menge von  $n$  Codewörtern der Länge  $n - 1$  überführen. Diese Codewörter weisen immer noch paarweise die gleiche Distanz zueinander auf, sodass dieser Code genau das ist, wonach wir suchen: ein Simplex-Code.

Die Generatormatrizen dieser Simplex-Codes lassen sich sofort aus den Generatormatrizen der Hadamard-Codes erzeugen. Wie die Beispiele in Abbildung 6.90 zeigen, erhalten wir sie ganz einfach dadurch, indem wir aus der Generatormatrix eines Hadamard-Codes die erste Zeile und die linke Spalte streichen. Beachten Sie, dass wir auf diese Weise

[3,2]-Simplex-Code		[7,3]-Simplex-Code		[15,4]-Simplex-Code	
$u$	$c(u)$	$u$	$c(u)$	$u$	$c(u)$
00	000	000	0000000	0000	0000000000000000
01	101	001	1101001	0001	00001111110001
10	110	010	1011010	0010	011100011110010
11	011	011	0110011	0011	011111100000011
		100	0111100	0100	101101100110100
		101	1010101	0101	101110011000101
		110	1100110	0110	110001111000110
		111	0001111	0111	110010000110111
				1000	110110101011000
				1001	110101010101001
				1010	101010110101010
				1011	101001001011011
				1100	011011001101100
				1101	011000110011101
				1110	000111010011110
				1111	0001001011011111

Abb. 6.89: Simplex-Codes, erzeugt aus den Generatormatrizen des Hamming-Codes

Simplex-Codes erzeugen, die zwar die gleiche Größe aufweisen wie jene, die wir in Abbildung 6.88 aus den Hamming-Codes gewonnen haben, aber nicht die gleichen Codewörter umfassen; die Codes sind also nicht identisch. Ein gezielter Blick auf die Generatormatrizen macht jedoch schnell klar, dass sie sich lediglich in der Anordnung der Spalten unterscheiden und damit im Sinne von Definition 3.9 äquivalent sind.

[3,2]-Simplex-Code	[7,3]-Simplex-Code	[15,4]-Simplex-Code																																																														
$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$ <p>1. Zeile und 1. Spalte streichen</p>  $\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$ <p></p> <table border="1"> <thead> <tr> <th><math>u</math></th><th><math>c(u)</math></th></tr> </thead> <tbody> <tr> <td>00</td><td>000</td></tr> <tr> <td>01</td><td>101</td></tr> <tr> <td>10</td><td>011</td></tr> <tr> <td>11</td><td>110</td></tr> </tbody> </table>	$u$	$c(u)$	00	000	01	101	10	011	11	110	$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$ <p>1. Zeile und 1. Spalte streichen</p>  $\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$ <p></p> <table border="1"> <thead> <tr> <th><math>u</math></th><th><math>c(u)</math></th></tr> </thead> <tbody> <tr> <td>000</td><td>0000000</td></tr> <tr> <td>001</td><td>1010101</td></tr> <tr> <td>010</td><td>0110011</td></tr> <tr> <td>011</td><td>1100110</td></tr> <tr> <td>100</td><td>0001111</td></tr> <tr> <td>101</td><td>1011010</td></tr> <tr> <td>110</td><td>0111100</td></tr> <tr> <td>111</td><td>1101001</td></tr> </tbody> </table>	$u$	$c(u)$	000	0000000	001	1010101	010	0110011	011	1100110	100	0001111	101	1011010	110	0111100	111	1101001	$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$ <p>1. Zeile und 1. Spalte streichen</p>  $\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$ <p></p> <table border="1"> <thead> <tr> <th><math>u</math></th><th><math>c(u)</math></th></tr> </thead> <tbody> <tr> <td>0000</td><td>0000000000000000</td></tr> <tr> <td>0001</td><td>101010101010101</td></tr> <tr> <td>0010</td><td>011001100110011</td></tr> <tr> <td>0011</td><td>110011001100110</td></tr> <tr> <td>0100</td><td>000111100001111</td></tr> <tr> <td>0101</td><td>101101001011010</td></tr> <tr> <td>0110</td><td>011110000111100</td></tr> <tr> <td>0111</td><td>110100101101001</td></tr> <tr> <td>1000</td><td>000000011111111</td></tr> <tr> <td>1001</td><td>101010110101010</td></tr> <tr> <td>1010</td><td>011001111001100</td></tr> <tr> <td>1011</td><td>110011010011001</td></tr> <tr> <td>1100</td><td>000111111110000</td></tr> <tr> <td>1101</td><td>101101010100101</td></tr> <tr> <td>1110</td><td>011110011000011</td></tr> <tr> <td>1111</td><td>110100110010110</td></tr> </tbody> </table>	$u$	$c(u)$	0000	0000000000000000	0001	101010101010101	0010	011001100110011	0011	110011001100110	0100	000111100001111	0101	101101001011010	0110	011110000111100	0111	110100101101001	1000	000000011111111	1001	101010110101010	1010	011001111001100	1011	110011010011001	1100	000111111110000	1101	101101010100101	1110	011110011000011	1111	110100110010110
$u$	$c(u)$																																																															
00	000																																																															
01	101																																																															
10	011																																																															
11	110																																																															
$u$	$c(u)$																																																															
000	0000000																																																															
001	1010101																																																															
010	0110011																																																															
011	1100110																																																															
100	0001111																																																															
101	1011010																																																															
110	0111100																																																															
111	1101001																																																															
$u$	$c(u)$																																																															
0000	0000000000000000																																																															
0001	101010101010101																																																															
0010	011001100110011																																																															
0011	110011001100110																																																															
0100	000111100001111																																																															
0101	101101001011010																																																															
0110	011110000111100																																																															
0111	110100101101001																																																															
1000	000000011111111																																																															
1001	101010110101010																																																															
1010	011001111001100																																																															
1011	110011010011001																																																															
1100	000111111110000																																																															
1101	101101010100101																																																															
1110	011110011000011																																																															
1111	110100110010110																																																															

Abb. 6.90: Simplex-Codes, erzeugt aus den Hadamard-Matrizen  $H_4$ ,  $H_8$  und  $H_{16}$

$$\begin{aligned}
 v_0 + v_1 &= u_1 \\
 v_2 + v_3 &= u_1 \\
 v_4 + v_5 &= u_1 \\
 v_6 + v_7 &= u_1 \\
 v_0 + v_2 &= u_2 \\
 v_1 + v_3 &= u_2 \\
 v_4 + v_6 &= u_2 \\
 v_5 + v_7 &= u_2 \\
 v_0 + v_4 &= u_3 \\
 v_1 + v_5 &= u_3 \\
 v_2 + v_6 &= u_3 \\
 v_3 + v_7 &= u_3
 \end{aligned}$$

**Abb. 6.91:** Reed-Muller-Codierungen basieren auf einem arithmetischen Zusammenhang zwischen den Nachrichtenbits  $u_i$  und den Codewortbits  $v_j$ , der eine besonders einfache Fehlerkorrektur erlaubt.

## 6.4.8 Reed-Muller-Codes

In den folgenden Betrachtungen steht  $c$  für eine Codierung, die eine Nachricht

$$\mathbf{u} = u_0 u_1 u_2 u_3 \quad (u_i \in \mathbb{Z}_2)$$

der Länge 4 auf ein Codewort

$$\mathbf{v} = c(\mathbf{u}) = v_0 v_1 v_2 v_3 v_4 v_5 v_6 v_7 \quad (v_i \in \mathbb{Z}_2)$$

der Länge 8 abbildet. Ferner nehmen wir an, dass die Codierung so gewählt wurde, dass die 12 in Abbildung 6.91 zusammengefassten Gleichungen erfüllt sind. Jede Gleichung lässt sich einer von drei Gruppen zuordnen. Bei den Gleichungen der ersten Gruppe steht das Nachrichtenbit  $u_1$  rechts des Gleichheitszeichens, bei den Gleichungen der zweiten Gruppe das Nachrichtenbit  $u_2$  und bei den Gleichungen der dritten Gruppe das Nachrichtenbit  $u_3$ .

Wird ein Codewort fehlerfrei übertragen, so lassen sich drei der vier Nachrichtenbits mithilfe der angegebenen Gleichungen auf einfache Weise decodieren. Hierfür muss der Empfänger lediglich die empfangenen Codewortbits in die Gleichungen einsetzen und die linken Seiten ausrechnen. Als Ergebnis erhält er Auskunft über die Bits  $u_1$ ,  $u_2$  und  $u_3$  (Abbildung 6.92 oben).

Das noch unbekannte Nachrichtenbit  $u_0$  wollen wir für den Moment ignorieren und uns stattdessen mit den Auswirkungen von Übertragungsfehlern beschäftigen. Hierzu nehmen wir an, dass eines der Codewortbits auf der Übertragungsstrecke verfälscht wurde. Ein Blick auf die aufgestellten Gleichungen zeigt, dass jedes Codewortbit in jeder Gleichungsgruppe genau einmal vorkommt, und das bedeutet, dass genau eine Gleichung den falschen Wert annehmen wird. Folgerichtig lässt sich die ursprüngliche Nachricht immer noch rekonstruieren, indem alle Gleichungen ausgewertet werden und in jeder Gruppe mit einem *Mehrheitsentscheider* derjenige Bitwert bestimmt wird, der auf der rechten Seite am häufigsten vorkommt (Abbildung 6.92 Mitte).

Werden auf der Übertragungsstrecke zwei Bits verfälscht, so existiert mindestens eine Gleichungsgruppe, in der zwei der vier durchgeföhrten Additionen als Ergebnis eine 0 hervorbringen und die anderen beiden eine 1. In diesem Fall kann zwar immer noch darauf geschlossen werden, dass Übertragungsfehler aufgetreten sind, das ursprünglich gesendete Nachrichtenbit lässt sich aber nicht mehr eindeutig rekonstruieren.

Die Fehlerkorrektur mithilfe eines Mehrheitsentscheidens durchzuführen, ist verlockend, und so stellt sich zwangsläufig die Frage, wie eine

$1011 \mapsto 11000011$  (fehlerfreie Übertragung)

$$v_0 + v_1 = 1 + 1 = 0$$

$$v_2 + v_3 = 0 + 0 = 0$$

$$v_4 + v_5 = 0 + 0 = 0$$

$$v_6 + v_7 = 1 + 1 = 0$$

$$v_0 + v_2 = 1 + 0 = 1$$

$$v_1 + v_3 = 1 + 0 = 1$$

$$v_4 + v_6 = 0 + 1 = 1$$

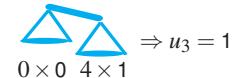
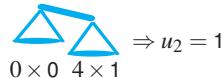
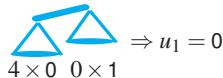
$$v_5 + v_7 = 0 + 1 = 1$$

$$v_0 + v_4 = 1 + 0 = 1$$

$$v_1 + v_5 = 1 + 0 = 1$$

$$v_2 + v_6 = 0 + 1 = 1$$

$$v_3 + v_7 = 0 + 1 = 1$$



$1011 \mapsto 11001011$  (1 Bitfehler)

$$v_0 + v_1 = 1 + 1 = 0$$

$$v_2 + v_3 = 0 + 0 = 0$$

$$v_4 + v_5 = 1 + 0 = 1$$

$$v_6 + v_7 = 1 + 1 = 0$$

$$v_0 + v_2 = 1 + 0 = 1$$

$$v_1 + v_3 = 1 + 0 = 1$$

$$v_4 + v_6 = 1 + 1 = 0$$

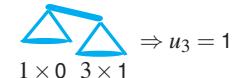
$$v_5 + v_7 = 0 + 1 = 1$$

$$v_0 + v_4 = 1 + 1 = 0$$

$$v_1 + v_5 = 1 + 0 = 1$$

$$v_2 + v_6 = 0 + 1 = 1$$

$$v_3 + v_7 = 0 + 1 = 1$$



$1011 \mapsto 11001010$  (2 Bitfehler)

$$v_0 + v_1 = 1 + 1 = 0$$

$$v_2 + v_3 = 0 + 0 = 0$$

$$v_4 + v_5 = 1 + 0 = 1$$

$$v_6 + v_7 = 1 + 0 = 1$$

$$v_0 + v_2 = 1 + 0 = 1$$

$$v_1 + v_3 = 1 + 0 = 1$$

$$v_4 + v_6 = 1 + 1 = 0$$

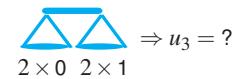
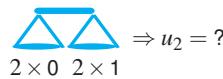
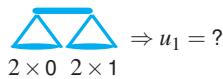
$$v_5 + v_7 = 0 + 0 = 0$$

$$v_0 + v_4 = 1 + 1 = 0$$

$$v_1 + v_5 = 1 + 0 = 1$$

$$v_2 + v_6 = 0 + 1 = 1$$

$$v_3 + v_7 = 0 + 0 = 0$$



**Abb. 6.92:** Ein einfacher Mehrheitsentscheider reicht aus, um Reed-Muller-Codes zu decodieren.

entsprechende Codierung konstruiert werden kann. Tatsächlich müssen wir gar nicht lange suchen; wir finden die Lösung in der uns wohlbekannten Klasse der linearen Codes.

Die Codierung, die wir im Sinn haben, wird durch die folgende Generatormatrix erzeugt:

$$G := \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Dass diese Codierung tatsächlich die gewünschten Eigenschaften erfüllt, wird offensichtlich, wenn wir die Berechnung der Codewortbits so aufschreiben, wie es in Abbildung 6.93 (unten) gezeigt ist.

Die angegebenen Gleichungen machen gleichsam deutlich, wie wir das führende Nachrichtenbit  $u_0$  zurückerhalten können. Sobald  $u_1$ ,  $u_2$  und  $u_3$  ausgerechnet sind, müssen wir deren Werte lediglich einsetzen und erhalten dann 8 Gleichungen, in denen  $u_0$  als einzige Unbekannte verbleibt. Aus diesen Gleichungen können wir  $u_0$  mit demselben Mehrheitsentscheider gewinnen, den wir für die Bestimmung der anderen Nachrichtenbits verwendet haben. Abbildung 6.94 zeigt die Berechnung von  $u_0$  für die drei Übertragungsszenarien, die uns in Abbildung 6.92 als Beispiele dienten.

#### 6.4.8.1 Reed-Muller-Codes erster Ordnung

Den Code, den wir als Beispiel verwendet haben, ist ein *Reed-Muller-Code* erster Ordnung.

$1011 \mapsto 11000011$	$1011 \mapsto 11001011$	$1011 \mapsto 11001010$
$u_1 = 0, u_2 = 1, u_3 = 1$ $\begin{array}{cccccccccc} u_0 & u_0 \\ + 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ + 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ + 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline = 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{array}$  $\begin{array}{cccccccccc} u_0 & u_0 \\ + 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ + 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ + 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline = 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}$  $\begin{array}{cccccccccc} u_0 & u_0 \\ + 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ + 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ + 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline = 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}$ 	$u_1 = 0, u_2 = 1, u_3 = 1$ $\begin{array}{cccccccccc} u_0 & u_0 \\ + 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ + 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ + 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline = 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{array}$  $\begin{array}{cccccccccc} u_0 & u_0 \\ + 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ + 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ + 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline = 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}$  $\begin{array}{cccccccccc} u_0 & u_0 \\ + 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ + 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ + 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline = 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{array}$ 	$u_1 = 0, u_2 = 1, u_3 = 1$ $\begin{array}{cccccccccc} u_0 & u_0 \\ + 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ + 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ + 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline = 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{array}$  $\begin{array}{cccccccccc} u_0 & u_0 \\ + 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ + 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ + 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline = 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{array}$ 
$0 \times 0 \quad 8 \times 1$  $\Rightarrow u_0 = 1$	$1 \times 0 \quad 7 \times 1$  $\Rightarrow u_0 = 1$	$2 \times 0 \quad 6 \times 1$  $\Rightarrow u_0 = 1$

Abb. 6.94: Sind  $u_1$ ,  $u_2$  und  $u_3$  bekannt, dann lässt sich auch  $u_0$  mithilfe eines Mehrheitsentscheiders zurückgewinnen.

RM(1,1)		RM(1,2)		RM(1,3)	
$u$	$c(u)$	$u$	$c(u)$	$u$	$c(u)$
00	00	000	0000	0000	00000000
01	01	001	0011	0001	00001111
10	11	010	0101	0010	00110011
11	10	011	0110	0011	00111100
		100	1111	0100	01010101
		101	1100	0101	01011010
		110	1010	0110	01100110
		111	1001	0111	01101001
				1000	11111111
				1001	11110000
				1010	11001100
				1011	11000011
				1100	10101010
				1101	10100101
				1110	10011001
				1111	10010110

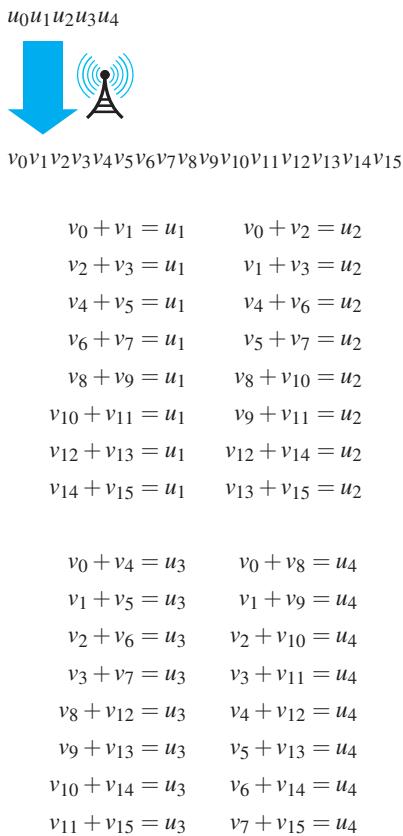
Abb. 6.95: Drei Reed-Muller-Codierungen erster Ordnung

Allgemein wird für Reed-Muller-Codes die Notation

$$\text{RM}(r, m)$$

verwendet, wobei  $r$  die Ordnung bezeichnet und zwischen der Codewortlänge  $n$  und der Zahl  $m$  die Beziehung  $n = 2^m$  besteht. Damit können wir auch unserem gewählten Beispiel einen Namen geben. Es ist der Reed-Muller-Code RM(1,3).

Die Reed-Muller-Codes RM(1,2) und RM(1,1) entstehen, indem die in Abbildung 6.95 dargestellten Bereiche aus der Generatormatrix von RM(1,3) herausgeschnitten werden.



**Abb. 6.96:** Erfüllt ein Code die dargestellten Gleichungen, so lassen sich bis zu drei Übertragungsfehler mithilfe eines Mehrheitsentscheiders korrigieren.

Ein gezielter Blick auf den Aufbau der Generatormatrizen macht deutlich, wie sich Reed-Muller-Codes mit längeren Codewörtern erzeugen lassen. Die Generatormatrix für RM(1,4) sieht beispielsweise so aus:

$$\left( \begin{array}{cccccccccccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array} \right)$$

An dieser Matrix lässt sich problemlos erkennen, wie die Einsen und Nullen innerhalb einer Reed-Muller-Generatormatrix verteilt sind. Die erste Zeile besteht ausnahmslos aus Einsen und in der zweiten Zeile wechseln sich Nullen und Einsen kontinuierlich ab. In den darunter liegenden Zeilen wird ebenfalls zwischen Nullen und Einsen gewechselt, allerdings mit einer jeweils halbierten Frequenz.

Dass die Generatormatrix 5 Zeilen und 16 Spalten aufweist, bedeutet, dass jeweils 5 Nachrichtenbits auf ein 16 Bit breites Codewort abgebildet werden. Für den Code RM(1,3) konnten wir Gleichungen angeben, die einen direkten Bezug zwischen den Codewortbits und den Nachrichtenbits herstellen, und das Gleiche gelingt auch für RM(1,4). Insgesamt erhalten wir 4 Gruppen mit jeweils 8 Gleichungen. Abbildung 6.96 zeigt, um welche Gleichungen es sich im Detail handelt.

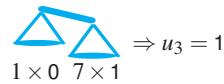
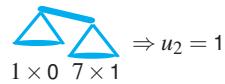
Werden die 16 übertragenen Codewortbits an  $i$  Positionen verfälscht, so werden in jeder Gruppe höchstens  $i$  Gleichungen falsch. Da wir aber jetzt für jedes Bit acht Gleichungen zur Verfügung haben, können bis zu drei Übertragungsfehler korrigiert werden (Abbildung 6.97). Treten vier Übertragungsfehler auf, so gibt es mindestens eine Gleichungsgruppe, in der vier Gleichungen falsch werden. In diesem Fall kann der Fehler zwar noch erkannt werden, das ursprüngliche Nachrichtenbit lässt sich aber nicht mehr eindeutig bestimmen.

Ist Ihnen aufgefallen, dass die Reed-Muller-Codes erster Ordnung eine Codeklasse bilden, die wir bereits kennen? Ein vergleichender Blick auf die Abbildungen 6.83 und 6.95 zeigt, dass wir es hier tatsächlich mit alten Bekannten zu tun haben: Die Reed-Muller-Codes erster Ordnung sind mit den Hadamard-Codes identisch, die wir ausführlich in Abschnitt 6.4.6 besprochen haben.

Die Arbeit, die wir in diesem Abschnitt geleistet haben, war dennoch nicht umsonst. Indem wir die Konstruktionsidee, die wir in diesem Abschnitt entwickelt haben, geringfügig verallgemeinern, können wir Codes konstruieren, die mit den Bordmitteln aus Abschnitt 6.4.6 nicht erreichbar gewesen wären. Diese Verallgemeinerung wird uns

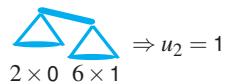
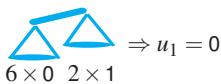
$10110 \mapsto 1100101111000011$  (1 Bitfehler)

$v_0 + v_1 = 1 + 1 = 0$	$v_0 + v_2 = 1 + 0 = 1$	$v_0 + v_4 = 1 + 1 = 0$	$v_0 + v_8 = 1 + 1 = 0$
$v_2 + v_3 = 0 + 0 = 0$	$v_1 + v_3 = 1 + 0 = 1$	$v_1 + v_5 = 1 + 0 = 1$	$v_1 + v_9 = 1 + 1 = 0$
$v_4 + v_5 = 1 + 0 = 1$	$v_4 + v_6 = 1 + 1 = 0$	$v_2 + v_6 = 0 + 1 = 1$	$v_2 + v_{10} = 0 + 0 = 0$
$v_6 + v_7 = 1 + 1 = 0$	$v_5 + v_7 = 0 + 1 = 1$	$v_3 + v_7 = 0 + 1 = 1$	$v_3 + v_{11} = 0 + 0 = 0$
$v_8 + v_9 = 1 + 1 = 0$	$v_8 + v_{10} = 1 + 0 = 1$	$v_8 + v_{12} = 1 + 0 = 1$	$v_4 + v_{12} = 1 + 0 = 1$
$v_{10} + v_{11} = 0 + 0 = 0$	$v_9 + v_{11} = 1 + 0 = 1$	$v_9 + v_{13} = 1 + 0 = 1$	$v_5 + v_{13} = 0 + 0 = 0$
$v_{12} + v_{13} = 0 + 0 = 0$	$v_{12} + v_{14} = 0 + 1 = 1$	$v_{10} + v_{14} = 0 + 1 = 1$	$v_6 + v_{14} = 1 + 1 = 0$
$v_{14} + v_{15} = 1 + 1 = 0$	$v_{13} + v_{15} = 0 + 1 = 1$	$v_{11} + v_{15} = 0 + 1 = 1$	$v_7 + v_{15} = 1 + 1 = 0$



$10110 \mapsto 1100101110100011$  (2 Bitfehler)

$v_0 + v_1 = 1 + 1 = 0$	$v_0 + v_2 = 1 + 0 = 1$	$v_0 + v_4 = 1 + 1 = 0$	$v_0 + v_8 = 1 + 0 = 1$
$v_2 + v_3 = 0 + 0 = 0$	$v_1 + v_3 = 1 + 0 = 1$	$v_1 + v_5 = 1 + 0 = 1$	$v_1 + v_9 = 1 + 1 = 0$
$v_4 + v_5 = 1 + 0 = 1$	$v_4 + v_6 = 1 + 1 = 0$	$v_2 + v_6 = 0 + 1 = 1$	$v_2 + v_{10} = 0 + 0 = 0$
$v_6 + v_7 = 1 + 1 = 0$	$v_5 + v_7 = 0 + 1 = 1$	$v_3 + v_7 = 0 + 1 = 1$	$v_3 + v_{11} = 0 + 0 = 0$
$v_8 + v_9 = 0 + 1 = 1$	$v_8 + v_{10} = 0 + 0 = 0$	$v_8 + v_{12} = 0 + 0 = 0$	$v_4 + v_{12} = 1 + 0 = 1$
$v_{10} + v_{11} = 0 + 0 = 0$	$v_9 + v_{11} = 1 + 0 = 1$	$v_9 + v_{13} = 1 + 0 = 1$	$v_5 + v_{13} = 0 + 0 = 0$
$v_{12} + v_{13} = 0 + 0 = 0$	$v_{12} + v_{14} = 0 + 1 = 1$	$v_{10} + v_{14} = 0 + 1 = 1$	$v_6 + v_{14} = 1 + 1 = 0$
$v_{14} + v_{15} = 1 + 1 = 0$	$v_{13} + v_{15} = 0 + 1 = 1$	$v_{11} + v_{15} = 0 + 1 = 1$	$v_7 + v_{15} = 1 + 1 = 0$



$10110 \mapsto 1100101110100011$  (3 Bitfehler)

$v_0 + v_1 = 1 + 1 = 0$	$v_0 + v_2 = 1 + 0 = 1$	$v_0 + v_4 = 1 + 1 = 0$	$v_0 + v_8 = 1 + 0 = 1$
$v_2 + v_3 = 0 + 0 = 0$	$v_1 + v_3 = 1 + 0 = 1$	$v_1 + v_5 = 1 + 0 = 1$	$v_1 + v_9 = 1 + 1 = 0$
$v_4 + v_5 = 1 + 0 = 1$	$v_4 + v_6 = 1 + 1 = 0$	$v_2 + v_6 = 0 + 1 = 1$	$v_2 + v_{10} = 0 + 0 = 0$
$v_6 + v_7 = 1 + 1 = 0$	$v_5 + v_7 = 0 + 1 = 1$	$v_3 + v_7 = 0 + 1 = 1$	$v_3 + v_{11} = 0 + 0 = 0$
$v_8 + v_9 = 0 + 1 = 1$	$v_8 + v_{10} = 0 + 0 = 0$	$v_8 + v_{12} = 0 + 1 = 1$	$v_4 + v_{12} = 1 + 1 = 0$
$v_{10} + v_{11} = 0 + 0 = 0$	$v_9 + v_{11} = 1 + 0 = 1$	$v_9 + v_{13} = 1 + 0 = 1$	$v_5 + v_{13} = 0 + 0 = 0$
$v_{12} + v_{13} = 1 + 0 = 1$	$v_{12} + v_{14} = 1 + 1 = 0$	$v_{10} + v_{14} = 0 + 1 = 1$	$v_6 + v_{14} = 1 + 1 = 0$
$v_{14} + v_{15} = 1 + 1 = 0$	$v_{13} + v_{15} = 0 + 1 = 1$	$v_{11} + v_{15} = 0 + 1 = 1$	$v_7 + v_{15} = 1 + 1 = 0$

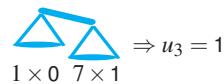
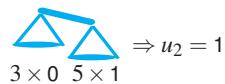
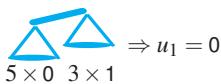


Abb. 6.97: Decodierung von RM(1,4) mithilfe eines Mehrheitsentscheiders

$$\begin{array}{l} r_0 \\ r_1 \\ r_2 \\ r_3 \end{array} \left( \begin{array}{ccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{array} \right)$$

$$\begin{array}{l} r_1 \\ r_2 \\ r_1r_2 \end{array} \left( \begin{array}{ccccccc} 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{array} \right)$$

$$\begin{array}{l} r_1 \\ r_3 \\ r_1r_3 \end{array} \left( \begin{array}{ccccccc} 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{array} \right)$$

$$\begin{array}{l} r_2 \\ r_3 \\ r_2r_3 \end{array} \left( \begin{array}{ccccccc} 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array} \right)$$

$$\begin{array}{l} r_0 \\ r_1 \\ r_2 \\ r_3 \\ r_1r_2 \\ r_1r_3 \\ r_2r_3 \end{array} \left( \begin{array}{ccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array} \right)$$

Systematische Form

$$\left( \begin{array}{ccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{array} \right)$$

**Abb. 6.98:** Durch die Hinzunahme konjunktiv verknüpfter Zeilen entsteht die Generatormatrix des Reed-Muller-Codes RM(2,3).

im nächsten Abschnitt direkt zu den Reed-Muller-Codes höherer Ordnung führen, und damit indirekt zu der Erkenntnis, dass die Hadamard-Codes lediglich ein kleiner Ausschnitt des viel größeren Reed-Muller-Universums sind.

#### 6.4.8.2 Reed-Muller-Codes höherer Ordnung

Die Konstruktion von Reed-Muller-Codes zweiter Ordnung basiert auf der folgenden Idee: Entnehmen wir einer Generatormatrix erster Ordnung zwei Zeilenvektoren und verknüpfen deren Elemente konjunktiv miteinander, so entsteht ein neuer Vektor, der in den meisten Fällen von den ursprünglichen Zeilenvektoren linear unabhängig ist. Verlängern wir die ursprüngliche Matrix um diese Vektoren, so gelangen wir zu einer Generatormatrix zweiter Ordnung. Abbildung 6.98 demonstriert die Konstruktion an einem konkreten Beispiel. Ausgehend von der Generatormatrix von RM(1,3) werden zunächst alle linear unabhängigen Vektoren erzeugt, die sich aus den vier Zeilenvektoren  $r_0, r_1, r_2$  und  $r_3$  bilden lassen. Konkret sind dies die Vektoren  $r_1r_2, r_1r_3$  und  $r_2r_3$ . Wird die Generatormatrix von RM(1,3) jetzt um diese Vektoren erweitert, so entsteht die Generatormatrix des Reed-Muller-Codes RM(2,3).

Die Matrix besitzt 7 Zeilen und 8 Spalten, d.h., es werden 7 Nachrichtenbits auf jeweils 8 Codewortbits abgebildet. In Abbildung 6.98 ist bereits angedeutet, dass sich die Matrix durch elementare Zeilenumformungen so umformen lässt, dass die ersten 7 Spalten die Einheitsmatrix bilden und die letzte Spalte ausschließlich Einsen enthält. Welchen Code diese Matrix erzeugt, wissen wir aus Abschnitt 3.4.3. Es ist der gerade Paritätscode der Länge 8.

Reed-Muller-Codes der Ordnung 3 lassen sich auf die gleiche Weise erzeugen. Hierzu wird die Generatormatrix eines Codes der Ordnung 1 nicht nur um alle linear unabhängigen Zeilenvektoren ergänzt, die sich durch die UND-Verknüpfung zweier Zeilen erzeugen lassen, sondern auch um die Zeilen, die sich aus der UND-Verknüpfung dreier Zeilen ergeben. Auf die gleiche Weise entstehen Reed-Muller-Codes der Ordnung 4, 5 und so fort.

Als Beispiel ist in Abbildung 6.99 die Reed-Muller-Matrix der Ordnung 4 zu sehen, die aus der Generatormatrix von RM(1,5) entstanden ist. Die Zeilen der Matrix teilen sich in 5 Gruppen auf. Die erste Gruppe wird durch die Zeile gebildet, die ausschließlich Einsen enthält, und die zweite Gruppe umfasst die Zeilen, in denen sich die Elemente 0 und 1 in unterschiedlicher Frequenz abwechseln. Die dritte, vierte und

$r_0$	(1 1)
$r_1$	(1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0)
$r_2$	(1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0)
$r_3$	(1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0)
$r_4$	(1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0)
$r_5$	(1 0 0 0)
$r_5r_4$	(1 1 1 1 1 1 1 1 0)
$r_5r_3$	(1 1 1 1 0 0 0 0 1 1 1 1 0)
$r_5r_2$	(1 1 0 0 1 1 0 0 1 1 0 0 1 1 0)
$r_5r_1$	(1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
$r_4r_3$	(1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
$r_4r_2$	(1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0)
$r_4r_1$	(1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0)
$r_3r_2$	(1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0)
$r_3r_1$	(1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0)
$r_2r_1$	(1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0)
$r_5r_4r_3$	(1 1 1 1 0)
$r_5r_4r_2$	(1 1 0 0 1 1 0)
$r_5r_4r_1$	(1 0 1 0 1 0 1 0)
$r_5r_3r_2$	(1 1 0 0 0 0 0 0 1 1 0)
$r_5r_3r_1$	(1 0 1 0 0 0 0 0 1 0 1 0)
$r_5r_2r_1$	(1 0 0 0 1 0 0 0 1 0 0 0 1 0)
$r_4r_3r_2$	(1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
$r_4r_3r_1$	(1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
$r_4r_2r_1$	(1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
$r_3r_2r_1$	(1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
$r_5r_4r_3r_2$	(1 1 0)
$r_5r_4r_3r_1$	(1 0 1 0)
$r_5r_4r_2r_1$	(1 0 0 0 1 0)
$r_5r_3r_2r_1$	(1 0 0 0 0 0 0 0 1 0)
$r_4r_3r_2r_1$	(1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)

Abb. 6.99: Generatormatrix des Reed-Muller-Codes RM(4,5)

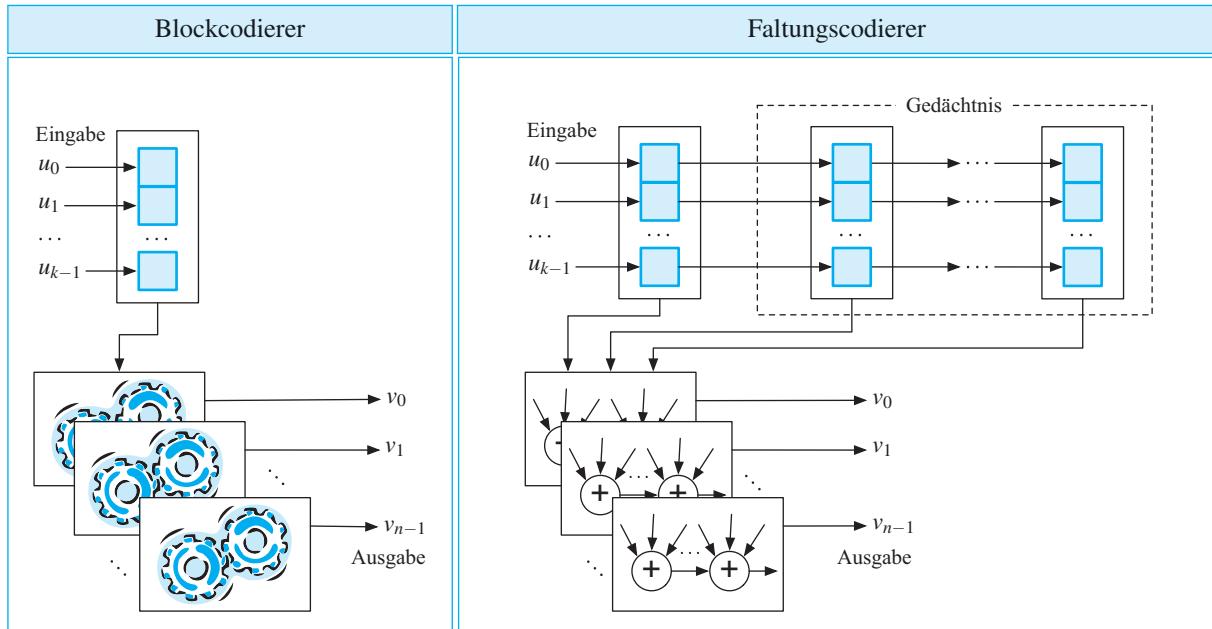
fünfte Gruppe enthalten alle linear unabhängigen Vektoren, die durch die UND-Verknüpfung von zwei, drei bzw. vier Vektoren der zweite Gruppe entstehen.

Insgesamt erzeugt die Matrix einen linearen [32,31,2]-Code, d. h., es werden jeweils 31 Nachrichtenbits so auf 32 Codewortbits abgebildet, dass die Codewörter eine minimale Distanz von 2 aufweisen. Das bedeutet nach Satz 6.3, dass der Code 1 Bitfehler erkennen und 0 Bitfehler korrigieren kann. Tatsächlich ist der entstandene Code wenig spektakulär: Wir haben erneut den Paritätscode vor uns.

Interessante Ergebnisse erhalten wir dann, wenn wir weniger als fünf Gruppen in einer Matrix zusammenfassen. Auf diese Weise entstehen die Generatormatrizen der Codes RM(0,5) bis RM(3,5), die wir uns nacheinander ansehen wollen:

- Die Generatormatrix von RM(0,5) entsteht, indem lediglich die erste Gruppe betrachtet wird. Die Matrix besteht in diesem Fall aus einer einzigen Zeile mit 32 Elementen und erzeugt eine Codierung, die 1 Nachrichtenbit auf 32 Codewortbits erweitert. Da die Matrix ausschließlich aus Einsen besteht, wird das Nachrichtenbit 32-mal repliziert. Damit ist klar, wer hinter dem Reed-Muller-Code RM(0,5) steckt: der Wiederholungscode der Länge 32.
- Fassen wir die ersten beiden Gruppen zusammen, so erreichen wir RM(1,5), ein Reed-Muller-Code erster Ordnung. Wer sich dahinter verbirgt, wissen wir schon: RM(1,5) ist der Hadamard-Code der Ordnung 32.
- Nehmen wir die dritte Gruppe hinzu, so entsteht der Code RM(2,5). Hierbei handelt es sich um einen linearen [32,16,8]-Code, d. h., es werden jeweils 16 Nachrichtenbits so auf 32 Codewortbits abgebildet, dass die Codewörter eine minimale Distanz von 8 aufweisen. Dieser Code ist 7-fehlererkennend und 3-fehlerkorrigierend.
- Die ersten vier Gruppen zusammen bilden die Generatormatrix von RM(3,5). Dieser Code ist ein linearer [32,26,4]-Code und zu dem erweiterten Hamming-Code der Länge 32 äquivalent. Dieser entsteht aus dem [31,26]-Hamming-Code durch das Anhängen eines Paritätsbits. Der [31,26]-Hamming-Code weist eine Distanz von 3 auf, d. h., er kann 2 Bitfehler erkennen und 1 Fehler korrigieren.

Das Anhängen eines Paritätsbits erhöht die Code-Distanz auf 4, so dass RM(3,5) in der Lage ist, 3 Fehler zu erkennen. Die Fehlerkorreureigenschaft bleibt durch die Distanzerweiterung unangetastet. Aus Satz 6.3 wissen wir, dass für die Korrektur von 2 Bitfehlern eine Code-Distanz von mindestens 5 benötigt wird.



**Abb. 6.100:** Einen Faltungscodierer können wir als einen Blockcodierer ansehen, der um ein Gedächtnis erweitert wurde. Die Ausgabebits werden nach einem einfachen Schema gebildet: Sie sind die Modulo-2-Summen gewisser Registerinhalte.

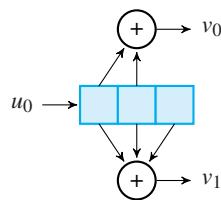
## 6.5 Faltungscodes

In diesem Abschnitt werden wir mit den *Faltungscodes* eine Codeklasse kennenlernen, die sich erheblich von allen bisher betrachteten unterscheidet. Um das Arbeitsprinzip eines Faltungscodierers zu verstehen, erinnern wir uns zunächst an die Funktionsweise eines klassischen Blockcodierers. Eingangsseitig nimmt ein solcher Codierer Blöcke der Länge  $k$  entgegen und bildet diese auf Codewörter der Länge  $n$  ab (Abbildung 6.100 links).

Im Folgenden beschränken wir uns auf Faltungscodierungen mit binären Eingabe- und Ausgabealphabeten; wir gehen also davon aus, dass sowohl auf der Eingabe- als auch auf der Ausgabeseite Bitströme verarbeitet werden.

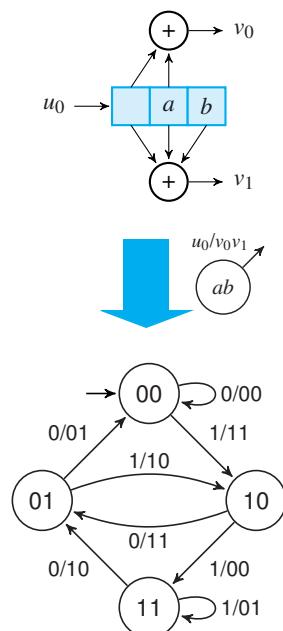
Es ist ein wichtiges Merkmal von Blockcodierern, keine Gedächtnis zu besitzen. Das bedeutet, dass jeder Block immer nach der gleichen Vorschrift verarbeitet wird und die Ausgabe daher nicht durch die Gestalt der vorher verarbeiteten Blöcke beeinflusst werden kann.

(1,2,3)-Faltungscodierer



- (1,2,3) 1 Eingabebit  
 (1,2,3) 2 Ausgabebits  
 (1,2,3) 3 Registerstufen

**Abb. 6.101:** Ein einfacher Faltungscodierer. Die Nachrichtenbits werden von links in das Schieberegister eingespeist und von dort in die beiden Modulo-2-Addierer weitergeleitet.



**Abb. 6.102:** Faltungscodierer lassen sich auf natürliche Weise mithilfe von endlichen Automaten modellieren.

Faltungscodierer weichen von diesem Schema ab. Sie verfügen über ein Gedächtnis, das ihnen erlaubt, für zwei nacheinander eingehende, gleich aussehende Blöcke eine jeweils andere Ausgabe zu erzeugen. Implementiert werden Faltungscodierer mithilfe von Schieberegistern, wie sie in Abbildung 6.100 skizzenhaft eingezeichnet sind. Dort steht jedes farblich hinterlegte Quadrat für ein binäres Speicherelement, das seinen Inhalt jeweils taktsynchron an das nachfolgende Element weiterreicht. Wie die Ausgabebits berechnet werden, ist in Abbildung 6.100 ebenfalls schon angedeutet. Sie entstehen, indem das Schieberegister an mehreren Stellen ausgelesen wird und die einzelnen Bits anschließend addiert werden. Die Addition wird dabei modulo-2 ausgeführt und entspricht auf der logischen Ebene einer XOR-Verknüpfung der Registerbits.

Für die Klassifikation von Faltungscodierungen greifen wir auf die Triplenschreibweise

$$(k, n, l)$$

zurück, wobei den einzelnen Komponenten die folgende Bedeutung zu kommt:

- $k$  ist die Anzahl der parallel entgegengenommenen Eingabebits.
- $n$  ist die Anzahl der parallel produzierten Ausgabebits.
- $l$  ist die Anzahl der Schieberegisterstufen

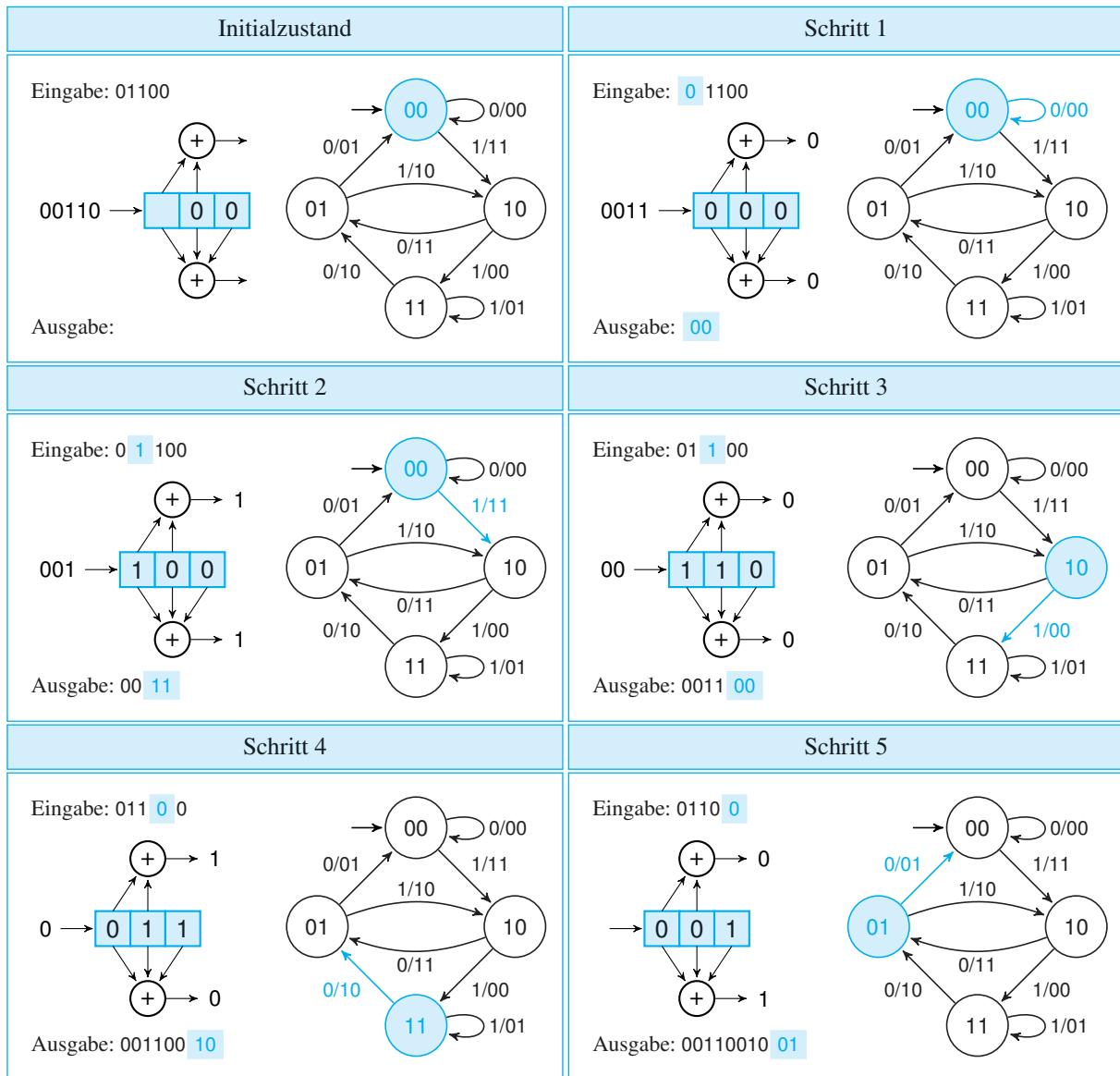
Ein Faltungscodierer bildet damit in jedem Schiebeschritt  $k$  Nachrichtenbits auf  $n$  Codewortbits ab. Der Quotient

$$R = \frac{k}{n}$$

ist die *Coderate* des Faltungscodes: Sie gibt an, wie viele Nachrichtenbits durch ein einzelnes Codewortbit abgedeckt werden, und ist damit gewissermaßen das Netto-Brutto-Verhältnis der Codierung. Sehr kleine Coderaten bedeuten, dass sich der Ausgabestrom stark verlängert.

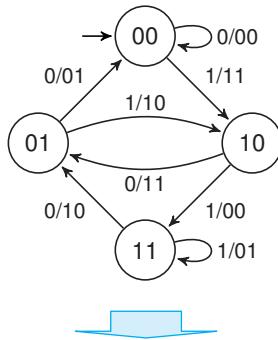
Als Beispiel ist in Abbildung 6.101 ein (1,2,3)-Faltungscodierer dargestellt. Er besteht aus einem einzigen Schieberegister mit 3 Stufen und weist eine Coderate von  $\frac{1}{2}$  auf. Das bedeutet, dass für jedes eingegebene Nachrichtenbit zwei Ausgabebits produziert werden.

Das Verhalten eines Faltungscodierers lässt sich auf verschiedene Weisen beschreiben. Beispielsweise können wir, wie es in Abbildung 6.102 für unseren Beispielcodierer gezeigt ist, aus der Schieberegisterdarstellung einen endlichen Automaten ableiten. Der Automat ist so gestaltet,



**Abb. 6.103:** Codierung der Nachricht 01100 mit dem Schieberegister aus Abbildung 6.101

dass der Gedächtnisteil des Schieberegisters – dies sind alle Registerelemente mit Ausnahme des am weitesten links stehenden – dem aktuell eingenommenen Automatzustand entspricht. Ein Schieberegister mit  $l$  Stufen erzeugt auf diese Weise einen endlichen Automaten mit  $2^{l-1}$



Zuständen. Von jedem Zustand gehen zwei Kanten aus, die in unserem Beispiel mit der Markierung

$$u_0 / v_0 v_1$$

versehen sind. Hierin steht  $u_0$  für das Nachrichtenbit, das von links in das Schieberegister eingespeist wird, und  $v_0$  und  $v_1$  sind die beiden Ausgabebits, die mithilfe der XOR-Logik aus dem aktuellen Registerinhalt berechnet werden.

### Codierung

Abbildung 6.103 zeigt am Beispiel der Nachricht

$$u = 01100,$$

wie die Schieberegisterdarstellung und der daraus konstruierte Automat zusammenhängen. Die Darstellung zeigt auch, dass wir mehrere Möglichkeiten haben, einen Faltungscodierer zu realisieren. Soll die Implementierung in Hardware erfolgen, so können wir die Schieberegisterdarstellung eins zu eins in eine Digitalschaltung übersetzen. Soll die Implementierung dagegen in Software erfolgen, so bietet es sich an, den endlichen Automaten in Form einer Tabelle zu beschreiben, wie sie in Abbildung 6.104 gezeigt ist. Diese Art der Darstellung reduziert die Codierung auf eine Reihe nacheinander ausgeführter Tabellenzugriffe.

Für unsere Beispieldaten erhalten wir in beiden Fällen das folgende Codewort:

$$v = 0011001001$$

Beachten Sie, dass wir eine Nachricht gewählt haben, die mit der Sequenz 00 endet. Dies führt dazu, dass der endliche Automat im nächsten Schritt immer in seinen Initialzustand übergehen wird, unabhängig davon, ob wir als Nächstes eine 0 oder eine 1 in das Register einspeisen. Warum es wichtig ist, dass der Automat im nächsten Schritt einen fest definierten Zustand erreicht, wird in Abschnitt 6.5.1 klar werden. Dort werden wir zeigen, wie sich mit dem Viterbi-Algorithmus eine effiziente Fehlerkorrektur durchführen lässt.

In der Praxis wird das geschilderte Verhalten künstlich herbeigeführt, indem eine Nachricht, die mit einem Schieberegister der Länge  $l$  codiert werden soll, um  $l - 1$  Nullen ergänzt wird. Das bedeutet, dass unsere Beispieldaten in Wirklichkeit nur aus der dreielementigen Bitsequenz 101 bestehen; die beiden letzten Bits sind, bei einem Schieberegister mit 3 Stufen, zwangsläufig 0.

Eingabe	Zustand	Ausgabe	Folgezustand
0	00	00	00
1	00	11	10
0	01	01	00
1	01	10	10
0	10	11	01
1	10	00	11
0	11	10	01
1	11	01	11

Abb. 6.104: Übergangstabelle des endlichen Automaten aus Abbildung 6.102

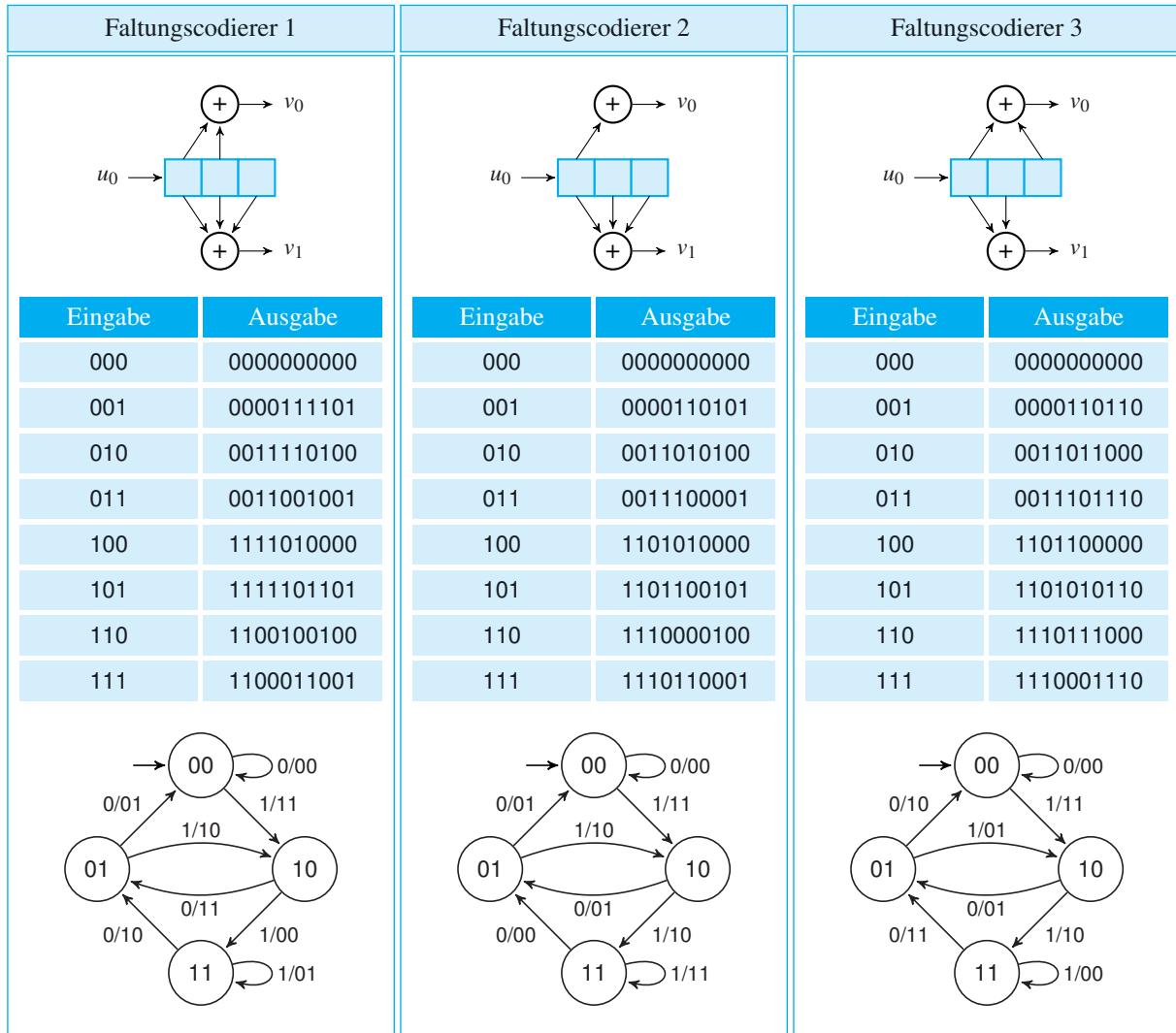
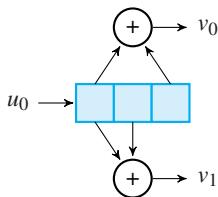


Abb. 6.105: Verschiedene Faltungscodierer im Vergleich

In der linken Spalte in Abbildung 6.105 sehen Sie unseren Beispielcodierer nochmals abgebildet, zusammen mit einer kompletten Liste der Codewörter, die bei der Eingabe von 3 Nachrichtenbits erzeugt werden können. Die beiden Nullen, die zur Terminierung der Faltungscodierung in das Schieberegister eingespeist werden, sind in der Tabelle weggelassen.



Eingabe	Ausgabe
000	0000000000
111	1110001110
0000	000000000000
1111	111000001110
00000	0000000000000000
11111	1110000000001110
000000	00000000000000000000
111111	11100000000000001110

**Abb. 6.106:** Speisen wir Bitsequenzen in den Codierer ein, die ausschließlich das Symbol 0 oder das Symbol 1 beinhalten, so entstehen zwei Codewörter, die sich, unabhängig von der Eingabelänge, immer an genau sechs Positionen unterscheiden.

In der mittleren und der rechten Spalte in Abbildung 6.105 sind zwei weitere Faltungscodierer zu sehen. Sie basieren ebenfalls auf einem dreistufigen Schieberegister und unterscheiden sich lediglich durch die XOR-Logik, mit der die Ausgabebits aus den Registerinhalten berechnet werden. Die beiden zusätzlichen Beispiele sind nicht zufällig gewählt. Sie zeichnen sich durch spezielle Eigenschaften aus, denen wir an dieser Stelle unsere Aufmerksamkeit schenken wollen.

- Der zweite Faltungscodierer ist so aufgebaut, dass die erste XOR-Logik nur ein einziges Register ausliest. Das bedeutet, dass die eingespeiste Nachricht im Klartext in der codierten Ausgabesequenz enthalten ist. In unserem Beispiel finden wir die Nachricht innerhalb des Codeworts an den Positionen 0, 2, 4, ... wieder. Codierungen mit einer solchen Eigenschaft haben wir auf Seite 191 als *systematische Codierung* bezeichnet, und diese Terminologie verwenden wir auch in diesem Kontext.
- Die Codetabelle des dritten Faltungscodierers kommt zunächst unscheinbar daher. Dass dieser Codierer gegenüber den anderen beiden einen gravierenden Nachteil hat, wird erst sichtbar, wenn wir längere Eingaben betrachten. Speisen wir ihn mit kontinuierlich wachsenden Nuller- und Einserketten, so erhalten wir das in Abbildung 6.106 angedeutete Ergebnis.

Egal, wie lange wir die Ketten auch wählen: Die produzierten Codewörter unterscheiden sich immer nur an 6 Bitpositionen. Das bedeutet, dass der Faltungscodierer Nachrichten, die sehr unterschiedlich sind, auf sehr ähnliche Bitsequenzen abbildet. Solche Codierungen heißen *Katastrophencodierungen*, denn die Fehlerkorrektur kann hier tatsächlich in einem sprichwörtlichen Desaster enden. Jetzt reichen nämlich wenige Übertragungsfehler aus, damit beispielsweise aus der gesendeten Nachricht 0 ... 0 auf der Empfängerseite die Nachricht 1 ... 1 wird. Mit anderen Worten: Wenige Bitfehler auf der Übertragungsstrecke führen zu vielen Bitfehlern in der decodierten Nachricht. Es ist eine wichtige Aufgabe bei der Konstruktion von Faltungscodierern, die Entstehung solcher Katastrophencodierungen zu vermeiden.

## Decodierung

Um die ursprüngliche Nachricht aus einem faltungscodierten Datenstrom zu extrahieren, müssen wir deutlich mehr Aufwand betreiben, als

es die Codierung erfordert hat. Im nächsten Abschnitt werden wir für diesen Zweck den *Viterbi-Algorithmus* vorstellen, der auf dem Prinzip der dynamischen Programmierung beruht.

Um den Algorithmus zu verstehen, führen wir zunächst eine weitere Darstellungsform ein, mit der sich das Verhalten eines Faltungscodierers beschreiben lässt. Hierbei handelt es sich um eine Baumdarstellung, wie sie in Abbildung 6.107 zu sehen ist. Wir erhalten diese ganz einfach dadurch, indem wir den endlichen Automaten *ausrollen*. Das bedeutet, dass wir den Automaten, im Startzustand beginnend, eine gewisse Anzahl an Schritten simulieren und dabei die möglichen Berechnungspfade aufzeichnen. Führen wir die Simulation  $n$  Schritte aus, so entsteht ein Baum mit  $2^n$  Pfaden. Jeder Pfad entspricht einer von  $2^n$  möglichen Bitsequenzen, die wir in  $n$  Berechnungsschritten in das Schieberegister einspeisen können. In Abbildung 6.107 zeigen die Blattmarkierungen an, welche Codewörter dabei entstehen.

Beachten Sie, dass wir nicht alle Bitsequenzen als Ausgabe beobachten können, wenn wir uns an das oben Gesagte halten und für ein Schieberegister der Länge  $l$  nur Eingaben zulassen, die mit  $l - 1$  Nullen enden. In unserem Beispiel ist  $l = 3$ , sodass wir alle Nachrichten verwerfen müssen, die nicht mit 00 enden. In der Baumdarstellung sind die Pfade, die diesen ungültigen Eingabesequenzen entsprechen, grau dargestellt.

Die Baumdarstellung enthält ein großes Maß an Redundanz, da der Automat nur über eine endliche Anzahl an Zuständen verfügt. Greifen wir zwei innere Knoten heraus, so wissen wir, dass die daran hängenden Teilbäume identisch sind, wenn sich der Automat beide Male in dem gleichen Zustand befindet. Das bedeutet für unseren Beispielautomaten mit 4 Zuständen, dass es nur vier verschiedene Teilbäume gibt, die sich periodisch wiederholen. Somit können wir den Baum wesentlich kompakter niederschreiben, indem wir diejenigen Knoten miteinander verschmelzen, die dem gleichen Automatenzustand entsprechen. Als Ergebnis erhalten wir das in Abbildung 6.108 dargestellte Diagramm, das als *Trellis-Diagramm* bezeichnet wird.

Trellis-Diagramme haben die Eigenschaft, dass wir jede Nachricht eindeutig mit einem Pfad identifizieren können, der den Eingangsknoten links oben mit dem Ausgangsknoten rechts oben verbindet. Dass alle Pfade in dem gleichen Ausgangsknoten enden, geht auf unsere Forderung zurück, nur solche Nachrichten zuzulassen, die den endlichen Automaten in seinen Ausgangszustand zurückführen.

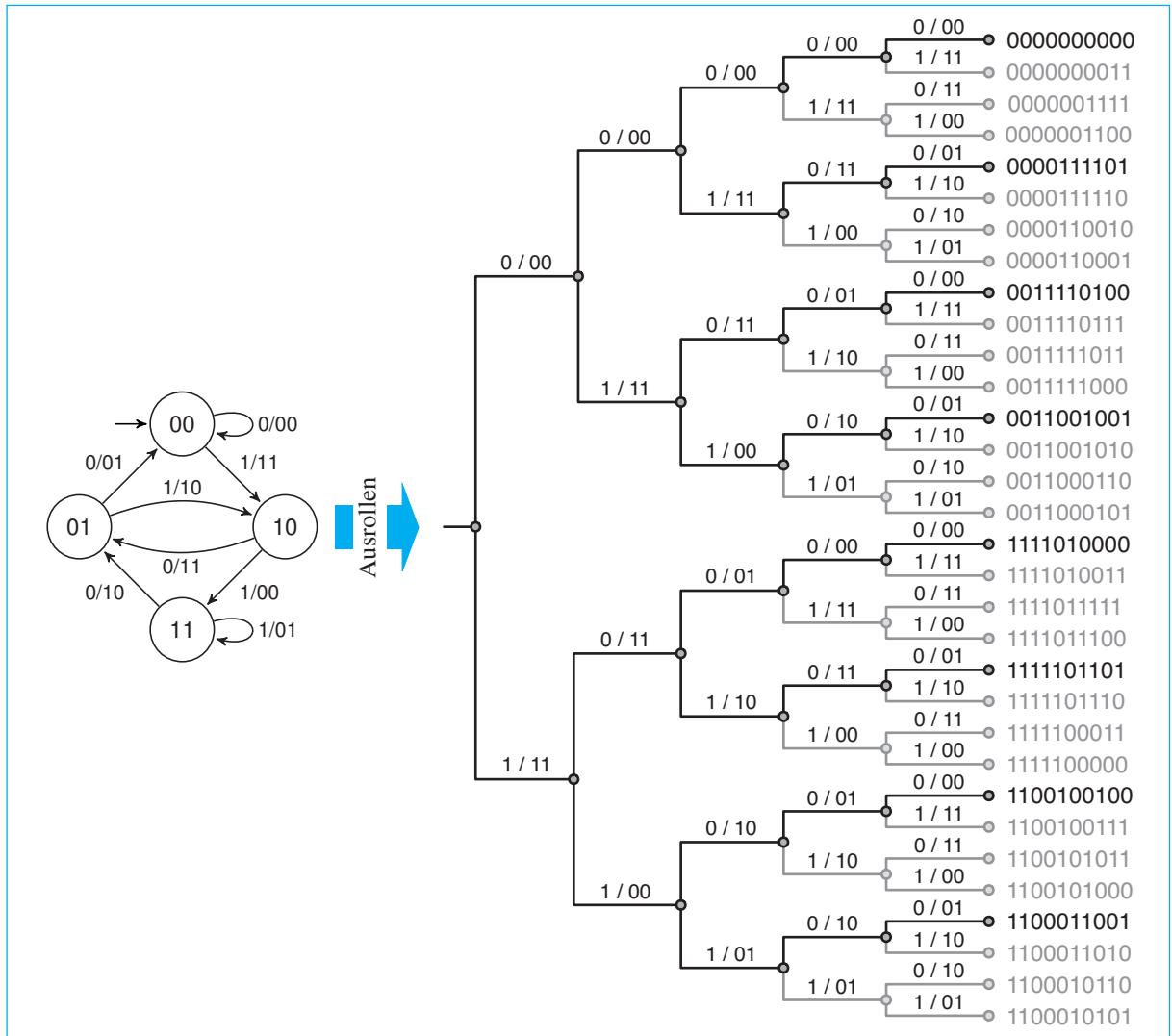
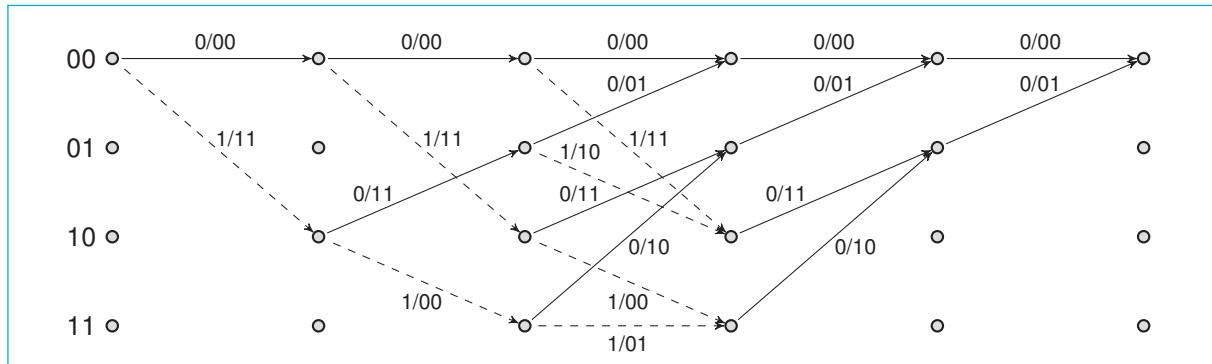


Abb. 6.107: Durch das Ausrollen des zugehörigen endlichen Automaten entsteht die Baumdarstellung eines Faltungscodierers.

### 6.5.1 Viterbi-Algorithmus

Im Jahr 1967 entdeckte der in Italien geborene und später in die USA emigrierte Elektroingenieur Andrew James Viterbi, dass sich das Decodierungsproblem mit einem effizienten Algorithmus lösen lässt, der auf dem Trellis-Diagramm eines Faltungscodierers arbeitet und sich des



**Abb. 6.108:** Durch die Verschmelzung der Knoten, die dem gleichen Automatenzustand entsprechen, entsteht aus der Baumdarstellung des ausgerollten Automaten ein Trellis-Diagramm.

Prinzips der *dynamischen Programmierung* bedient. Anwenden lässt sich die dynamische Programmierung immer dann, wenn sich ein Problem in kleinere Teile zerlegen lässt und die optimale Lösung des Gesamtproblems aus den optimalen Lösungen der Teilprobleme berechnet werden kann. Es ist ein wesentliches Merkmal dieser Technik, dass während der Lösung der Teilprobleme im Allgemeinen noch nicht bekannt ist, welche davon für die Lösung des Gesamtproblems benötigt werden. Die Algorithmen sind deshalb darauf angewiesen, die Lösungen der Teilprobleme in einem Zwischenspeicher abzulegen, aus dem die benötigten Teillösungen dann später ausgelesen werden.

Wir wollen das abstrakt geschilderte Prinzip konkretisieren und nehmen an,  $u$  sei die zu übertragende Nachricht und  $v$  die faltungscodierte Sequenz. Wir wollen unserem oben eingeführten Beispiel treu bleiben und verwenden für  $u$  und  $v$  die folgenden Bitsequenzen:

$$u = 01100$$

$$v = 0011001001$$

Wie gewohnt, bezeichnen wir mit  $w$  die empfangene Nachricht. Wurde die Nachricht korrekt übertragen, ist also  $w = v$ , so können wir  $w$  mit einem eindeutigen Pfad identifizieren, der das Trellis-Diagramm von links nach rechts durchzieht und die Bitfolge  $w$  hervorbringt. In Abbildung 6.109 ist dieser Pfad farblich hervorgehoben.

Für uns interessant ist der Fehlerfall. Wir nehmen deshalb an, dass den Empfänger eine Bitsequenz  $w$  mit  $w \neq v$  erreicht, sich  $v$  und  $w$  also an mindestens einer Position unterscheiden. Wird  $v$  beispielsweise an der

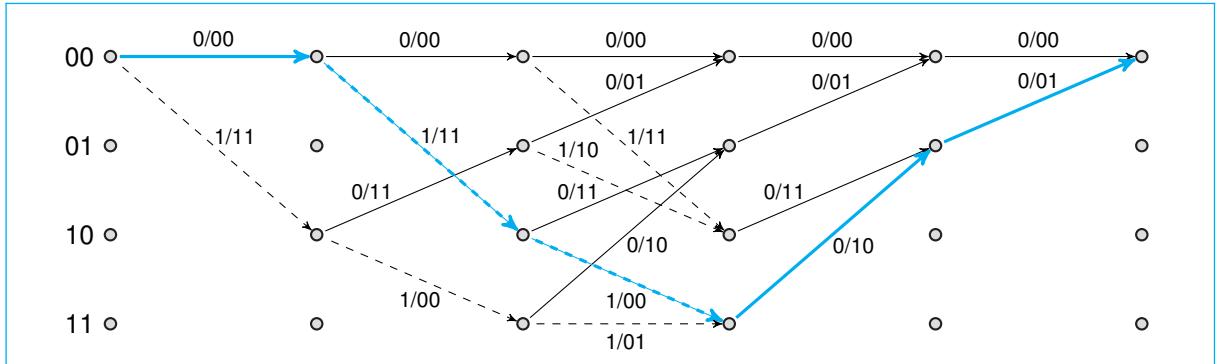


Erliegen Sie nicht der Vermutung, *Trellis-Diagramme* seien nach ihrem geistigen Entdecker benannt. *Trellis* ist der englische Begriff für ein Rankgitter, das an die Hauswand montiert wird und der Begrünung der Fassade dient.

„**trel·lis** (trel'is) **1** a structure of thin strips, esp. of wood, crossing each other in an open pattern of squares, diamonds, etc., on which vines or other creeping plants are trained“

Webster's Dictionary [100]

Der Name ist treffend gewählt. Codieren wir eine Nachricht und verfolgen dabei den Pfad durch das Trellis-Diagramm, so erinnert die optische Struktur tatsächlich an eine rankende Pflanze, die sich von links nach rechts einen Weg durch das Rahmengeflecht bahnt.



**Abb. 6.109:** Jede Nachricht  $u$  und jedes Codewort  $v$  lässt sich eineindeutig einem Pfad zuordnen, der von links oben nach rechts oben durch das Trellis-Diagramm führt. Der farblich hervorgehobene Pfad entspricht der Nachricht 01100 und dem Codewort 0011001001.

sechsten Bitposition verfälscht, so erreicht die Bitsequenz

$$w = 0011011001$$

den Empfänger. Da  $w$  keinem Codewort entspricht, existiert innerhalb des Trellis-Diagramms kein Pfad, dessen Bitfolge exakt mit  $w$  übereinstimmt. In diesem Fall muss der Decoder denjenigen Pfad bestimmen, dessen Bitfolge die geringste Hamming-Distanz zu  $w$  aufweist, und die zu diesem Pfad gehörende Nachricht  $u$  ausgeben. Dies ist exakt das Funktionsprinzip eines *Maximum-Likelihood-Decoders*, dem wir bereits auf Seite 362 begegnet sind.

Aber wie lässt sich der Pfad mit der geringsten Hamming-Distanz zu  $w$  bestimmen? Natürlich könnte der Empfänger sämtliche Pfade der Reihe nach durchkämmen, die Distanz zwischen der Pfadsequenz und  $w$  notieren und am Ende denjenigen Pfad mit der minimalen Distanz auswählen. Da die Anzahl der Pfade exponentiell mit der Länge des Trellis-Diagramms wächst, ist dieses Vorgehen allerdings alles andere als praktikabel.

Glücklicherweise müssen wir gar nicht alle Pfade ansehen; es reicht aus, das Trellis-Diagramm von links nach rechts zu durchkämmen und in jedem Schritt die Punkte einer Spalte mit Zahlen zu markieren.

Wie werden diese Zahlen berechnet? Zunächst halten wir fest, dass es für einen Punkt  $p$  in der Regel mehrere Pfade  $P_1, P_2, \dots$  gibt, die den Anfangspunkt links oben mit  $p$  verbinden, und wir für jeden dieser Pfade die Hamming-Distanz zu dem entsprechenden Anfangsstück

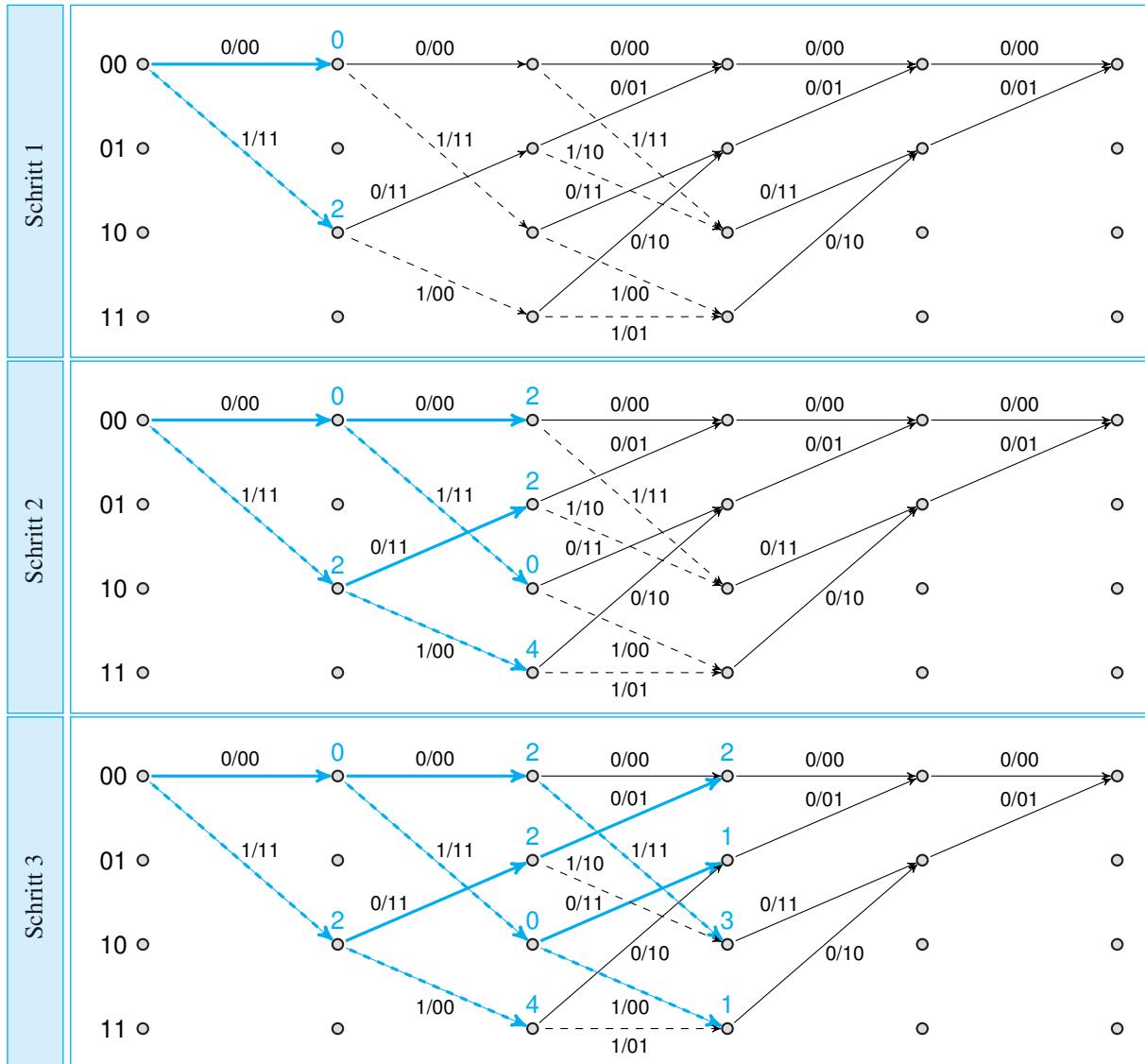


Abb. 6.110: Der Viterbi-Algorithmus in Aktion

von  $w$  berechnen können. Es ist eine der zentralen Ideen des Viterbi-Algorithmus, dass wir gar nicht alle diese Distanzen benötigen, sondern nur das Minimum davon. Genau dieses Minimum ist die Zahl  $\omega$ , mit der wir den Punkt  $p$  markieren.

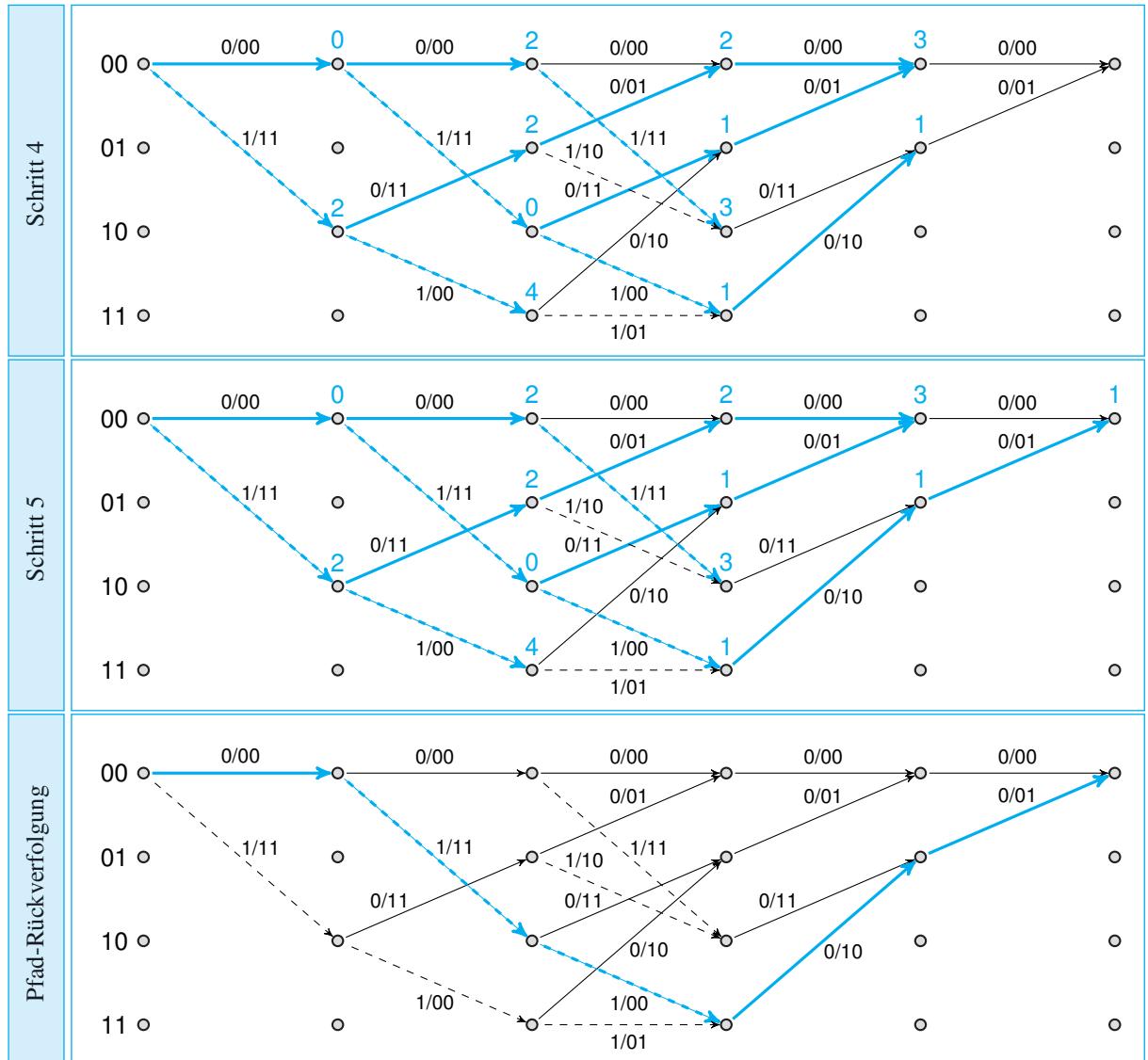


Abb. 6.111: Der Viterbi-Algorithmus in Aktion (Fortsetzung)

Die Abbildungen 6.110 und 6.111 demonstrieren die Berechnung an einem konkreten Beispiel. Wie oben beschrieben, wird das Trellis-Diagramm spaltenweise von links nach rechts durchkämmt, und die einzelnen Punkte mit dem Gewicht  $\omega$  markiert. In den ersten beiden Schritten geht die Berechnung von  $\omega$  sehr einfach von der Hand. Da die

besuchten Punkte nur einen einzigen Vorgängerknoten besitzen, lässt sich  $\omega$  berechnen, indem der Wert des Vorgängerknotens um die Anzahl der Bitpositionen erhöht wird, an denen sich  $w$  von den Bits der jeweiligen Kantenmarkierung unterscheidet.

Im dritten Schritt müssen wir geringfügig mehr Arbeit investieren, da für die Knoten in der bearbeiteten Spalte jeweils zwei Vorgänger existieren. Um den richtigen Ergebniswert zu erhalten, führen wir die Addition zweimal durch, einmal für jede eingehende Kante. Danach markieren wir den Knoten mit der kleinsten Summe, die wir ausgerechnet haben.

Bearbeiten wir das gesamte Trellis-Diagramm auf diese Weise, so ist der Ausgangsknoten mit jenem Wert markiert, den wir suchen. Wir können dort die minimale Hamming-Distanz ablesen, die  $w$  zu einem der Pfade innerhalb des Diagramms aufweist. Ist der Wert gleich 0, so wurde die Nachricht fehlerfrei empfangen. Ist er ungleich 0, so wurde das Codewort auf der Übertragungsstrecke verfälscht.

Natürlich reicht es nicht aus, die minimale Distanz zu kennen. Damit der Empfänger die gesendete Nachricht rekonstruieren kann, muss er den Pfad bestimmen, der die berechnete Distanz aufweist. Glücklicherweise ist dies jetzt sehr einfach möglich. Der Empfänger muss hierfür lediglich vom Ausgangsknoten zum Eingangsknoten zurückgehen und in jedem Schritt das Segment auswählen, das uns während der Berechnung die kleinste Zwischensumme geliefert hat. Das letzte Diagramm in Abbildung 6.111 zeigt, dass wir auf diese Weise genau den Pfad zurückgewinnen, den wir in Abbildung 6.109 bereits angegeben hatten. Die zu diesem Pfad gehörende Nachricht ist

$$\mathbf{u} = 01100,$$

und der Übertragungsfehler ist erfolgreich korrigiert.

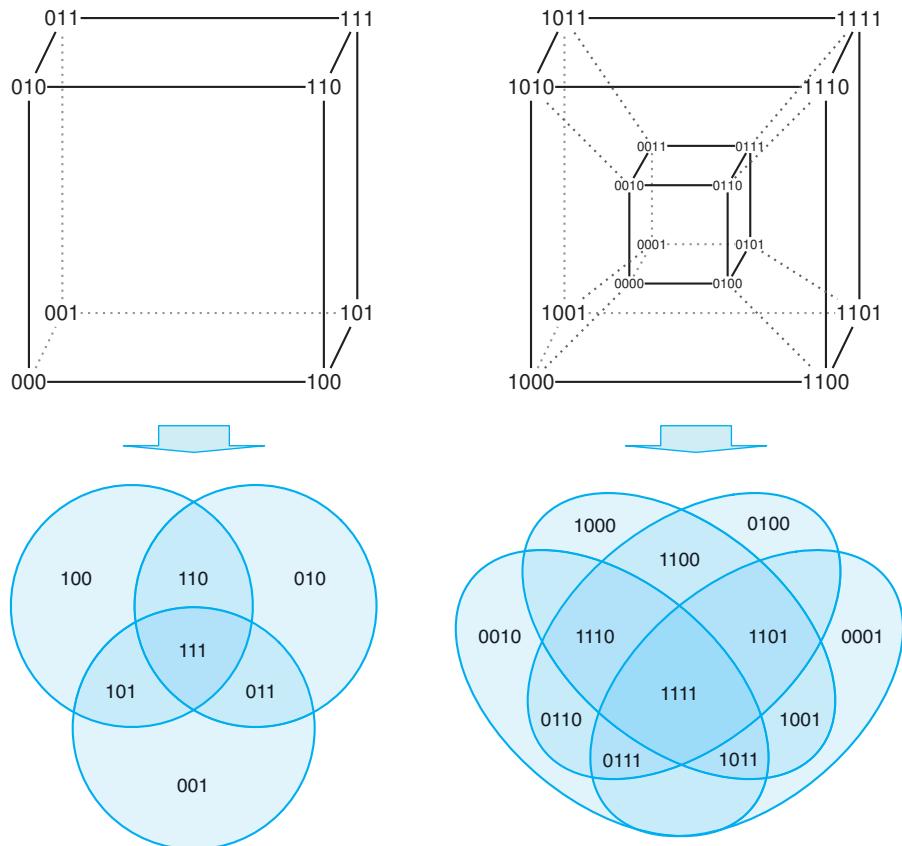
## 6.6 Übungsaufgaben

### Aufgabe 6.1



**Webcode**  
6010

In Abschnitt 6.3 haben wir dargelegt, wie sich Binärcodes mithilfe von *Hamming-Würfeln* visualisieren lassen. Eine Alternative ist die Verwendung von Venn-Diagrammen, wie sie nachstehend für Binärcodes der Längen 3 und 4 dargestellt sind:



Für  $n \geq 5$  ist es schwierig, eine übersichtliche Würfeldarstellung zu konstruieren, eine Darstellung mithilfe von Venn-Diagrammen ist aber immer noch möglich. Versuchen Sie, ein solches Venn-Diagramm durch die Überlagerung von 5 Kopien der gleichen geometrischen Figur herzustellen. Achten Sie darauf, dass die grundlegende Eigenschaft von Venn-Diagrammen erhalten bleibt: Zwei Bitsequenzen, die benachbarten Feldern zugeordnet sind, unterscheiden sich an genau einer Position.

In den nachfolgend abgedruckten Pharmazentralnummern sind mehrere Ziffern verloren gegangen. Berechnen Sie die fehlenden Ziffern oder begründen Sie, warum dies unmöglich ist.



PZN - 070910



PZN - 578675



PZN - 590751

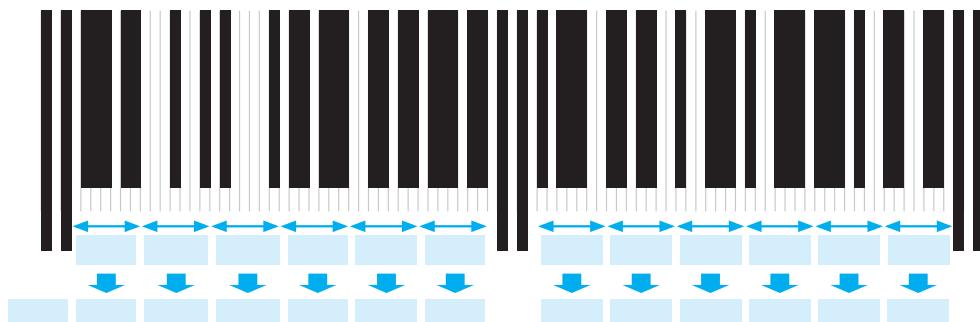
Ihnen liegen die folgenden Kreditkarten vor:



Welche Karten besitzen gültige Nummern und welche sind offensichtliche Fälschungen? Korrigieren Sie die Prüfziffer der gefälschten Karten so, dass gültige Kartenzahlen entstehen.

Übrigens: Die Rechnung zeigt, wie einfach sich gültige Kartenzahlen erzeugen lassen. Ein wirklicher Makel ist dies nicht, da die Prüfziffern gar nicht dafür gedacht sind, um die Echtheit von Kartenzahlen zu überprüfen. Sie dienen lediglich dazu, falsch oder vertauscht eingetippte Kartenzahlen zu erkennen.

Decodieren Sie den folgenden Barcode:



### Aufgabe 6.2



**Webcode**  
6034

### Aufgabe 6.3



**Webcode**  
6055

### Aufgabe 6.4

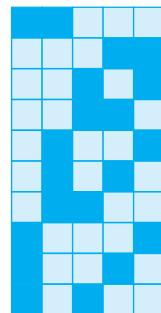
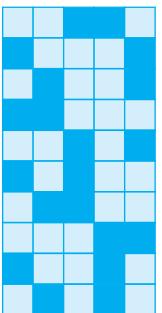
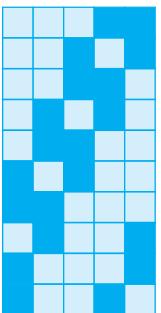
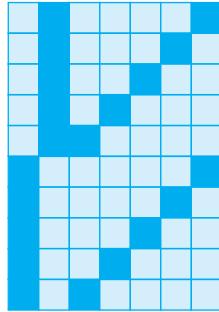
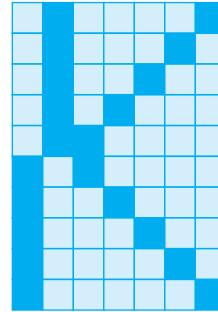
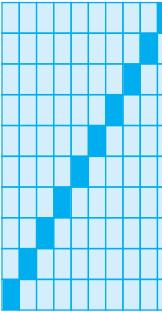


**Webcode**  
6089

**Aufgabe 6.5**

In der Codierungstheorie wird unter einem

 **Webcode**  
**6113**
*n-aus-m*-Codeein Binärkode der Länge  $m$  verstanden, in dessen Codewörtern die Ziffer 1 genau  $n$ -mal vorkommt. Nachstehend sehen Sie eine Auswahl solcher Codes.

74210-Code	Interleaved 2-aus-5-Code	Walking-Code
Kategorie: 2-aus-5  0 11000 1 00011 2 00101 3 00110 4 01001 5 01010 6 01100 7 10001 8 10010 9 10100	Kategorie: 2-aus-5  0 00110 1 10001 2 01001 3 11000 4 00101 5 10100 6 01100 7 00011 8 10010 9 01010	Kategorie: 2-aus-5  0 00011 1 00101 2 00110 3 01010 4 01100 5 10100 6 11000 7 01001 8 10001 9 10010
Biquinär-Code	Reflektierter Biquinär-Code	One-Hot-Code
Kategorie: 2-aus-7  0 0100001 1 0100010 2 0100100 3 0101000 4 0110000 5 1000001 6 1000010 7 1000100 8 1001000 9 1010000	Kategorie: 2-aus-7  0 0100001 1 0100010 2 0100100 3 0101000 4 0110000 5 1010000 6 1001000 7 1000100 8 1000010 9 1000001	Kategorie: 1-aus-10  0 0000000001 1 0000000010 2 0000000100 3 0000001000 4 0000010000 5 0000100000 6 0001000000 7 0010000000 8 0100000000 9 1000000000

- Bestimmen Sie für jeden Code die Code-Distanz.
- Wie viele Fehler können die Codes erkennen bzw. korrigieren?
- Der 74210-Code wird so genannt, weil sich der Zahlenwert  $x$ , der einem Codewort

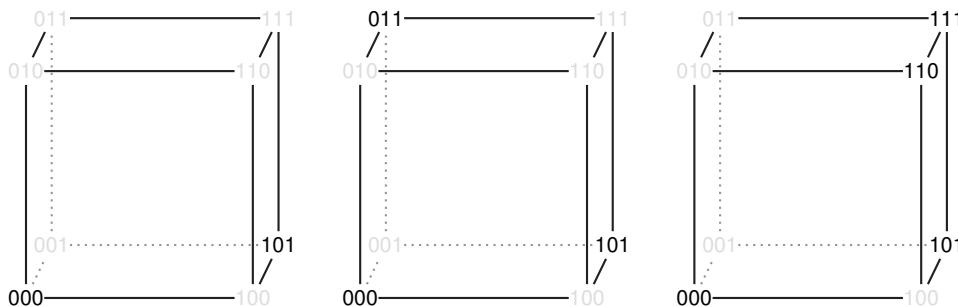
$$\mathbf{v} = v_0 v_1 v_2 v_3 v_4$$

zugeordnet ist, für  $x \neq 0$  mit der folgenden Formel berechnen lässt:

$$x = 7v_0 + 4v_1 + 2v_2 + 1v_3 + 0v_4$$

Für welche der anderen Codes existieren ähnliche Formeln?

Bestimmen Sie für die nachfolgend abgebildeten Codes, wie viele Fehler erkannt bzw. korrigiert werden können:



Der Code ist ...

Der Code ist ...

Der Code ist ...

-fehlererkennend und  
 -fehlerkorrigierend.

-fehlererkennend und  
 -fehlerkorrigierend.

-fehlererkennend und  
 -fehlerkorrigierend.

Für welche Codes lassen sich die Fehlererkennungs- und die Fehlerkorrektur-eigenschaft durch eine Umordnung der Codewörter verbessern?

Es sei  $C$  ein Code. Welche der folgenden Aussagen sind richtig und welche sind falsch?

Aufgabe 6.7



Webcode  
6153

Wahr Falsch

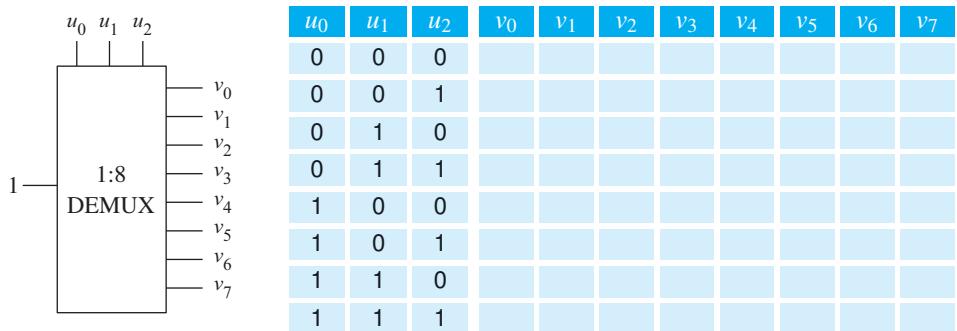
<input checked="" type="checkbox"/> $C$ ist genau dann $r$ -fehlererkennend, wenn $\Delta C \geq r + 1$ ist.	<input type="radio"/>	<input type="radio"/>
<input checked="" type="checkbox"/> $C$ ist genau dann $r$ -fehlerkorrigierend, wenn $\Delta C \geq r + 1$ ist.	<input type="radio"/>	<input type="radio"/>
<input checked="" type="checkbox"/> $C$ ist genau dann $r$ -fehlererkennend, wenn $\Delta C \geq 2r + 1$ ist.	<input type="radio"/>	<input type="radio"/>
<input checked="" type="checkbox"/> $C$ ist genau dann $r$ -fehlerkorrigierend, wenn $\Delta C \geq 2r + 1$ ist.	<input type="radio"/>	<input type="radio"/>

**Aufgabe 6.8**

**Webcode**  
**6183**

Nachstehend ist eine Encoder-Schaltung abgebildet, die eine binäre Nachricht der Form  $u_0u_1u_2$  auf ein binäres Codewort der Form  $v_0v_1\dots v_7$  abbildet. Die Schaltung kommt mit einem einzigen Demultiplexer aus, dessen Eingangsleitung konstant mit 1 beschaltet ist. Die Nachrichtenleitungen sind direkt mit den Steuerleitungen des Demultiplexers verbunden und die Ausgangsleitungen bilden das Codewort.

- a) Füllen Sie die rechts neben der Encoder-Schaltung abgebildete Tabelle aus, und geben Sie an, welcher Ihnen bekannte Code hier erzeugt wird?



- b) Wo und für welchen Zweck haben wir eine Schaltung dieser Art eingesetzt?

**Aufgabe 6.9**

**Webcode**  
**6201**

In der folgenden Tabelle ist eine Wiederholungscodierung angegeben, die jedem Nachrichtenwort der Länge 2 ein Codewort der Länge 6 zuordnet. Das Codewort wird durch die dreifache Wiederholung der Nachricht gebildet:

$u$	$c(u)$
00	000000
01	010101
10	101010
11	111111

- a) Wie groß ist die Code-Distanz?  
 b) Wie groß darf der Radius der Hamming-Kugeln maximal gewählt werden, damit sich die Kugeln weder schneiden noch berühren?

Radius  $r =$

- c) Geben Sie für jede Hamming-Kugel die darin enthaltenen Bitsequenzen an. Füllen Sie hierzu die nachstehende Tabelle aus:

Hamming-Kugel von $c(00)$	Hamming-Kugel von $c(01)$	Hamming-Kugel von $c(10)$	Hamming-Kugel von $c(11)$

- d) Lässt sich jede Bitsequenz eindeutig einer Hamming-Kugel zuordnen? Begründen Sie Ihre Antwort.

In dieser Aufgabe sind drei Kontrollmatrizen und drei Syndromtabellen gegeben. Leider ist die Information verloren gegangen, welche Matrix zu welcher Syndromtabelle passt. Stellen Sie die Zuordnung wieder her.

### Aufgabe 6.10



Webcode  
6213

$$H_1 = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix} \quad H_2 = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \quad H_3 = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$

	$w \cdot H^T$	$e$
0	000	0000000
1	001	1000000
2	010	0100000
3	011	0001000
4	100	0000100
5	101	0001000
6	110	0000001
7	111	0000010

Tabelle 1

	$w \cdot H^T$	$e$
0	000	0000000
1	001	0010000
2	010	0001000
3	011	0000001
4	100	0000100
5	101	0000010
6	110	1000000
7	111	0100000

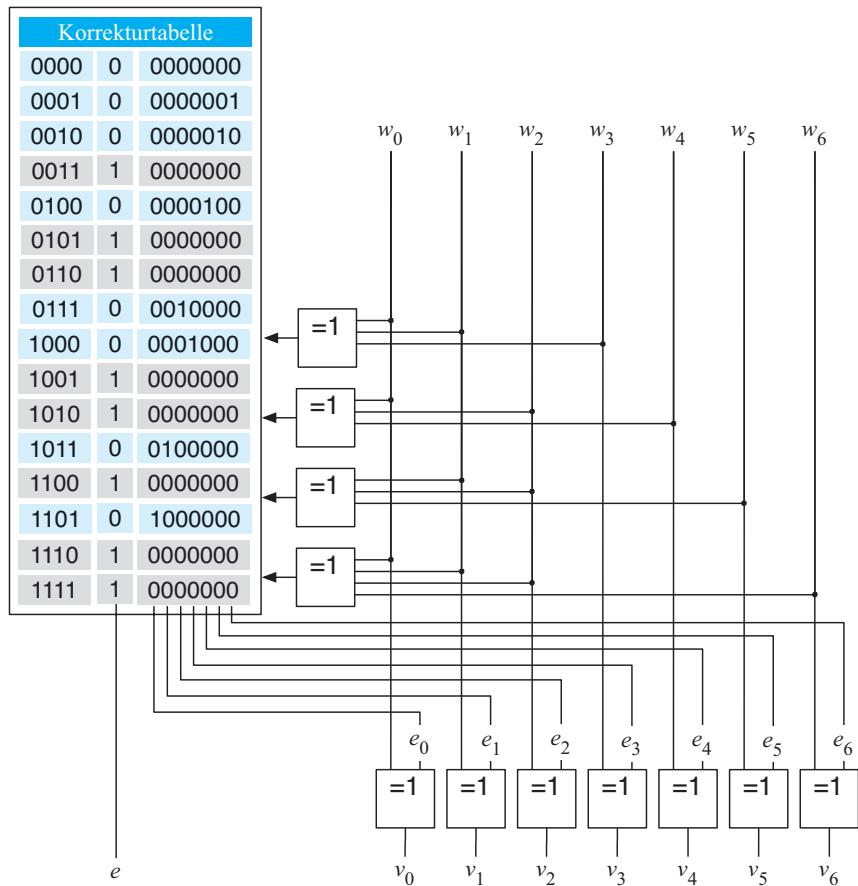
Tabelle 2

	$w \cdot H^T$	$e$
0	000	0000000
1	001	1000000
2	010	0100000
3	011	0010000
4	100	0001000
5	101	0000100
6	110	0000010
7	111	0000001

Tabelle 3

**Aufgabe 6.11**

Gegeben sei der folgende in Hardware implementierte Syndromdecoder:

**Webcode****6250**

- Leiten Sie aus dem Schaltplan die Kontrollmatrix ab.
- Erzeugen Sie die Generatormatrix durch den Übergang in den Orthogonalraum.
- Welcher Ihnen bekannte Code wird hier verwendet?
- Wie groß ist die Code-Distanz?
- Erklären Sie, warum der Fehlervektor **0** in der Korrekturtabelle mehrfach vorkommt. Was ist die Bedeutung der zusätzlichen Ausgangsleitung  $e$ ?

In diesem Kapitel haben Sie mit dem Hamming-Code einen wichtigen Kanalcode kennengelernt.

**Aufgabe 6.12**

**Webcode**
**6275**

- a) Codieren Sie mithilfe der folgenden beiden Tabellen die Nachrichten 00110101101 und 11001010000:

$u$	$2^3$	$2^2$	$2^1$	$2^0$
3				
5				
6				
7				
9				
10				
11				
12				
13				
14				
15				

$\oplus$      $\oplus$      $\oplus$      $\oplus$ 
  
 $\hline$

$u$	$2^3$	$2^2$	$2^1$	$2^0$
3				
5				
6				
7				
9				
10				
11				
12				
13				
14				
15				

$\oplus$      $\oplus$      $\oplus$      $\oplus$ 
  
 $\hline$

- b) Nehmen Sie an, die in a) codierten Nachrichten werden nacheinander übertragen. Anstatt jede Bitsequenz getrennt zu codieren, könnte die zusammengesetzte Nachricht auch als Ganzes codiert und übertragen werden. Erzeugen Sie das Hamming-Codewort für die Nachricht 0011010110111001010000 und nennen Sie die Vor- und Nachteile gegenüber der getrennten Codierung.

Rekonstruieren Sie die Originalnachricht aus den folgenden Hamming-codierten Zeichenketten. Gehen Sie davon aus, dass höchstens 1 Bit bei der Übertragung verfälscht wurde.

- a) 0110010110101
- b) 0001101001011
- c) 0111111100001

**Aufgabe 6.13**

**Webcode**
**6306**

**Aufgabe 6.14****Webcode  
6363**

Hamming-Codes sind 1-fehlerkorrigierende Codes, d. h., alle Übertragungsfehler, die durch ein einzelnes verfälschtes Bit entstanden sind, können erkannt und korrigiert werden.

- Wie viele Bits müssen mindestens kippen, damit ein Hamming-Decoder einen Übertragungsfehler nicht mehr in jedem Fall erkennt?
- Konstruieren Sie ein entsprechendes Beispiel.

**Aufgabe 6.15****Webcode  
6380**

Nachfolgend ist dargestellt, wie ein Empfänger eine Hamming-codierte Nachricht entgegennimmt und einen aufgetretenen Übertragungsfehler korrigiert. Leider sind in der Darstellung einige Einträge verloren gegangen. Stellen Sie diese Einträge wieder her.

<b><i>u</i></b>	<b><math>2^3</math></b>	<b><math>2^2</math></b>	<b><math>2^1</math></b>	<b><math>2^0</math></b>
3				
5	1	1		1
6	1	1	1	
7				
9				
10	0	0	0	
11	1	1	1	1
12	0	0	0	
13	1	1	1	1
14	0	0	0	0
15	0	0	0	0
	$\oplus$	$\oplus$	$\oplus$	$\oplus$
	1	1	0	0
$\oplus$	0	1	1	1
=				
		Fehler an Position 11		

<b><i>u</i></b>	<b><math>2^3</math></b>	<b><math>2^2</math></b>	<b><math>2^1</math></b>	<b><math>2^0</math></b>
3				
5	0	0	0	0
6	1		1	1
7				
9				
10	0	0		0
11	0	0	0	0
12	1	1	1	
13	1	1	1	1
14	0	0	0	0
15	0	0	0	0
	$\oplus$	$\oplus$	$\oplus$	$\oplus$
	0	1	1	1
$\oplus$	0	1	1	1
=				
		Fehler an Position 13		

Ein perfekter Code ist ein Code, dessen Hamming-Kugeln den Coderaum erschöpfend ausfüllen. Das bedeutet, dass sich alle Bitsequenzen eindeutig einer Hamming-Kugel zuordnen lassen.

**Aufgabe 6.16**

**Webcode**  
**6387**

- Geben Sie einen perfekten Code  $C$  mit  $\Delta C = 3$  an, der nicht das Nullwort enthält.
- Ist dieser Code ein Hamming-Code?

- Geben Sie in den nachstehenden Tabellen die Codewörter an, die von den Generatorpolynomen  $x^3 + x^2 + 1$  und  $x^3 + x + 1$  erzeugt werden. Jedes Codewort soll dabei eine Länge von 6 Bits aufweisen.

Codewortpolynom	=	Codewort
$(x^3 + x^2 + 1) \cdot 0$	= 0	000000
$(x^3 + x^2 + 1) \cdot 1$	= $x^3 + x^2 + 1$	001101
$(x^3 + x^2 + 1) \cdot x$	= $x^4 + x^3 + x$	011010
$(x^3 + x^2 + 1) \cdot (x + 1)$	=	
$(x^3 + x^2 + 1) \cdot x^2$	=	
$(x^3 + x^2 + 1) \cdot (x^2 + 1)$	=	
$(x^3 + x^2 + 1) \cdot (x^2 + x)$	=	
$(x^3 + x^2 + 1) \cdot (x^2 + x + 1)$	=	

Codewortpolynom	=	Codewort
$(x^3 + x + 1) \cdot 0$	= 0	000000
$(x^3 + x + 1) \cdot 1$	= $x^3 + x + 1$	001011
$(x^3 + x + 1) \cdot x$	= $x^4 + x^2 + x$	010110
$(x^3 + x + 1) \cdot (x + 1)$	=	
$(x^3 + x + 1) \cdot x^2$	=	
$(x^3 + x + 1) \cdot (x^2 + 1)$	=	
$(x^3 + x + 1) \cdot (x^2 + x)$	=	
$(x^3 + x + 1) \cdot (x^2 + x + 1)$	=	

- Sind die erzeugten Codes zyklisch?

**Aufgabe 6.17**

**Webcode**  
**6403**

**Aufgabe 6.18**

**Webcode  
6446**

Gegeben sei das Generatorpolynom  $g(x) = x^5 + x^4 + x^2 + 1$  sowie die Nachricht 1010001101.

- Codieren Sie die Nachricht mit dem CRC-Algorithmus.
- Verifizieren Sie Ihr Ergebnis, indem Sie das Codewort durch das Generatorpolynom dividieren.
- Simulieren Sie einen Fehlerfall, indem Sie ein einzelnes Bit kippen. Zeigen Sie, dass der eingefügte Fehler durch den CRC-Algorithmus erkannt wird.

**Aufgabe 6.19**

**Webcode  
6465**

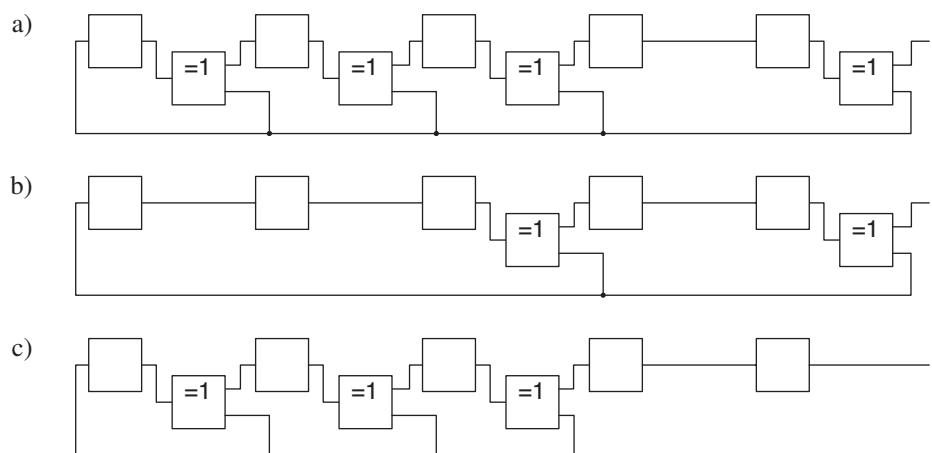
Die Nachricht 10101110 soll mithilfe von CRC-Codewörtern der Bitbreite 7 übertragen werden. Das Generatorpolynom sei  $g(x) = x^3 + 1$ .

- Wie viele Datenpakete werden für die Übertragung der Nachricht benötigt?
- Berechnen Sie die Pakete.

**Aufgabe 6.20**

**Webcode  
6483**

In Abschnitt 6.4.3.2 haben Sie gelernt, dass sich die CRC-Codierung mithilfe von Schieberegistern realisieren lässt, wie sie hier exemplarisch abgebildet sind. Geben Sie für jedes Schaltbild das verwendete Generatorpolynom an.



In der folgenden Tabelle finden Sie mehrere Generatorpolynome vor, die in der Praxis für die Erzeugung von CRC-Codes benutzt werden.

**Aufgabe 6.21**

**Webcode  
6543**

Name	Generatorpolynom
CRC-4 (CCITT)	$x^4 + x + 1$
CRC-5 (USB)	$x^5 + x^2 + 1$
CRC-7 (SD/MMC)	$x^7 + x^3 + 1$
CRC-8 (ITUT)	$x^8 + x^2 + x + 1$
CRC-15 (CAN)	$x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$
CRC-16 (CCITT)	$x^{16} + x^{12} + x^5 + 1$
CRC-32 (IEEE 802.3)	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
CRC-64 (ISO 3309)	$x^{64} + x^4 + x^3 + x + 1$

- a) Finden Sie heraus, welche Generatorpolynome für die nachstehend angegebenen Codewortlängen  $n$  einen zyklischen Code erzeugen, und beschriften Sie die Felder der Tabelle dementsprechend mit *Ja* oder *Nein*.

	$n = 15$	$n = 31$	$n = 63$	$n = 127$
CRC-4				
CRC-5				
CRC-7				
CRC-8				
CRC-15				
CRC-16				
CRC-32				
CRC-64				

- b) Warum fehlen in der Tabelle einige Felder?

**Aufgabe 6.22****Webcode  
6565**

In Abschnitt 6.5 haben wir erarbeitet, dass jedes Generatorpolynom  $g(x)$  mit der Eigenschaft,  $x^n + 1$  ohne Rest zu teilen, einen zyklischen Code der Länge  $n$  erzeugt. Gilt auch die Umkehrung dieses Sachverhalts? Lässt sich jeder zyklische Code der Länge  $n$  durch ein entsprechendes Generatorpolynom erzeugen?

**Aufgabe 6.23****Webcode  
6571**

Das Polynom

$$x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$$

erzeugt den CRC-Code, mit dem z. B. in Kraftfahrzeugen die CAN-Bus-Kommunikation von Steuergeräten abgesichert wird. Wie lautet das Codewort für die Nachricht 111101?

**Aufgabe 6.24****Webcode  
6580**

Reed-Solomon-Codes basieren auf der Idee, eine Nachricht mit einem eindeutig bestimmten Polynom zu identifizieren. Die Stützwerte dieses Polynoms werden übertragen, und zwar so viele davon, dass der Verlust bzw. die Verfälschung von Datenpaketen kompensiert werden kann.

In dieser Aufgabe betrachten wir einen Sender, der eine Folge von Zahlen aus der Menge  $\{0, \dots, 6\}$  überträgt. Der Datenstrom wird dabei in Dreiergruppen der Form  $m_0 m_1 m_2$  unterteilt und jede Gruppe separat codiert. Im ersten Schritt interpoliert der Sender aus den drei Zahlen  $m_0, m_1$  und  $m_2$  ein Polynom  $p(x)$  mit der folgenden Eigenschaft:

$$p(0) = m_0 \quad p(1) = m_1 \quad p(2) = m_2 \quad (6.60)$$

Im zweiten Schritt wird das Polynom an den Stützstellen  $x = 0$  bis  $x = 4$  ausgewertet, und im dritten Schritt werden die Funktionswerte gesendet.

- Es sei  $m_0 = 1, m_1 = 2, m_2 = 2$ . Bestimmen Sie ein Polynom  $p(x) \in \mathbb{Q}[x]$ , das (6.60) erfüllt.
- Da Polynome mit rationalen Koeffizienten in der Praxis umständlich zu handhaben sind, sollen Sie die Rechnung mit dem endlichen Körper  $\mathbb{Z}_7$  wiederholen. Bestimmen Sie hierzu ein Polynom  $p(x) \in \mathbb{Z}_7[x]$ , das (6.60) erfüllt, und berechnen Sie damit die zu übertragenen Werte.
- Nehmen Sie an, den Empfänger erreichen die Datenpakete (1 2 1 1 6). Stellen Sie mit dem Berlekamp-Welch-Algorithmus fest, ob ein Übertragungsfehler aufgetreten ist, und rekonstruieren Sie gegebenenfalls die gesendete Nachricht. Führen Sie Ihre Rechnung unter der Annahme durch, dass maximal 1 Datenpaket bei der Übertragung verfälscht wurde.

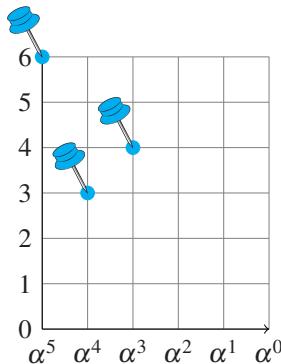
In dieser Aufgabe geht es um die Polynominterpolation in endlichen Körpern. Für unsere Berechnungen verwenden wir den endlichen Körper  $\mathbb{Z}_7$  und das primitive Element  $\alpha = 3$ .

**Aufgabe 6.25**

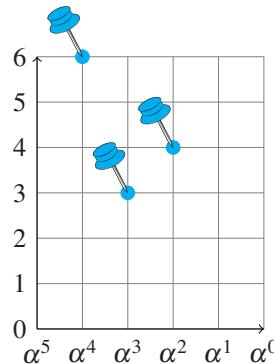
**Webcode**  
**6619**

- a) Interpolieren Sie für die eingezeichneten Stützpunkte jeweils ein Polynom vom Grad 2 und tragen Sie die fehlenden Stützwerte in die Diagramme ein.

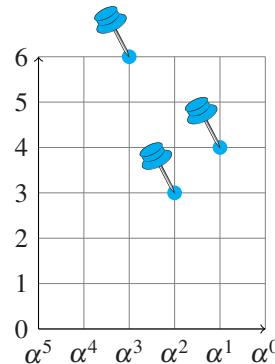
$$p_1(\alpha^5) = 6, p_1(\alpha^4) = 3, p_1(\alpha^3) = 4 \quad p_2(\alpha^4) = 6, p_2(\alpha^3) = 3, p_2(\alpha^2) = 4 \quad p_3(\alpha^3) = 6, p_3(\alpha^2) = 3, p_3(\alpha^1) = 4$$



$$p_1(x) =$$

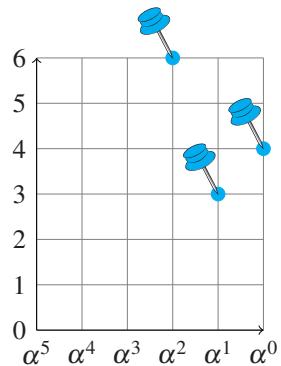


$$p_2(x) =$$

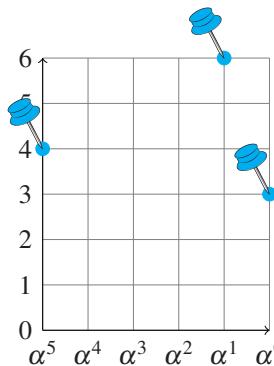


$$p_3(x) =$$

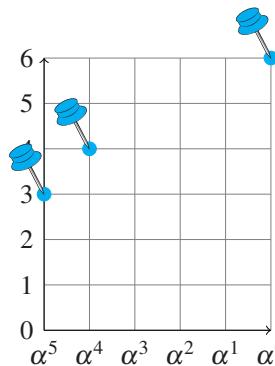
$$p_4(\alpha^2) = 6, p_4(\alpha^1) = 3, p_4(\alpha^0) = 4 \quad p_5(\alpha^5) = 4, p_5(\alpha^1) = 6, p_5(\alpha^0) = 3 \quad p_6(\alpha^5) = 3, p_6(\alpha^4) = 4, p_6(\alpha^0) = 6$$



$$p_4(x) =$$



$$p_5(x) =$$



$$p_6(x) =$$

- b) Vergleichen Sie die Werteverläufe miteinander. Was stellen Sie fest?

- c) Geht die in Teil b) beobachtete Eigenschaft verloren, wenn die  $x$ -Achse um das Element 0 erweitert wird?

**Aufgabe 6.26****Webcode  
6629**

Auf Seite 429 haben wir gezeigt, dass Reed-Solomon-Codes lineare Codes sind. Ist  $\alpha$  ein primitives Element,  $k$  die Nachrichtenlänge und  $n$  die Länge der Codewörter, so hat die Generatormatrix die folgende Gestalt:

$$\begin{pmatrix} (\alpha^{n-1})^{k-1} & \dots & (\alpha^2)^{k-1} & (\alpha)^{k-1} & 1 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ (\alpha^{n-1})^2 & \dots & (\alpha^2)^2 & (\alpha)^2 & 1 \\ (\alpha^{n-1}) & \dots & (\alpha^2) & (\alpha) & 1 \\ 1 & \dots & 1 & 1 & 1 \end{pmatrix}$$

Wir wollen diese Matrix nun folgendermaßen ergänzen:

$$G = \begin{pmatrix} (\alpha^{n-1})^{k-1} & \dots & (\alpha^2)^{k-1} & (\alpha)^{k-1} & 1 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ (\alpha^{n-1})^2 & \dots & (\alpha^2)^2 & (\alpha)^2 & 1 & 0 \\ (\alpha^{n-1}) & \dots & (\alpha^2) & (\alpha) & 1 & 0 \\ 1 & \dots & 1 & 1 & 1 & 1 \end{pmatrix}$$

- a) Was bewirkt die neu hinzugefügte Spalte?
- b) Wie haben sich die Fehlererkennungs- und -korrektureigenschaften verändert?
- c) Ist der entstandene Code selbstdual? Falls ja, unter welchen Bedingungen?

**Aufgabe 6.27****Webcode  
6640**

In dieser Aufgabe geht es darum, die Nachricht  $(3 \ 4 \ 2 \ 1)$  mithilfe eines Reed-Solomon-Codes zu übertragen.

- a) Bestimmen Sie aus der zu sendenden Nachricht das für die Codierung benötigte Polynom  $p(x)$ . Legen Sie für Ihre Berechnung zunächst den vertrauten Polynomring  $\mathbb{Q}[x]$  zugrunde, und wählen Sie  $p(x)$  so, dass eine systematische Codierung entsteht.
- b) Wie viele Werte müssen übertragen werden, damit der entstehende Code 1-fehlerkorrigierend ist? Bestimmen Sie die Datenpakete.
- c) Nehmen Sie an, dass nur die ersten 4 Datenpakete den Empfänger erreichen. Kann die Nachricht trotzdem vollständig rekonstruiert werden? Wenn ja, wie?

- d) Nehmen Sie an, dass nur die letzten 4 Datenpakete den Empfänger erreichen. Kann die Nachricht immer noch verlustfrei rekonstruiert werden? Wenn ja, wie?
- e) Begründen Sie, warum der Polynomring  $\mathbb{Q}[x]$  in der Praxis ungeeignet ist, um eine Nachricht zu codieren.
- f) In diesem Kapitel haben Sie gelernt, dass Polynome über endlichen Körpern die bessere Wahl sind. Wiederholen Sie die getätigten Berechnungen, diesmal jedoch im Polynomring  $\mathbb{Z}_7[x]$ .
- g) Warum ist auch der Polynomring  $\mathbb{Z}_7[x]$  im Allgemeinen keine gute Wahl? Welcher Ring wird in der Praxis stattdessen verwendet?
- h) Die gewählte Zuordnung von Nachrichten zu Polynomen ist nicht die einzige mögliche. Wir können der Nachricht (3 4 2 1) genauso gut das Polynom

$$p(x) = 3x^3 + 4x^2 + 2x + 1$$

zuordnen. Welche Datenpakete müssen übertragen werden, wenn wir dieses Polynom für die Codierung verwenden? Rechnen Sie im Polynomring  $\mathbb{Z}_7[x]$ . Welche Vor- und welche Nachteile bietet diese Art der Codierung?

Nachstehend sehen Sie die Verknüpfungstabellen des Körpers  $\mathbb{Z}_4$ , den wir in dieser Aufgabe für die Reed-Solomon-Codierung einsetzen wollen:

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

·	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

### Aufgabe 6.28



Webcode  
6661

- a) Berechnen Sie ...

$$0^2 = \quad , 1^2 = \quad , 2^2 = \quad , 2 \cdot 0 = \quad , 2 \cdot 1 = \quad , 2 \cdot 3 = \quad$$

- b) Es sei (1 2 2) die zu übertragende Nachricht. Zeigen Sie, dass

$$p(x) = 2x^2 + x + 1$$

das dazu passende Reed-Solomon-Polynom ist, wenn wir eine systematische Codierung anstreben.

- c) Sichern Sie die Datenpakete mit einem zusätzlichen Prüfwert ab. Wie viele Fehler kann der Empfänger erkennen und wie viele kann er korrigieren?

**Aufgabe 6.29****Webcode  
6710**

Wir betrachten einen Reed-Solomon-Codierer, der jeweils zwei gesendete Datenbytes mit einem Prüfbyte absichert.

- Gegeben seien die Datenbytes **0x0F** und **0x15**. Berechnen Sie das Prüfbyte.
- Jetzt betrachten wir einen Reed-Solomon-Code, der zwei gesendete Datenbytes mit zwei Prüfbytes absichert. Wir empfangen die folgende Sequenz:

**0x18 0x1A 0x1C 0x1F**

Ist die Nachricht fehlerfrei? Falls ja, begründen Sie, warum. Falls nein, geben Sie das falsch übertragene Byte an, und korrigieren Sie den Fehler.

**Aufgabe 6.30****Webcode  
6747**

In dieser Aufgabe betrachten wir ein Übertragungsszenario, in dem der Sender 64 Datenpakete an den Empfänger schickt. Für die Decodierung trägt der Empfänger die Pakete in eine quadratische Matrix der Größe  $8 \times 8$  ein. Die Matrix ist so gestaltet, dass die ersten 4 Zeilen sowie alle 8 Spalten ein Codewort von RS(8,4) bilden und der linke obere Quadrant die Nachrichtenpakete enthält. Alle übrigen Datenpakete sind Prüfwerte.

Die folgende Matrix zeigt auf grafische Weise, wie die Nachrichtenpakete und die Prüfwerte verteilt sind. An den mit ‚N‘ markierten Positionen befinden sich die Nachrichtenpakete und an den mit ‚P‘ markierten Positionen die Prüfwerte.

RS(8,4)	N	N	N	N	P	P	P	P	RS(8,4)
RS(8,4)	N	N	N	N	P	P	P	P	RS(8,4)
RS(8,4)	N	N	N	N	P	P	P	P	RS(8,4)
RS(8,4)	N	N	N	N	P	P	P	P	RS(8,4)
	P	P	P	P	P	P	P	P	
	P	P	P	P	P	P	P	P	
	P	P	P	P	P	P	P	P	
	P	P	P	P	P	P	P	P	

- Wir nehmen an, dass der Empfänger eine codierte Nachricht empfängt, in der alle mit einem Kreuz versehenen Datenpakete falsch übertragen wurden:

$\times$	$\times$								RS(8,4)
$\times$	$\times$	$\times$	$\times$						RS(8,4)
$\times$	$\times$	$\times$	$\times$						RS(8,4)
		$\times$	$\times$	$\times$	$\times$	$\times$			RS(8,4)
		$\times$	$\times$	$\times$	$\times$				RS(8,4)
				$\times$	$\times$	$\times$	$\times$		
					$\times$	$\times$			
						$\times$	$\times$		
RS(8,4)	RS(8,4)								

Zeigen Sie, dass der Empfänger die ursprünglich gesendete Nachricht, den vielen Übertragungsfehlern zum Trotz, dennoch rekonstruieren kann.

Wir wollen die Fehlerkorrektur-eigenschaften dieses Codes mit einem gewöhnlichen Reed-Solomon-Code vergleichen. Da in den 64 gesendeten Datenpaketen 16 Nachrichtenpakete enthalten sind, ist RS(64,16) ein geeigneter Kandidat.

- b) Für welche Werte von  $r$  ist der Code RS(64,16)  $r$ -fehlerkorrigierend?
- c) Für welche Werte von  $r$  ist unser Code  $r$ -fehlerkorrigierend?
- d) In unserem Beispiel wurden 24 Datenpakete während der Übertragung beschädigt. Ist es möglich, ein Fehlerszenario zu konstruieren, in dem 25 Datenpakete falsch sind und die Nachricht dennoch fehlerfrei rekonstruiert werden kann?
- e) Wahrscheinlich haben Sie in den vorangegangenen Teilaufgaben festgestellt, dass die matrixförmige Anordnung von Reed-Solomon-Codewörtern die Fehlerkorrektur-eigenschaft formal verschlechtert. Heißt das, die zweidimensionale Anordnung bietet nur Nachteile? Falls nein, wo liegen die Vorteile?

In Abschnitt 6.4.5.4 haben Sie die EFM-Codierung kennengelernt, die bei der Speicherung von Daten auf CDs zum Einsatz kommt. Ein Ausschnitt aus der EFM-Tabelle ist auf Seite 442 abgedruckt.

### Aufgabe 6.31



Webcode  
6790

- a) Was ist die Aufgabe der EFM-Codierung?

- b) Auf einer Audio-CD befindet sich die folgende Sequenz aus Pits (P) und Lands (L):

PPPPPLLLPPPPPPPPPLLPPPLLLLLPPPPPPP

Welche Datenworte sind auf der CD gespeichert? Nehmen Sie für die Decodierung an, dass die ersten 3 Symbole EFM-Koppelbits sind.

- c) Recherchieren Sie, was sich hinter dem Begriff EFM+ verbirgt. Worin unterscheiden sich EFM+ und EFM?
- d) Der EFM-Code besitzt die Eigenschaft, dass in jedem Codewort zwei Einsen durch mindestens 2 Nullen getrennt werden und nicht mehr als 10 Nullen hintereinander vorkommen. In dieser Teilaufgabe soll gezeigt werden, dass genau 267 Bitsequenzen der Länge 14 die beiden EFM-Eigenschaften erfüllen. Wir führen den Beweis in zwei Schritten:
- 1) Es sei  $N(n)$  die Anzahl der Bitsequenzen der Länge  $n$ , die die erste EFM-Bedingung erfüllen: Zwei Einsen werden durch mindestens 2 Nullen getrennt. Rechnen Sie  $N(1)$  bis  $N(3)$  direkt aus, und geben Sie für  $N(n)$  eine Rekursionsbeziehung an, die den Funktionswert mithilfe berechneter Werte ausdrückt:

$$N(1) = \boxed{\phantom{00}}$$

$$N(2) = \boxed{\phantom{00}}$$

$$N(3) = \boxed{\phantom{00}}$$

$$N(n+3) = \boxed{\phantom{000}}$$

Benutzen Sie die ermittelte Rekursionsformel, um die folgende Tabelle auszufüllen:

$n$	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$N(n)$														

Der letzte Eintrag ist der Wert, den wir im nächsten Schritt benötigen:

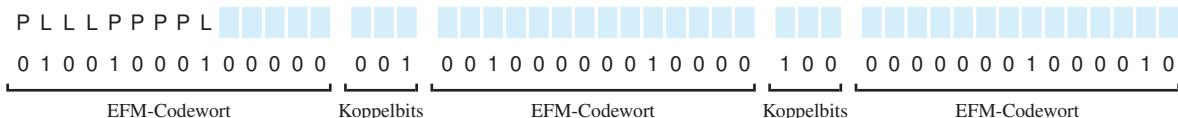
$$N(14) = \boxed{\phantom{000}}$$

- 2) Da wir in 1) auch solche Bitsequenzen mitgezählt haben, die gegen die zweite EFM-Bedingung verstößen, ist  $N(14)$  lediglich eine obere Schranke. Zeigen Sie, dass genau 267 Bitsequenzen verbleiben, wenn die Sequenzen, in denen mehr als 10 Nullen hintereinander vorkommen, eliminiert werden.
- e) Zeigen oder widerlegen Sie, dass der EFM-Code 1-fehlerkorrigierend ist. Führen Sie den Beweis, ohne einen Blick in die EFM-Tabelle zu werfen. Mit anderen Worten: Sie haben kein Wissen darüber, wie die EFM-Codewörter im Einzelnen aussehen.

Sie haben gelernt, dass ein binärer Datenstrom auf einer CD durch den Wechsel zwischen Vertiefungen (*Pits*, P) und Erhöhungen (*Lands*, L) codiert wird.

**Aufgabe 6.32****Webcode****6798**

- a) Ergänzen Sie die leer gelassenen Felder:



- b) Wenn Sie die Sequenz richtig erstellt haben, kommen darin deutlich mehr L als P vor. In der Praxis ist die Verteilung der Pits und Lands durchaus von Bedeutung: Ist sie über längere Zeit unausgewogen, so erzeugt dies ein störendes niederfrequentes Signal. Sehen Sie eine Möglichkeit, die Entstehung eines solchen niederfrequenten Signals durch eine geeignete Umcodierung zu verhindern?

Bei der Datenspeicherung auf optischen Datenträgern spielt das Interleaving-Prinzip eine große Rolle. Unter anderem sorgt es dafür, dass zeitlich aufeinanderfolgende Audiodaten auf einer CD-DA auf verschiedenen Frames verteilt und somit physikalisch voneinander entfernt gespeichert werden.

**Aufgabe 6.33****Webcode****6819**

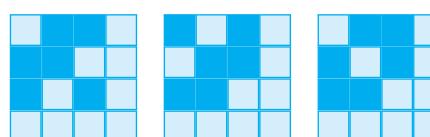
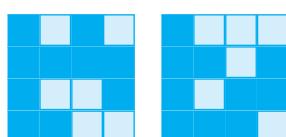
Früher wurde das Interleaving-Prinzip auch bei Festplatten eingesetzt.

- a) Welchen Zweck hat die Interleaving-Technik dort erfüllt?  
 b) Warum wird das Interleaving-Prinzip bei Festplatten nicht mehr eingesetzt, bei der optischen Datenspeicherung aber immer noch?

Eine Hadamard-Matrix heißt normalisiert, wenn die letzte Spalte und die letzte Zeile ausschließlich Nullen enthält:

Nicht normalisiert

Normalisiert

**Aufgabe 6.34****Webcode****6843**

- a) Zeigen Sie, dass zu jeder Hadamard-Matrix der Ordnung  $n$  eine normalisierte Hadamard-Matrix  $H'$  der gleichen Ordnung existiert.

- b) Zeigen Sie, dass jede normalisierte Hadamard-Matrix die folgende Eigenschaft erfüllt: Ist eine Zeile nicht die Nullzeile, so kommen dort genauso viele Nullen wie Einsen vor.

**Aufgabe 6.35****Webcode  
6874**

Es sei  $H$  eine Hadamard-Matrix der Ordnung  $n$  mit  $n > 2$ . In dieser Aufgabe wollen wir versuchen, mehr über die notwendigen Eigenschaften von  $n$  herauszubekommen.

Zu diesem Zweck werden wir weiter unten zwei Zeilen einer Hadamard-Matrix herausgreifen. Diese könnten beispielsweise so aussehen:



Vergleichen wir die beiden Zeilen miteinander, so können wir an jeder Position einen von vier Fällen beobachten. In den ersten beiden Fällen kommt in beiden Zeilen das gleiche Symbol vor (■ oder □). In den anderen beiden Fällen sind die Symbole unterschiedlich. Im Folgenden benutzen wir die Notation

$$\#(\square\square), \#(\square\square), \#(\square\square), \#(\square\square),$$

um anzugeben, wie oft die verschiedenen Fälle eintreten. In dem oben abgebildeten Beispiel würde also gelten:

$$\#(\square\square) = 3, \#(\square\square) = 1, \#(\square\square) = 2, \#(\square\square) = 2$$

Aus der vorangegangenen Übungsaufgabe wissen wir, dass eine Hadamard-Matrix  $H'$  der Ordnung  $n$  in normalisierter Form existiert. Aus dieser Matrix greifen wir nun zwei Zeilen heraus und nehmen an, beide seien von der Nullzeile verschieden. Dass wir zwei solche Zeilen finden können, garantiert die Forderung  $n > 2$ . Jetzt sind Sie an der Reihe!

- a) Überlegen Sie, wie die folgenden Gleichungen ergänzt werden müssen und begründen Sie Ihre Antwort.

$$\#(\square\square) + \#(\square\square) = \quad \quad \quad (6.61)$$

$$\#(\square\square) + \#(\square\square) = \quad \quad \quad (6.62)$$

- b) Was können wir über die folgenden Summen aussagen?

$$\#(\square\square) + \#(\square\square) = \quad \quad \quad (6.63)$$

$$\#(\square\square) + \#(\square\square) = \quad \quad \quad (6.64)$$

- c) Folgern Sie aus den Teilergebnissen, dass die Ordnung von  $H'$  und damit auch die Ordnung von  $H$  eine durch 4 teilbare Zahl sein muss.
- d) Bedeutet das Ergebnis der vorherigen Teilaufgabe, dass wir soeben die Hadamard-Vermutung bewiesen haben?

Die rekursive Vorschrift, die wir auf Seite 447 für die Konstruktion von Hadamard-Matrizen verwendet haben, ist nicht die einzige mögliche. Die folgenden drei erfüllen den gleichen Zweck:

**Aufgabe 6.36**

**Webcode  
6896**

Rekursionsvorschrift 1

$$\begin{aligned} H_1 &:= (1) \\ H_{2n} &:= \begin{pmatrix} -H_n & H_n \\ H_n & H_n \end{pmatrix} \end{aligned}$$

Rekursionsvorschrift 2

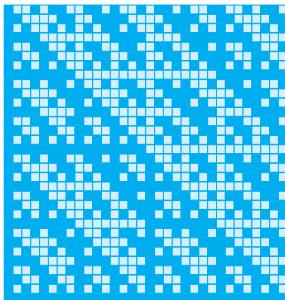
$$\begin{aligned} H_1 &:= (1) \\ H_{2n} &:= \begin{pmatrix} H_n & -H_n \\ H_n & H_n \end{pmatrix} \end{aligned}$$

Rekursionsvorschrift 3

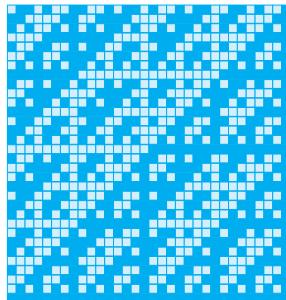
$$\begin{aligned} H_1 &:= (1) \\ H_{2n} &:= \begin{pmatrix} H_n & H_n \\ -H_n & H_n \end{pmatrix} \end{aligned}$$

Ordnen Sie die nachfolgend abgebildeten Grafiken den Konstruktionsvorschriften zu.

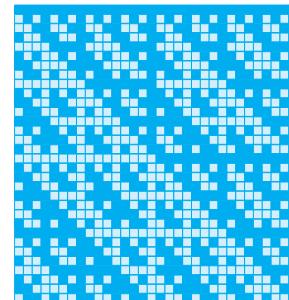
Matrix 1



Matrix 2



Matrix 3



In der linearen Algebra wird eine  $\mathbb{R} \times \mathbb{R}$ -Matrix als *orthogonal* bezeichnet, wenn die Zeilen- und Spaltenvektoren paarweise *orthonormal* sind. Das bedeutet, dass für diese Vektoren die folgende Eigenschaft gilt:

$$\mathbf{x} \cdot \mathbf{y} = \begin{cases} 0 & \text{falls } \mathbf{x} \neq \mathbf{y} \\ 1 & \text{falls } \mathbf{x} = \mathbf{y} \end{cases}$$

- a) Warum sind die allermeisten Hadamard-Matrizen nicht orthogonal?

**Aufgabe 6.37**

**Webcode  
6903**

- b) Gibt es eine einfache Möglichkeit, eine Hadamard-Matrix in eine orthogonale Matrix umzuformen?

## Aufgabe 6.38

40

**Webcode  
6906**

Welche Codes werden durch die folgenden Generatormatrizen erzeugt?

$r_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$r_1$	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
$r_2$	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1
$r_3$	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1
$r_4$	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
a)	$r_1 r_2$	0	0	0	1	0	0	0	1	0	0	0	1	0	0
	$r_1 r_3$	0	0	0	0	0	1	0	1	0	0	0	0	1	0
	$r_1 r_4$	0	0	0	0	0	0	0	0	0	1	0	1	0	1
	$r_2 r_3$	0	0	0	0	0	0	1	1	0	0	0	0	0	1
	$r_2 r_4$	0	0	0	0	0	0	0	0	0	0	1	1	0	0
	$r_3 r_4$	0	0	0	0	0	0	0	0	0	0	0	1	1	1

$r_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$r_1$	0	1	0	1	0	1	0	1	0	1	0	1	0	1
$r_2$	0	0	1	1	0	0	1	1	0	0	1	1	0	0
$r_3$	0	0	0	0	1	1	1	1	0	0	0	0	1	1
$r_4$	0	0	0	0	0	0	0	0	1	1	1	1	1	1
$r_1 r_2$	0	0	0	1	0	0	0	1	0	0	0	1	0	0
$r_1 r_3$	0	0	0	0	0	1	0	1	0	0	0	0	1	0
$r_1 r_4$	0	0	0	0	0	0	0	0	0	1	0	1	0	1
$r_2 r_3$	0	0	0	0	0	0	1	1	0	0	0	0	0	1
$r_2 r_4$	0	0	0	0	0	0	0	0	0	0	1	1	0	0
$r_3 r_4$	0	0	0	0	0	0	0	0	0	0	0	0	1	1
$r_1 r_2 r_3$	0	0	0	0	0	0	0	0	0	0	0	0	0	1
$r_1 r_2 r_4$	0	0	0	0	0	0	0	0	0	0	0	0	1	0
$r_1 r_3 r_4$	0	0	0	0	0	0	0	0	0	0	1	0	0	0
$r_2 r_3 r_4$	0	0	0	0	0	0	0	1	0	0	0	0	0	0

Simulieren Sie schrittweise das nachstehende Schieberegister für die Nachricht  $u = 10100$ . Geben Sie in jedem Schritt an, welche Bits in den Registern gespeichert sind und in welchem Zustand sich der korrespondierender Automat befindet.

**Aufgabe 6.39**

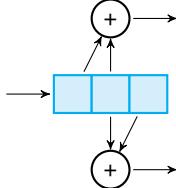
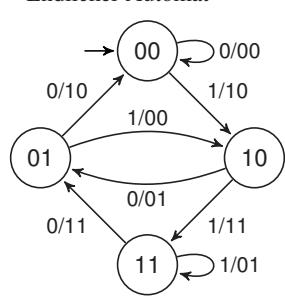
**Webcode  
6973**

Initialzustand	Schritt 1
<p>Eingabe: 10100</p> <p>Ausgabe:</p>	<p>Eingabe: 1 0100</p> <p>Ausgabe:</p>
<p>Schritt 2</p> <p>Eingabe: 1 0 100</p> <p>Ausgabe:</p>	<p>Schritt 3</p> <p>Eingabe: 10 1 00</p> <p>Ausgabe:</p>
<p>Schritt 4</p> <p>Eingabe: 101 0 0</p> <p>Ausgabe:</p>	<p>Schritt 5</p> <p>Eingabe: 1010 0</p> <p>Ausgabe:</p>

**Aufgabe 6.40**

**Webcode**  
**6977**

Gegenstand dieser Aufgabe sind zwei Faltungscodierungen, von denen die eine durch ein Schieberegisterdiagramm und die andere durch einen endlichen Automaten definiert ist:

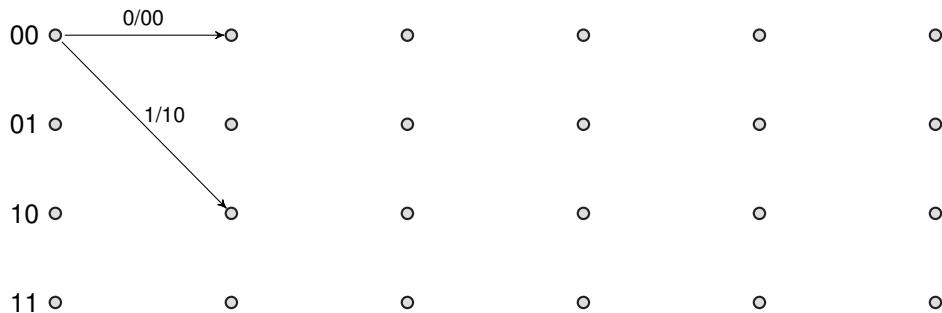
Faltungscodierung 1	Faltungscodierung 2
Schieberegisterdiagramm  	Endlicher Automat  

- a) Geben Sie für jede Codierung die jeweils andere Darstellung an.  
 b) Tragen Sie für jede Codierung die fehlenden Codewörter in die Tabellen ein:

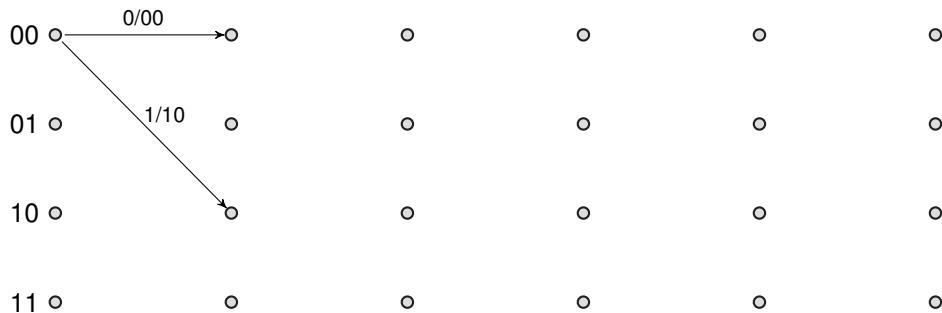
Faltungscodierung 1		Faltungscodierung 2	
Eingabe	Ausgabe	Eingabe	Ausgabe
000	0000000000	000	0000000000
001		001	
010		010	
011		011	
100		100	
101		101	
110		110	
111		111	

- c) Bestimmen Sie die Distanz beider Codes.  
 d) Wie viele Fehler können erkannt bzw. korrigiert werden?  
 e) Vervollständigen Sie die nachstehenden Trellis-Diagramme:

Faltungscodierung 1:

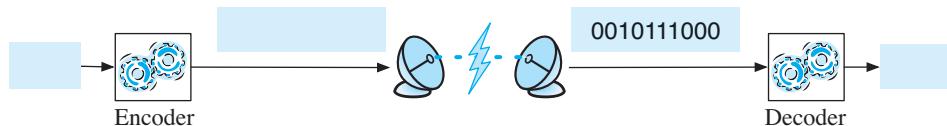


Faltungscodierung 2:

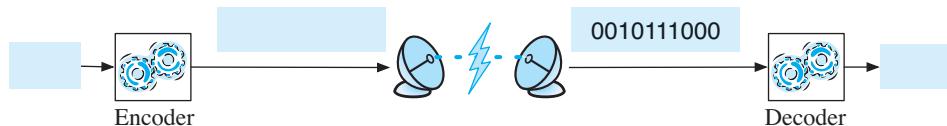


- f) Ergänzen Sie die leeren Felder in den abgebildeten Kommunikationsszenarien.

Faltungscodierung 1:



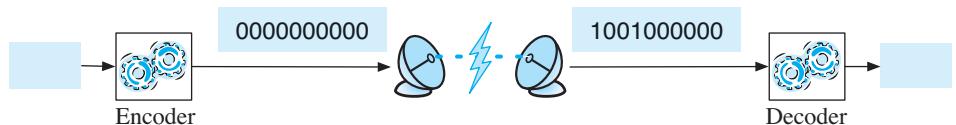
Faltungscodierung 2:



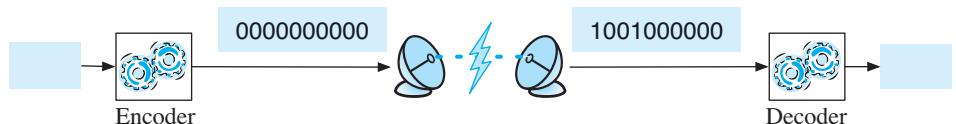
Hinweis: Verwenden Sie die Trellis-Diagramme, um die gesendeten Originalnachrichten zu rekonstruieren. Ist die Lösung in jedem Fall eindeutig bestimmt?

- g) In den nächsten beiden Übertragungsszenarien wurde das Codewort 0000000000 gesendet und an jeweils zwei Bitstellen verfälscht. Kann die Originalnachricht in beiden Fällen wiederhergestellt werden?

Faltungscodierung 1:

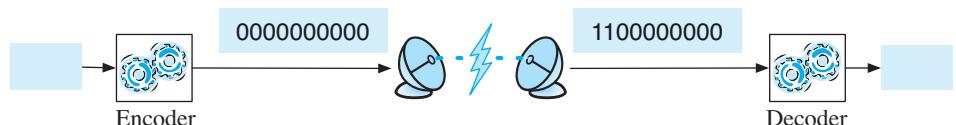


Faltungscodierung 2:

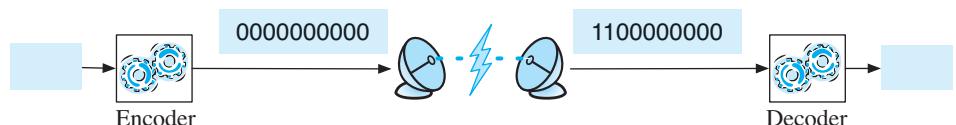


- h) Ändert sich das Ergebnis für das folgende Fehlerszenario?

Faltungscodierung 1:



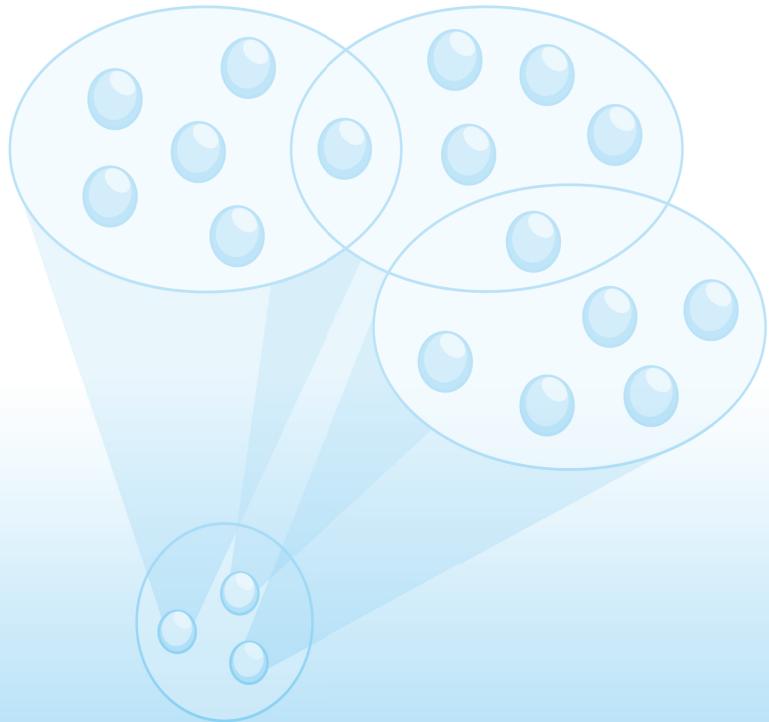
Faltungscodierung 2:



# 7 Grenzen der Kanalcodierung

In diesem Kapitel werden Sie ...

- die Singleton-Schranke kennenlernen,
- den Unterschied zwischen MDS-Codes und perfekten Codes verstehen,
- verschiedene Varianten des Golay-Codes entwickeln,
- Restfehlerwahrscheinlichkeiten berechnen,
- mit dem Kanalcodierungstheorem eine Perle der Codierungstheorie entdecken.



## 7.1 Motivation

Nachdem wir uns in Kapitel 6 ausgiebig mit den gängigen Kanalcodierungen beschäftigt haben, wollen wir uns in diesem Kapitel mit der prinzipiellen Leistungsfähigkeit fehlererkennender und fehlerkorrigierender Codes befassen. Wir wissen bereits, dass die Anzahl der Fehler, die ein Code erkennen bzw. korrigieren kann, durch die Code-Distanz bestimmt wird. Ferner wissen wir, dass die Erhöhung der Code-Distanz, wenn wir einen binären Blockcode der Länge  $n$  zugrunde legen, nur erreicht werden kann, indem die Codewörter innerhalb der  $2^n$  zur Verfügung stehenden Bitmuster möglichst weit voneinander entfernt platziert werden.

Die Zusammenhänge, die zwischen der Codewortlänge, der Anzahl der Codewörter und der Code-Distanz bestehen, haben wir bisher hauptsächlich qualitativ erfasst. Dies werden wir in diesem Kapitel ändern und in Abschnitt 7.2 genauer betrachten, wie diese Kenngrößen quantitativ zusammenhängen. Neben der Singleton-Schranke und den MDS-Codes werden wir insbesondere auch den Begriff des *perfekten Codes* formal einführen und anschließend, in Abschnitt 7.3, mit dem *Golay-Code* einen ganz speziellen Vertreter dieser Codeklasse vorstellen.

In Abschnitt 7.4 werden wir unsere quantitative Analyse weiter verfeinern und herausarbeiten, wie sich die *Restfehlerwahrscheinlichkeit* eines Codes berechnen lässt. Im Falle von fehlererkennenden Codes verbirgt sich hinter diesem Begriff die Wahrscheinlichkeit, dass ein Übertragungsfehler, allen Absicherungen zum Trotz, auf der Empfängerseite unbemerkt bleibt. Im Falle von fehlerkorrigierenden Codes ist damit die Wahrscheinlichkeit gemeint, dass das gesendete Codewort auf der Empfängerseite nicht mehr originalgetreu rekonstruiert werden kann.

Den Höhepunkt dieses Kapitels erreichen wir in Abschnitt 7.5 mit der Vorstellung des Kanalcodierungstheorems. Entdeckt wurde es von Claude Shannon, als er sich in den 40er-Jahren intensiv mit der Frage beschäftigte, wie die Restfehlerwahrscheinlichkeit eines Codes mit seiner Coderate zusammenhängt. So viel vorweg: Shannons Entdeckung ist nicht nur in seiner Tragweite verblüffend. Es ist vor allem die kontraintuitiv wirkende Aussage, die das Kanalcodierungstheorem zu einer Perle der Informations- und Codierungstheorie macht. Seien Sie gespannt!

## 7.2 Was kostet die Fehlerkorrektur?

### 7.2.1 Singleton-Schranke

„A  $q$ -nary error-correcting code with  $N = q^k$  code words of length  $n = k + r$  can have no greater minimum distance  $d$  than  $r + 1$ .“

R. C. Singleton [84]

Wir haben inzwischen ein ausgereiftes Verständnis dafür, wie die Distanz eines Codes dessen Fähigkeit beeinflusst, Fehler erkennen bzw. korrigieren zu können. Der Schlüssel hierfür ist Satz 6.3, der einen direkten Zusammenhang zwischen diesen Begriffen herstellt. Um die Fehlererkennungs- oder die Fehlerkorrektureigenschaft eines Codes zu verbessern, müssen wir dessen Distanz erhöhen. Das wiederum bedeutet, dass bei gleich bleibender Codewortlänge weniger Codewörter zur Verfügung stehen. In diesem Abschnitt werden wir versuchen, einen quantitativen Zusammenhang zwischen der Distanz  $\Delta C$  eines Codes und der Anzahl  $|C|$  seiner Codewörter herzustellen. Dabei gehen wir davon aus, dass  $C$  ein Blockcode der Länge  $n$  ist, der über einem Codealphabet  $\Pi$  mit  $q$  Symbolen definiert ist.

Ein vergleichsweise simpler Trick wird uns zum Ziel führen. Er besteht darin, aus  $C$  einen Code  $C'$  zu konstruieren, indem aus den Codewörtern von  $C$  jeweils die ersten  $\Delta C - 1$  Symbole herausgestrichen werden. Was können wir über diesen neuen Code aussagen? Da die Code-Distanz  $\Delta C$  um eins größer ist als die Anzahl der herausgestrichenen Symbole, müssen die gekürzten Codewörter immer noch paarweise verschiedenen sein. Das bedeutet, dass  $C'$  immer noch genauso viele Codewörter enthält wie  $C$ . In  $C'$  haben die Codewörter die folgende Länge:

$$n' = n - \Delta C + 1$$

Da es insgesamt nur  $q^{n-\Delta C+1}$  mögliche Codewörter der Länge  $n'$  gibt, können wir die folgende Abschätzung vornehmen:

$$|C| = |C'| \leq q^{n-\Delta C+1}$$

Damit haben wir ein wichtiges Zwischenergebnis erzielt:

 **Satz 7.1**

Ist  $\Pi$  ein Codealphabet mit  $q$  Elementen, so erfüllt jeder Code  $C \subseteq \Pi^n$  die folgende Beziehung:

$$|C| \leq q^{n-\Delta C+1}$$

Wir wollen uns über die Konsequenzen klar werden, die sich hieraus für einen  $[n, k]$ -Code ergeben. Die Notation  $[n, k]$  ist uns aus den Abschnitten über lineare Codes vertraut und bedeutet das Folgende: Ist  $q$  die Anzahl der Symbole des Codealphabets, so gibt es in einem  $[n, k]$ -Code genau  $q^k$  Codewörter der Länge  $n$ . Für einen Code mit diesen Parametern können wir aus Satz 7.1 den folgenden Schluss ziehen:

$$q^k \leq q^{n-\Delta C+1}$$

Diese Ungleichung ist äquivalent zu

$$k \leq n - \Delta C + 1,$$

und nach  $\Delta C$  aufgelöst, erhalten wir

$$\Delta C \leq n - k + 1. \quad (7.1)$$

Auf der rechten Seite von (7.1) steht die sogenannte *Singleton-Schranke*, die im Jahr 1964 von Richard Collom Singleton gefunden wurde. Sie versetzt uns in die Lage, die Distanz eines  $[n, k]$ -Codes nach oben abzuschätzen.

 **Satz 7.2 (Singleton, 1964)**

Jeder  $[n, k]$ -Code erfüllt die Abschätzung  $\Delta C \leq n - k + 1$

### 7.2.2 MDS-Codes

Die Singleton-Schranke begrenzt die Code-Distanz nur in eine Richtung. Sie besagt, dass  $\Delta C$  für einen  $[n, k]$ -Code niemals größer sein kann als  $n - k + 1$ , schließt aber nicht aus, dass sie kleiner ist. Ein Blockcode, dessen Distanz die Singleton-Schranke exakt trifft, besitzt in der Codierungstheorie einen speziellen Namen: Er wird als *Maximum Distance Separable Code*, kurz *MDS-Code*, bezeichnet:

**Definition 7.1 (MDS-Code)**

Ein  $[n, k]$ -Code mit  $\Delta C = n - k + 1$  heißt *MDS-Code*.

In Abbildung 7.1 sind drei sehr einfache MDS-Codes zu sehen, die in der Literatur als *triviale MDS-Codes* bezeichnet werden. Halten wir nach weiteren MDS-Codes Ausschau, so tritt ein interessantes Phänomen zu Tage. Beschränken wir unsere Suche, wie so oft in diesem Buch, auf Binärcodes, so werden wir nicht fündig. Es lässt sich zeigen, dass außer den trivialen Codes aus Abbildung 7.1 überhaupt keine anderen MDS-Binärcodes existieren [97].

Lassen wir dagegen beliebige Codealphabete zu, so werden wir bei einer Codeklasse fündig, die wir gut kennen: den Reed-Solomon-Codes. Dass wir es hier tatsächlich mit MDS-Codes zu tun haben, lässt sich leicht nachrechnen. Da ein  $[n, k]$ -Reed-Solomon-Code nach Satz 6.14  $(n-k)$ -fehlererkennend, aber nicht  $(n-k+1)$ -fehlererkennend ist, folgt aus Satz 6.3 sofort der gesuchte Zusammenhang:

$$\Delta C = n - k + 1$$

Wir halten fest:

**Satz 7.3**

Reed-Solomon-Codes sind MDS-Codes.

Gerade haben wir erwähnt, dass innerhalb der Klasse der Binärcodes ausschließlich triviale MDS-Codes existieren. Folgerichtig müssen auch die in der Praxis so wichtigen Hamming-Codes die Singleton-Schranke verfehlen, und am Beispiel des  $[7,4]$ -Hamming-Code können wir uns auch leicht davon überzeugen. Für diesen Code liefert die Singleton-Schranke den Wert

$$n - k + 1 = 7 - 4 + 1 = 4,$$

die Code-Distanz liegt mit  $\Delta C = 3$  aber darunter. Für den  $[15,11]$ -Hamming-Code sieht es nicht besser aus. In diesem Fall berechnet sich die Singleton-Schranke zu

$$n - k + 1 = 15 - 11 + 1 = 5,$$

die Code-Distanz  $\Delta C$  ist aber immer noch 3. Dies offenbart ein interessantes Phänomen. Mit jedem zusätzlichen Prüfbit vergrößert sich der

 **$[n, n-1]$ -Paritätscode**

Ein Codewort der Länge  $n$  besteht aus  $n-1$  Nachrichtenbits und einem Paritätsbit.

$$\left. \begin{array}{l} \Delta C = 2 \\ k = n-1 \end{array} \right\} \Rightarrow \Delta C = n - k + 1$$

 **$[n, 1]$ -Wiederholungscode**

Es gibt nur zwei Codewörter:  
 $\underbrace{000 \dots 0}_n$  und  $\underbrace{111 \dots 1}_n$

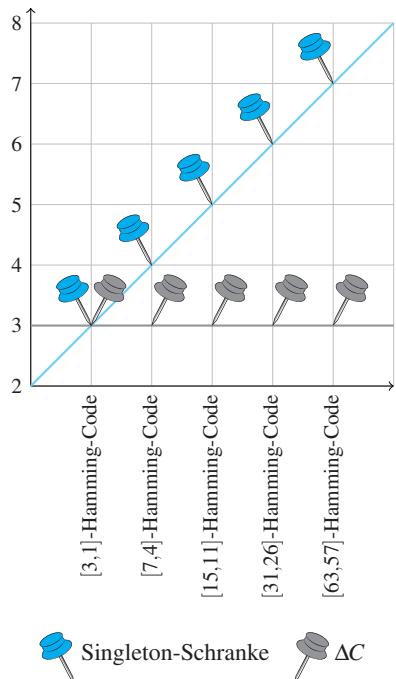
$$\left. \begin{array}{l} \Delta C = n \\ k = 1 \end{array} \right\} \Rightarrow \Delta C = n - k + 1$$

 **$[n, n]$ -Code**

Sämtliche Sequenzen der Länge  $n$  sind Codewörter:  
 $C = \Pi^n$

$$\left. \begin{array}{l} \Delta C = 1 \\ k = n \end{array} \right\} \Rightarrow \Delta C = n - k + 1$$

**Abb. 7.1:** Ein MDS-Code besitzt die Eigenschaft, dass seine Code-Distanz mit der Singleton-Schranke übereinstimmt.



**Abb. 7.2:** Mit jedem weiteren Prüfbit entfernt sich der Hamming-Code weiter von der Singleton-Schranke.

Abstand zur Singleton-Schranke um 1. Da die Code-Distanz aber konstant bleibt, können wir für jeden beliebig vorgegebenen Wert einen Hamming-Code finden, der die Singleton-Schranke entsprechend weit verfehlt (Abbildung 7.2).

Sind die Hamming-Codes also gar nicht so optimal, wie wir sie immer dargestellt haben? Lassen sich die Codewörter vielleicht so umordnen, dass sich die Code-Distanz vergrößert? Die Antwort lautet Nein, und im nächsten Abschnitt werden wir dies zweifelsfrei belegen. Sie werden dort sehen, dass die Hamming-Codes, in einem sehr wahren Sinne des Wortes, perfekt sind.

### 7.2.3 Perfekte Codes

In den vorangegangenen Abschnitten haben wir einen einfachen Zusammenhang zwischen der Anzahl der Codewörter und der Distanz eines Codes aufgestellt und sind auf diese Weise direkt zu den MDS-Codes gelangt. Wir beginnen diesen Abschnitt mit einer ähnlichen Überlegung, verfolgen dabei aber eine geringfügig abweichende Fragestellung. Anstatt uns über die Anzahl der Codewörter eines  $[n, k]$ -Codes  $C$  Gedanken zu machen, wollen wir uns überlegen, um welchen Betrag die Codewortlänge  $n$  den Parameter  $k$  übersteigen muss, damit ein  $r$ -fehlerkorrigierender Code entstehen kann. Denken Sie daran, dass die Differenz  $n - k$  bei systematischen Codierungen eine ganz intuitive Bedeutung besitzt: Sie gibt an, um wie viele Prüfbits eine Nachricht der Länge  $k$  im Zuge der Codierung ergänzt wird.

Wir werden das Ergebnis dieses Abschnitts auf zwei verschiedene Weisen herleiten: Einmal über eine kombinatorische Überlegung, indem wir über die Anzahl der möglichen Fehlerkombinationen argumentieren, und ein anderes Mal über eine räumliche Überlegung, die auf den Begriff der Hamming-Kugel zurückgreift.

#### Kombinatorische Überlegung

Wir nehmen an,  $C$  sei ein  $r$ -fehlerkorrigierender  $[n, k]$ -Code, es gelte also

$$\Delta C = 2r + 1$$

Das kombinatorische Argument, auf das wir in diesem Abschnitt zurückgreifen, basiert auf der folgenden Überlegung: Wenn ein Code  $r$ -fehlerkorrigierend ist, so ist dies nur möglich, wenn der Empfänger für

eine ankommende Bitsequenz  $w$  der Länge  $n$  die folgenden Szenarien unterscheiden kann:

- In  $w$  sind alle Bits korrekt.  1 Möglichkeit
- In  $w$  wurde 1 Bit verfälscht.   $n$  Möglichkeiten
- In  $w$  wurden 2 Bits verfälscht.   $\binom{n}{2}$  Möglichkeiten
- In  $w$  wurden 3 Bits verfälscht.   $\binom{n}{3}$  Möglichkeiten
- ...
- In  $w$  wurden  $r$  Bits verfälscht.   $\binom{n}{r}$  Möglichkeiten

Insgesamt muss der Empfänger somit

$$1 + n + \sum_{i=2}^r \binom{n}{i} \quad (7.2)$$

Fehlerszenarien unterscheiden. Etwas schöner können wir (7.2) auch so aufschreiben:

$$\sum_{i=0}^r \binom{n}{i}$$

Da sich mit  $n - k$  zusätzlich übertragenen Prüfbits maximal

$$2^{n-k}$$

verschiedene Fehlerszenarien codieren lassen, können wir die folgende Abschätzung vornehmen:

$$2^{n-k} \geq \sum_{i=0}^r \binom{n}{i} \quad (7.3)$$

Logarithmieren wir beide Seiten, so entsteht das gesuchte Ergebnis:

### Satz 7.4

Jeder  $r$ -fehlerkorrigierende  $[n, k]$ -Code über einem binären Codealphabet erfüllt die folgende Abschätzung:

$$n - k \geq \log_2 \sum_{i=0}^r \binom{n}{i}$$

## Räumliche Überlegung

Wir wollen Satz 7.4 ein zweites Mal herleiten und dabei auf ein Argument zurückgreifen, das sehr deutlich aufzeigt, was sich hinter der Abschätzung wirklich verbirgt. Um die Überlegungen möglichst einfach zu halten, beschränken wir uns zunächst auf die Untersuchung von Binärcodes, also Codes über dem Codealphabet  $\Pi = \{0, 1\}$ . Als Erstes werden wir untersuchen, wie viele Bitsequenzen in der Hamming-Kugel  $K_r(v)$  eines Codeworts  $v$  vorhanden sind. Die ermittelte Anzahl bezeichnen wir als die Größe einer Hamming-Kugel und benutzen dafür die Abkürzung  $|K_r(v)|$ :

$$|K_r(v)| := \begin{array}{l} \text{Größe der Hamming-Kugel mit dem} \\ \text{Mittelpunkt } v \text{ und dem Radius } r \end{array}$$

Ist  $v$  ein Codewort der Länge  $n$ , so können wir die folgende Überlegung anstellen: In der Hamming-Kugel  $K_r(v)$  sind alle Bitsequenzen enthalten, die

- zu  $v$  die Distanz 0 aufweisen oder ☞ 1 Sequenz
- zu  $v$  die Distanz 1 aufweisen oder ☞  $n$  Sequenzen
- zu  $v$  die Distanz 2 aufweisen oder ☞  $\binom{n}{2}$  Sequenzen
- zu  $v$  die Distanz 3 aufweisen oder ☞  $\binom{n}{3}$  Sequenzen
- ...
- zu  $v$  die Distanz  $r$  aufweisen. ☞  $\binom{n}{r}$  Sequenzen

Damit können wir die Größe einer Hamming-Kugel über die folgende Formel quantitativ erfassen:

$$|K_r(v)| = \sum_{i=0}^r \binom{n}{i} \quad (7.4)$$

Insgesamt gibt es  $2^n$  verschiedene Binärsequenzen der Länge  $n$ . Aus den Überlegungen, die wir auf Seite 358 angestellt haben, wissen wir, dass sich die Hamming-Kugeln eines  $r$ -fehlerkorrigierenden Codes weder schneiden noch berühren dürfen, d. h., die Menge  $\mathbb{Z}_2^n$  muss so viele Bitvektoren vorhalten, dass  $2^k$  Hamming-Kugeln mit dem Radius  $r$  darin Platz finden. Dies spielt uns sofort die folgende Abschätzung in die Hände:

$$2^n \geq 2^k \cdot |K_r(v)|$$

Bringen wir den Faktor  $2^k$  auf die linke Seite, so erhalten wir, unter Berücksichtigung von (7.4), die folgende Beziehung:

$$2^{n-k} \geq \sum_{i=0}^r \binom{n}{i}$$

Damit sind wir am Ziel. Wir haben die Abschätzung (7.3) ein zweites Mal hergeleitet.

Als Nächstes wollen wir unsere Untersuchungen auf Codes ausweiten, die über beliebigen Codealphabeten definiert sind. Die Überlegung, die uns zum Ziel führen wird, ist die gleiche wie oben: In der Hamming-Kugel  $K_r(v)$  sind alle Bitsequenzen enthalten, die

- zu  $v$  die Distanz 0 aufweisen oder 👉 1 Sequenz
- zu  $v$  die Distanz 1 aufweisen oder 👉  $n \cdot (q - 1)$  Sequenzen
- zu  $v$  die Distanz 2 aufweisen oder 👉  $\binom{n}{2} \cdot (q - 1)^2$  Sequenzen
- zu  $v$  die Distanz 3 aufweisen oder 👉  $\binom{n}{3} \cdot (q - 1)^3$  Sequenzen
- ...
- zu  $v$  die Distanz  $r$  aufweisen. 👉  $\binom{n}{r} \cdot (q - 1)^r$  Sequenzen

Hieraus ergibt sich die folgende Formel für  $|K_r(v)|$ :

$$|K_r(v)| = \sum_{i=0}^r \binom{n}{i} (q - 1)^i \quad (7.5)$$

Für  $q = 2$  wird (7.5) zu Formel (7.4), die wir weiter oben hergeleitet hatten.

Auch die weitere Argumentation verläuft analog: Ist ein Code  $r$ -korrigierend, so dürfen sich die Hamming-Kugeln weder schneiden noch berühren. Das bedeutet, dass die Menge  $\Pi^n$  so viele Bitvektoren vorhalten muss, dass  $q^k$  Hamming-Kugeln mit dem Radius  $r$  darin Platz finden. Dies ergibt die folgende Abschätzung:

$$q^n \geq q^k \cdot |K_r(v)|$$

Bringen wir den Faktor  $q^k$  auf die linke Seite, so erhalten wir, unter Berücksichtigung von (7.5), die folgende Beziehung:

$$q^{n-k} \geq \sum_{i=0}^r \binom{n}{i} (q - 1)^i$$

Logarithmieren wir beide Seiten, so sind wir am Ziel. Wir haben es geschafft, die inhaltliche Aussage von Satz 7.4 auf Codes auszuweiten, die über beliebigen Codealphabeten definiert sind:


**Satz 7.5**

Jeder  $r$ -fehlerkorrigierende  $[n, k]$ -Code über einem Codealphabet mit  $q$  Elementen erfüllt die folgende Abschätzung:

$$n - k \geq \log_q \sum_{i=0}^r \binom{n}{i} (q-1)^i$$

Wir wollen das erworbene Wissen auf eine spezielle Codeklasse anwenden, zu denen auch die Hamming-Codes gehören: die Klasse der 1-fehlerkorrigierenden Codes über dem binären Codealphabet  $\Pi = \{0, 1\}$ . Für diese gilt  $r = 1$ , und wir erhalten aus Satz 7.4:

$$n - k \geq \log_2 (1 + n) \quad (7.6)$$

Abbildung 7.3 zeigt, welche Werte die linke und rechte Seite dieser Ungleichung für Hamming-Codes verschiedener Längen annehmen.

Besonders interessant sind die Hamming-Codes mit den Parametern  $[3, 1]$ ,  $[7, 4]$ ,  $[15, 11]$  und  $[31, 26]$ . Bei ihnen sind die linke und die rechte Seite von (7.6) identisch und die Menge  $\Pi^n$  enthält genau so viele Bitmuster, dass die  $2^k$  Hamming-Kugeln exakt hineinpassen. Das bedeutet, dass die Hamming-Kugeln den Raum so perfekt ausfüllen, dass keine Zwischenräume verbleiben: Jedes Bitmuster gehört zu einer Hamming-Kugel, und für jedes Bitmuster ist diese Kugel eindeutig bestimmt. In der Codierungstheorie haben solche Codes einen eigenen Namen. Sie werden als *perfekte Codes* bezeichnet:

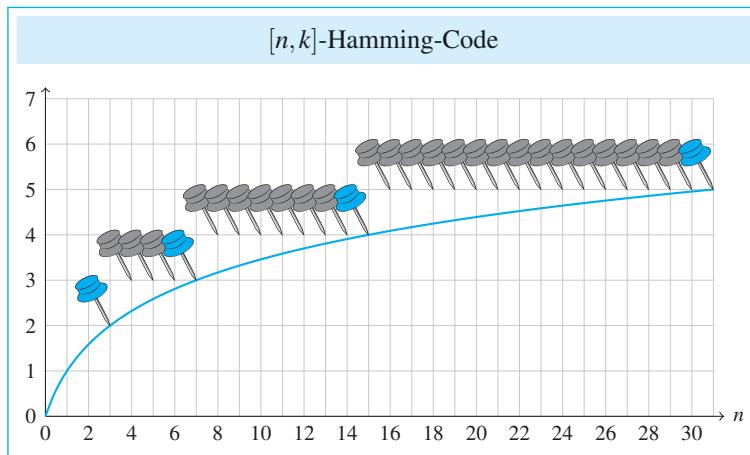

**Definition 7.2**

Ein  $r$ -fehlerkorrigierender  $[n, k]$ -Code über einem Codealphabet mit  $q$  Elementen heißt *perfekt*, wenn er die folgende Beziehung erfüllt:

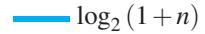
$$|\mathbf{K}_r(\mathbf{v})| = q^{n-k}$$

$|\mathbf{K}_r(\mathbf{v})|$  ist die Größe einer Hamming-Kugel mit dem Radius  $r$ .

Beschränken wir uns auf 1-fehlerkorrigierende Binärcodes, so können wir die Eigenschaft, ein perfekter Code zu sein, so formulieren, wie wir



**Abb. 7.3:** Unter den dargestellten Hamming-Codes spielen jene mit den Codewortlängen 3, 7, 15 und 31 eine besondere Rolle. Bei ihnen entspricht die Anzahl der Bitsequenzen in den Hamming-Kugeln exakt der Anzahl der insgesamt zur Verfügung stehenden Bitmustern.

  $n - k$    $\log_2(1 + n)$

es weiter oben bereits getan haben. Es reicht dann aus, in Ungleichung (7.6) die linke und die rechte Seite gleichsetzen:

### Satz 7.6

Für einen 1-fehlerkorrigierenden  $[n, k]$ -Code über einem binären Codealphabet gilt:

$$C \text{ ist perfekt} \Leftrightarrow n - k = \log_2(1 + n)$$

Dieser Satz macht klar, welche Bedingung ein Hamming-Code erfüllen muss, um perfekt zu sein. Da auf beiden Seiten der Gleichung eine ganze Zahl steht, kann ein Hamming-Code nur dann perfekt sein, wenn die Länge seiner Codewörter einer um 1 reduzierten Zweierpotenz entspricht. In diesem Fall lässt sich die Codewortlänge  $n$  in der Form  $2^m - 1$  darstellen, wobei  $m$  dann gleichzeitig die Anzahl der Prüfbits ist. Daraus folgt:

$$n - k = m = \log_2(2^m) = \log_2(1 + 2^m - 1) = \log_2(1 + n)$$

Spätestens an dieser Stelle ist klar, warum wir in den vorangegangenen Abschnitten so häufig auf die Hamming-Codes mit den Parametern [7,4] und [15,11] als Beispiele zurückgegriffen haben. Sie gehören zu den perfekten Hamming-Codes und nehmen aus diesem Grund eine Sonderstellung ein.

Wir wollen uns auf die Suche nach weiteren perfekten Codes begeben und zu diesem Zweck einen Blick auf Tabelle 7.1 werfen. Exemplarisch



Wir wollen uns an dieser Stelle eine banal klingende Frage stellen: Was genau ist ein *Hamming-Code*? Rekapitulieren wir das bisher Dagewesene im Detail, so ist die Beantwortung gar nicht so einfach, denn wir haben den Namen *Hamming-Code* an verschiedenen Stellen dieses Buchs für unterschiedliche Codes benutzt. Neben den Hamming-Codes in ihrer historischen Variante, wie wir sie in Abschnitt 6.4.2 beschrieben haben, ist uns auch eine zyklische Variante des Hamming-Codes begegnet, genauso wie eine systematische Variante, in der die Prüfbits gesammelt am Ende des Codeworts vorkommen. Was also genau ist ein Hamming-Code? Dass der Name *Hamming-Code* in der Informations- und Codierungstheorie so freizügig verwendet wird, hat seinen Grund darin, dass zumeist gar nicht die genaue Bitverteilung innerhalb der Codewörter eine Rolle spielt, sondern

nur seine abstrakten Eigenschaften. Zuallererst ist dies die Eigenschaft, ein linearer Code zu sein und eine Distanz von 3 aufzuweisen; Hamming-Codes sind also allesamt 1-fehlerkorrigierend. Zusätzlich wird in den meisten Publikationen gefordert, dass die Hamming-Kugeln den Code Raum erschöpfend ausfüllen. Nach dieser Definition sind Hamming-Codes also perfekte lineare Codes mit  $\Delta C = 3$ . In Abschnitt 6.4.2 waren wir weniger streng. Dort haben wir einen Code als Hamming-Code bezeichnet, wenn er nach dem dort geschilderten Schema konstruiert wurde.

Legen wir die strengere Definition zugrunde, die nur perfekte Codes als Hamming-Codes bezeichnet, so wären von den Codes aus Abbildung 7.3 lediglich die farblich hervorgehobenen auch Hamming-Codes. Behalten Sie die unterschiedlichen Definitionen im Hinterkopf, um augenscheinliche Widersprüche zu vermeiden.

finden wir dort den Wert  $|K_r(v)|$  für verschiedene Codewortlängen und verschiedene Kugelradien ausgerechnet. Aus Definition 7.2 folgt unmittelbar, dass ein Binärkode überhaupt nur dann perfekt sein kann, wenn die Größe der Hamming-Kugeln eine Zweierpotenz ist. Das Ergebnis ist ernüchternd. Durchsuchen wir die Zahlen nach Zweierpotenzen, so werden wir nur vereinzelt fündig.

Die meisten Treffer finden wir in der ersten Spalte, unter den Hamming-Kugeln mit dem Radius 1. Überraschend ist dieses Ergebnis nicht. Hinter den markierten Einträgen verborgen sich die Hamming-Codes, die wir bereits weiter oben als perfekte Codes identifiziert haben.

Der [3,1]-Hamming-Code verdient eine weitere Betrachtung. Er besteht aus den beiden Codewörtern 000 und 111 und ist damit nichts anderes als ein Wiederholungscode der Länge 3. Den Wiederholungscode finden wir auch in den anderen Spalten wieder. Der markierte Eintrag, mit dem die zweite Spalte beginnt, ist der 2-fehlerkorrigierende Wiederholungscode der Länge 5 und der markierte Eintrag, mit dem die dritte Spalte beginnt, ist der 3-fehlerkorrigierende Wiederholungscode der Länge 7. Die Wiederholungscodes sind hauptsächlich von theoretischem Interesse und werden als *triviale perfekte Codes* bezeichnet.

Dennoch sind wir mit unserer Diskussion noch nicht ganz am Ende. In der dritten Spalte verbirgt sich, unscheinbar versteckt, noch eine andere Kombination aus Codewortlänge und Hamming-Radius, die eine Zweierpotenz hervorbringt. Sie ist es Wert, genauer hinzusehen!

Radius 1	Radius 2	Radius 3
$\binom{3}{0} + \binom{3}{1} = 4$		
$\binom{4}{0} + \binom{4}{1} = 5$		
$\binom{5}{0} + \binom{5}{1} = 6$	$\binom{5}{0} + \binom{5}{1} + \binom{5}{2} = 16$	
$\binom{6}{0} + \binom{6}{1} = 7$	$\binom{6}{0} + \binom{6}{1} + \binom{6}{2} = 22$	
$\binom{7}{0} + \binom{7}{1} = 8$	$\binom{7}{0} + \binom{7}{1} + \binom{7}{2} = 29$	$\binom{7}{0} + \binom{7}{1} + \binom{7}{2} + \binom{7}{3} = 64$
$\binom{8}{0} + \binom{8}{1} = 9$	$\binom{8}{0} + \binom{8}{1} + \binom{8}{2} = 37$	$\binom{8}{0} + \binom{8}{1} + \binom{8}{2} + \binom{8}{3} = 93$
$\binom{9}{0} + \binom{9}{1} = 10$	$\binom{9}{0} + \binom{9}{1} + \binom{9}{2} = 46$	$\binom{9}{0} + \binom{9}{1} + \binom{9}{2} + \binom{9}{3} = 130$
$\binom{10}{0} + \binom{10}{1} = 11$	$\binom{10}{0} + \binom{10}{1} + \binom{10}{2} = 56$	$\binom{10}{0} + \binom{10}{1} + \binom{10}{2} + \binom{10}{3} = 176$
$\binom{11}{0} + \binom{11}{1} = 12$	$\binom{11}{0} + \binom{11}{1} + \binom{11}{2} = 67$	$\binom{11}{0} + \binom{11}{1} + \binom{11}{2} + \binom{11}{3} = 232$
$\binom{12}{0} + \binom{12}{1} = 13$	$\binom{12}{0} + \binom{12}{1} + \binom{12}{2} = 79$	$\binom{12}{0} + \binom{12}{1} + \binom{12}{2} + \binom{12}{3} = 299$
$\binom{13}{0} + \binom{13}{1} = 14$	$\binom{13}{0} + \binom{13}{1} + \binom{13}{2} = 92$	$\binom{13}{0} + \binom{13}{1} + \binom{13}{2} + \binom{13}{3} = 378$
$\binom{14}{0} + \binom{14}{1} = 15$	$\binom{14}{0} + \binom{14}{1} + \binom{14}{2} = 106$	$\binom{14}{0} + \binom{14}{1} + \binom{14}{2} + \binom{14}{3} = 470$
$\binom{15}{0} + \binom{15}{1} = 16$	$\binom{15}{0} + \binom{15}{1} + \binom{15}{2} = 121$	$\binom{15}{0} + \binom{15}{1} + \binom{15}{2} + \binom{15}{3} = 576$
$\binom{16}{0} + \binom{16}{1} = 17$	$\binom{16}{0} + \binom{16}{1} + \binom{16}{2} = 137$	$\binom{16}{0} + \binom{16}{1} + \binom{16}{2} + \binom{16}{3} = 697$
$\binom{17}{0} + \binom{17}{1} = 18$	$\binom{17}{0} + \binom{17}{1} + \binom{17}{2} = 154$	$\binom{17}{0} + \binom{17}{1} + \binom{17}{2} + \binom{17}{3} = 834$
$\binom{18}{0} + \binom{18}{1} = 19$	$\binom{18}{0} + \binom{18}{1} + \binom{18}{2} = 172$	$\binom{18}{0} + \binom{18}{1} + \binom{18}{2} + \binom{18}{3} = 988$
$\binom{19}{0} + \binom{19}{1} = 20$	$\binom{19}{0} + \binom{19}{1} + \binom{19}{2} = 191$	$\binom{19}{0} + \binom{19}{1} + \binom{19}{2} + \binom{19}{3} = 1160$
$\binom{20}{0} + \binom{20}{1} = 21$	$\binom{20}{0} + \binom{20}{1} + \binom{20}{2} = 211$	$\binom{20}{0} + \binom{20}{1} + \binom{20}{2} + \binom{20}{3} = 1351$
$\binom{21}{0} + \binom{21}{1} = 22$	$\binom{21}{0} + \binom{21}{1} + \binom{21}{2} = 232$	$\binom{21}{0} + \binom{21}{1} + \binom{21}{2} + \binom{21}{3} = 1562$
$\binom{22}{0} + \binom{22}{1} = 23$	$\binom{22}{0} + \binom{22}{1} + \binom{22}{2} = 254$	$\binom{22}{0} + \binom{22}{1} + \binom{22}{2} + \binom{22}{3} = 1794$
$\binom{23}{0} + \binom{23}{1} = 24$	$\binom{23}{0} + \binom{23}{1} + \binom{23}{2} = 277$	$\binom{23}{0} + \binom{23}{1} + \binom{23}{2} + \binom{23}{3} = 2048$
$\binom{24}{0} + \binom{24}{1} = 25$	$\binom{24}{0} + \binom{24}{1} + \binom{24}{2} = 301$	$\binom{24}{0} + \binom{24}{1} + \binom{24}{2} + \binom{24}{3} = 2325$
$\binom{25}{0} + \binom{25}{1} = 26$	$\binom{25}{0} + \binom{25}{1} + \binom{25}{2} = 326$	$\binom{25}{0} + \binom{25}{1} + \binom{25}{2} + \binom{25}{3} = 2626$
$\binom{26}{0} + \binom{26}{1} = 27$	$\binom{26}{0} + \binom{26}{1} + \binom{26}{2} = 352$	$\binom{26}{0} + \binom{26}{1} + \binom{26}{2} + \binom{26}{3} = 2952$
$\binom{27}{0} + \binom{27}{1} = 28$	$\binom{27}{0} + \binom{27}{1} + \binom{27}{2} = 379$	$\binom{27}{0} + \binom{27}{1} + \binom{27}{2} + \binom{27}{3} = 3304$
$\binom{28}{0} + \binom{28}{1} = 29$	$\binom{28}{0} + \binom{28}{1} + \binom{28}{2} = 407$	$\binom{28}{0} + \binom{28}{1} + \binom{28}{2} + \binom{28}{3} = 3683$
$\binom{29}{0} + \binom{29}{1} = 30$	$\binom{29}{0} + \binom{29}{1} + \binom{29}{2} = 436$	$\binom{29}{0} + \binom{29}{1} + \binom{29}{2} + \binom{29}{3} = 4090$
$\binom{30}{0} + \binom{30}{1} = 31$	$\binom{30}{0} + \binom{30}{1} + \binom{30}{2} = 466$	$\binom{30}{0} + \binom{30}{1} + \binom{30}{2} + \binom{30}{3} = 4526$
$\binom{31}{0} + \binom{31}{1} = 32$	$\binom{31}{0} + \binom{31}{1} + \binom{31}{2} = 497$	$\binom{31}{0} + \binom{31}{1} + \binom{31}{2} + \binom{31}{3} = 4992$

Tab. 7.1: Perfekte Codes sind rar!

TABLE I

	$Y_1$	$Y_2$	$Y_3$	$Y_4$	$Y_5$	$Y_6$	$Y_7$	$Y_8$	$Y_9$	$Y_{10}$	$Y_{11}$	$Y_{12}$		$Y_1$	$Y_2$	$Y_3$	$Y_4$	$Y_5$	$Y_6$	
$X_1$	1	0	0	1	1	1	0	0	0	1	1	1		$X_1$	1	1	1	2	2	0
$X_2$	1	0	1	0	1	1	0	1	1	0	0	1		$X_2$	1	1	2	1	0	2
$X_3$	1	0	1	1	0	1	1	0	1	0	1	0		$X_3$	1	2	1	0	1	2
$X_4$	1	0	1	1	1	0	1	1	1	0	0	0		$X_4$	1	2	0	1	2	1
$X_5$	1	1	0	0	1	1	1	0	1	1	0	0		$X_5$	1	0	2	2	1	1
$X_6$	1	1	0	1	0	1	1	1	0	0	0	0								
$X_7$	1	1	0	1	1	0	0	1	1	0	1	0								
$X_8$	1	1	1	0	0	1	0	1	0	1	1	0								
$X_9$	1	1	1	0	1	0	1	0	0	0	0	1								
$X_{10}$	1	1	1	1	0	0	0	0	0	1	1	0								
$X_{11}$	0	1	1	1	1	1	1	1	1	1	1	1								

Abb. 7.4: Auszug aus der Originalarbeit von Marcel J. E. Golay aus dem Jahr 1949 [35]

### 7.3 Golay-Codes

Setzen sich die Codewörter eines Binärcodes aus 23 Bits zusammen, so können wir die Anzahl der Bitsequenzen, die eine Hamming-Kugel mit dem Radius 3 umfasst, folgendermaßen beziffern:

$$\binom{23}{0} + \binom{23}{1} + \binom{23}{2} + \binom{23}{3} = 2048$$

Die Anzahl ist eine Zweierpotenz ( $2^{11}$ ), und das bedeutet nach Satz 7.2, dass es einen perfekten 3-fehlerkorrigierenden Code der Länge 23 geben könnte. Ein solcher Code existiert tatsächlich! Er wurde von dem Schweizer Elektrotechniker Marcel Jules Edouard Golay entdeckt und im Jahr 1949 in den *Proceedings of the Institute of Radio Engineers* veröffentlicht. Golays Artikel ist kaum länger als eine drei viertel Seite und gilt dennoch als eine der wichtigsten Arbeiten auf dem Gebiet der Codierungstheorie. Der von ihm entdeckte Code ist von so hoher mathematischer Schönheit, dass in den Folgejahren mehrere Querbezüge zu anderen algebraischen Strukturen gefunden wurden.

Abbildung 7.4 zeigt einen Auszug aus Golays Originalarbeit. Auf der linken Seite ist der relevante Teil der Kontrollmatrix zu sehen, mit der Golay seinen Code definiert. Jedes Codewort besteht aus insgesamt 23 Bits, die sich aus 12 Nachrichtenbits und 11 Prüfbits zusammensetzen. Um die vollständige Kontrollmatrix zu erhalten, muss Golays Matrix noch um die Einheitsmatrix ergänzt werden. Wird diese rechts hinzugefügt, so entsteht die Kontrollmatrix, die in der linken Hälfte in Abbildung 7.5 zu sehen ist. Sie beschreibt eine systematische Codierung,

Kontrollmatrix	Generatormatrix
$\begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$

Abb. 7.5: Kontroll- und Generatormatrix des [23,12]-Golay-Codes aus der Originalarbeit von 1949

deren Codewörter mit den Nachrichtenbits beginnen und mit den Prüfbits enden.

Aus der Kontrollmatrix können wir mithilfe der Sätze 6.7 und 6.8 die Code-Distanz ableiten. Benennen wir die Spalten der Kontrollmatrix mit  $S_1, \dots, S_{23}$ , so ergibt die Addition von  $S_{12}, S_{13}, S_{14}, S_{18}, S_{21}, S_{22}$  und  $S_{23}$  den Nullvektor:

$$\begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \mathbf{0}$$

Das bedeutet, dass die Kontrollmatrix 7 linear abhängige Spalten besitzt, und die Code-Distanz, nach Satz 6.8, nicht größer als 7 sein kann. Greifen wir 6 beliebige Spalten heraus, so sind diese stets linear unabhängig, es ist also nicht möglich, mit 6 oder weniger Spalten den Nullvektor darzustellen. Das bedeutet nach Satz 6.7, dass die Code-Distanz größer als 6 sein muss. Damit ist gezeigt, dass der Golay-Code eine Code-Distanz von 7 aufweist und 3-fehlerkorrigierend ist.

Aus der Kontrollmatrix können wir unmittelbar die Generatormatrix ableiten. Hierfür müssen wir lediglich die Zeilen aus Golays Originaltafel als Spalten aufschreiben und links die Einheitsmatrix hinzufügen. Das Ergebnis ist in der rechten Hälfte in Abbildung 7.5 zu sehen.

### 7.3.1 Zyklischer Golay-Code

Wir wollen nach anderen Codeeigenschaften suchen und uns zunächst die Frage stellen, ob der Golay-Code zyklisch ist. Die Antwort lautet Nein. Beispielsweise ist

$$00000000000111000100111$$

ein Codewort, die zyklische Verschiebung

$$000000000001110001001110$$

hingegen nicht. Wir wollen versuchen, den Golay-Code in einen äquivalenten zyklischen Code zu überführen. In Abschnitt 6.4.3 haben wir uns ausführlich mit der Klasse der zyklischen Codes beschäftigt und sind nun in der Lage, unser dort erworbene Wissen ganz praktisch anzuwenden. Wir wissen, dass ein zyklischer  $[n, k]$ -Code von einem Generatorpolynom  $g(x)$  mit den folgenden Eigenschaften erzeugt wird:

- $\deg g(x) = n - k$
- $g(x)$  teilt  $x^n + 1$  ohne Rest

Faktorisieren wir das Polynom  $x^{23} + 1$ , so erhalten wir folgendes ermutigendes Ergebnis:

$$x^{23} + 1 = g_1(x) \cdot g_2(x) \cdot g_3(x) \quad \text{mit}$$

$$\begin{aligned} g_1(x) &= x^{11} + x^{10} + x^6 + x^5 + x^4 + x^2 + 1 \\ g_2(x) &= x^{11} + x^9 + x^7 + x^6 + x^5 + x + 1 \\ g_3(x) &= x + 1 \end{aligned}$$

Zwei der drei Faktorpolynome haben den passenden Grad und könnten einen zyklischen Golay-Code hervorbringen. Für die weiteren Untersuchungen benutzen wir zunächst das folgende Polynom:

$$g_1(x) = x^{11} + x^{10} + x^6 + x^5 + x^4 + x^2 + 1$$

Aus Abschnitt 6.4.3 wissen wir, dass sich die Generatormatrix eines zyklischen Codes direkt aus dessen Generatorpolynom ableiten lässt und wir hierfür, bezogen auf unser Beispiel, die Divisionsreste

$$x^{22} \bmod g(x)$$

$$x^{21} \bmod g(x)$$

...

$$x^{11} \bmod g(x)$$

ausrechnen müssen. Dividieren wir durch  $g_1(x)$ , so erhalten wir das folgende Ergebnis:

$$x^{22} \bmod g_1(x) = x^{10} + x^9 + x^5 + x^4 + x^3 + x \quad (\text{☞ } 11000111010)$$

$$x^{21} \bmod g_1(x) = x^9 + x^8 + x^4 + x^3 + x^2 + 1 \quad (\text{☞ } 01100011101)$$

$$x^{20} \bmod g_1(x) = x^{10} + x^9 + x^8 + x^7 + x^5 + x^4 + x^2 \quad (\text{☞ } 11110110100)$$

$$x^{19} \bmod g_1(x) = x^9 + x^8 + x^7 + x^6 + x^4 + x^3 + x \quad (\text{☞ } 01111011010)$$

$$x^{18} \bmod g_1(x) = x^8 + x^7 + x^6 + x^5 + x^3 + x^2 + 1 \quad (\text{☞ } 00111101101)$$

$$x^{17} \bmod g_1(x) = x^{10} + x^9 + x^7 + x^6 + x^3 + x^2 \quad (\text{☞ } 11011001100)$$

$$x^{16} \bmod g_1(x) = x^9 + x^8 + x^6 + x^5 + x^2 + x \quad (\text{☞ } 01101100110)$$

$$x^{15} \bmod g_1(x) = x^8 + x^7 + x^5 + x^4 + x + 1 \quad (\text{☞ } 00110110011)$$

$$x^{14} \bmod g_1(x) = x^{10} + x^9 + x^7 + x^6 + x^5 + x + 1 \quad (\text{☞ } 11011100011)$$

$$x^{13} \bmod g_1(x) = x^{10} + x^8 + x^6 + x^3 + x + 1 \quad (\text{☞ } 10101001011)$$

$$x^{12} \bmod g_1(x) = x^{10} + x^7 + x^4 + x^3 + x^2 + x + 1 \quad (\text{☞ } 10010011111)$$

$$x^{11} \bmod g_1(x) = x^{10} + x^6 + x^5 + x^4 + x^2 + 1 \quad (\text{☞ } 10001110101)$$

Schreiben wir die Divisionsreste zeilenweise auf und fügen von links die Einheitsmatrix hinzu, so entsteht die Generatormatrix, die auf der rechten Seite in Abbildung 7.6 zu sehen ist.

Mit einer ähnlichen Analyse wie oben lässt sich zeigen, dass dieser Code tatsächlich eine Distanz von 7 aufweist, genau wie der ursprüngliche Golay-Code. Damit sind wir am Ziel: Wir haben einen zyklischen Golay-Codes gefunden.

Bevor wir fortfahren, wollen wir uns zunächst noch ansehen, welchen Code das Polynom

$$g_2(x) = x^{11} + x^9 + x^7 + x^6 + x^5 + x + 1$$

hervorbringt, das genau wie  $g_1(x)$  ein Faktor von  $x^{23} + 1$  ist. Um die Generatormatrix zu erhalten, berechnen wir, wie oben, zunächst die Reste,

**Abb. 7.6:** Kontroll- und Generatormatrix des zyklischen [23,12]-Golay-Codes, generiert durch das Polynom  $g_1(x) = x^{11} + x^{10} + x^6 + x^5 + x^4 + x^2 + 1$

die sich aus der Division von  $x^{22}$  bis  $x^{11}$  durch das Generatorpolynom ergeben. Wir erhalten:

$$x^{22} \bmod g_2(x) \equiv x^{10} + x^8 + x^6 + x^5 + x^4 + 1 \quad (\text{指向 } 10101110001)$$

$$x^{21} \bmod g_2(x) \equiv x^{10} + x^9 + x^8 + x^7 + x^6 + x^3 + 1 \quad (\text{👉 } 11111001001)$$

$$x^{20} \bmod g_2(x) \equiv x^{10} + x^9 + x^7 + x^4 + x^2 + 1 \quad (\text{指向 } 11010010101)$$

$$x^{19} \bmod g_2(x) = x^{10} + x^9 + x^5 + x^4 + x^3 + x + 1 \quad (\text{指向 } 11000111011)$$

$$r^{18} \bmod g_2(r) = r^{10} + r^9 + r^6 + r^5 + r^3 + r^2 \quad (\text{指向 } 11001101100)$$

$$x^{17} \bmod g_2(x) = x^9 + x^8 + x^5 + x^4 + x^2 + x \quad (\text{Hand } 01100110110)$$

$$x^{16} \bmod g_2(x) = x^8 + x^7 + x^4 + x^3 + x + 1 \quad (\text{Handwritten: } 00110011011)$$

$$x \mod g_2(x) = x + x + x + x + x + 1 \quad (\text{blue} \rightarrow 00110011011)$$

$$x \mod g_2(x) = x + x + x + x + x + x + x + x \quad (\text{blue } 1011011100)$$

$$x \mod g_2(x) = x + x + x + x + x + x + x + x \quad (\text{blue} \quad 01011011110)$$

$$x^{-1} \bmod g_2(x) \equiv x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \quad (\text{00101101111})$$

$$x^{17} \bmod g_2(x) \equiv x^{15} + x^5 + x^3 + x^2 + x^1 + x^0 \quad (\text{blue } 10111000110)$$

$$x^{11} \bmod g_2(x) = x^7 + x^5 + x^4 + x^3 + x + 1 \quad (\text{01011100011})$$

Aus den berechneten Ergebnissen können wir wieder die Generatorma-

Aus den berechneten Ergebnissen können wir wieder die Generatormatrix aufstellen. Sie ist, zusammen mit ihrer Kontrollmatrix, in Abbildung 7.7 zu sehen.

Was sich hinter diesem Code verbirgt, wird klar, wenn wir die Generatormatrix geringfügig umformen. Hierfür greifen wir auf zwei element-

Kontrollmatrix $H_{g_2(x)}$	Generatormatrix $G_{g_2(x)}$
$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$

Abb. 7.7: Kontroll- und Generatormatrix des zyklischen [23,12]-Golay-Codes, generiert durch das Polynom  $g_2(x) = x^{11} + x^9 + x^7 + x^6 + x^5 + x + 1$

tare Eigenschaften von Generatormatrizen zurück. Zum einen nutzen wir aus, dass eine Generatormatrix invariant in Bezug auf elementare Zeilenoperationen ist. Hieraus folgt im Besonderen, dass die Reihenfolge, in der die Zeilen einer Generatormatrix aufgelistet sind, keinen Einfluss auf die generierten Codewörter hat. Zum anderen nutzen wir aus, dass die Generatormatrix eines zyklischen Codes invariant in Bezug auf die Rotation ihrer Spalten ist. Das bedeutet, dass wir die Spalten beliebig nach links oder rechts rotieren können, ohne die Menge der Codewörter zu verändern.

Rotieren wir die Spalten der Generatormatrix  $G_{g_2(x)}$ , wie es in Abbildung 7.8 (oben) gezeigt ist, nach rechts und ordnen die Zeilen in umgekehrter Reihenfolge an, so entsteht eine Matrix, die zu der Generatormatrix  $G_{g_1(x)}$  spiegelsymmetrisch ist. Damit ist klar, welchen Code  $g_2(x)$  erzeugt. Er unterscheidet sich von unserem ersten zyklischen Code nur dadurch, dass die Bits innerhalb der Codewörter in umgekehrter Reihenfolge erscheinen.

Die Spiegelsymmetrie ist übrigens schon in den Generatorpolynomen enthalten. Kehren wir die Reihenfolge der Koeffizienten in  $g_1(x)$  um, so erhalten wir das Polynom  $g_2(x)$  und umgekehrt (Abbildung 7.8 unten).

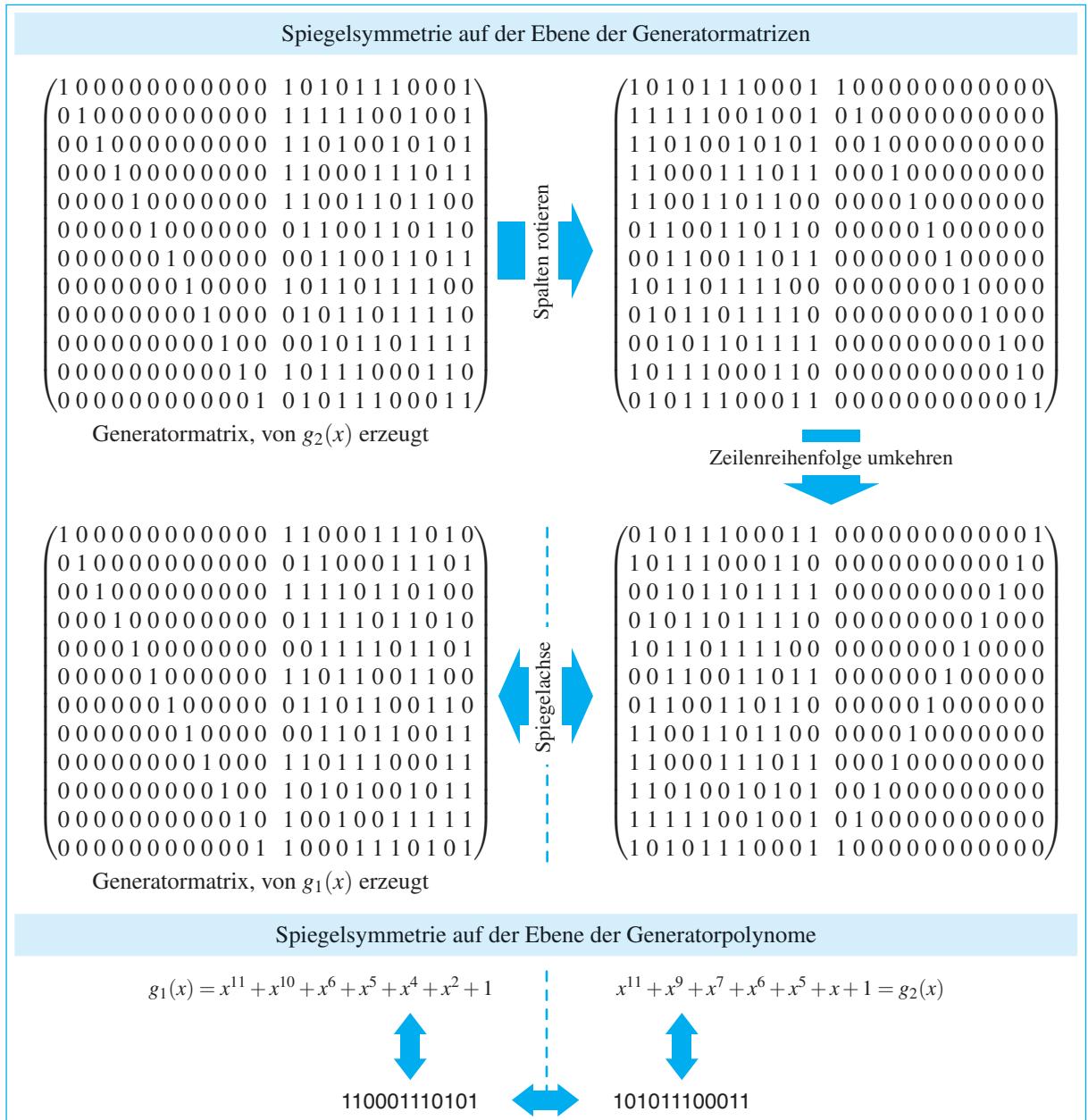


Abb. 7.8: Die Polynome  $g_1(x)$  und  $g_2(x)$  erzeugen zueinander spiegelsymmetrische Codes.

Kontrollmatrix	Generatormatrix
$\begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \textcolor{blue}{1} & \textcolor{blue}{1} \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \textcolor{blue}{0} & \textcolor{blue}{1} & \textcolor{blue}{1} & \textcolor{blue}{0} \end{pmatrix}$

**Abb. 7.9:** Durch die Hinzunahme eines Paritätsbits entsteht ein [24,12,8]-Code. Codes mit diesen Parametern werden als *erweiterter Golay-Codes* bezeichnet.

### 7.3.2 Erweiterter Golay-Code

Auf Seite 360 haben wir gezeigt, wie sich die Fehlererkennungseigenschaft von Codes mit einer ungeraden Code-Distanz verbessern lässt. Nach Satz 6.4 reicht die Hinzunahme eines Paritätsbits aus, um die Distanz um 1 zu erhöhen. Wenden wir dieses Prinzip auf den [23,12,7]-Golay-Code aus Abbildung 7.5 an, so entsteht ein linearer [24,12,8]-Code, der durch die Generatormatrix und die Kontrollmatrix in Abbildung 7.9 gegeben ist. In der Codierungstheorie werden lineare [24,12,8]-Codes als *erweiterter Golay-Codes* bezeichnet.

Ein historisch bedeutender [24,12,8]-Code wird durch die Generatormatrix in Abbildung 7.10 erzeugt [40]. Er wurde von den Raumsonden Voyager 1 und Voyager 2 der NASA verwendet, um übermittelte Bilddaten gegen Übertragungsfehler abzusichern (Abbildung 7.12).

Die beiden Voyager-Sonden wurden in den 70er-Jahren für die Erforschung der äußeren Planeten und des interstellaren Raums gebaut. Am 20. August 1977 wurde Voyager 2 gestartet; Voyager 1 folgte wenige Wochen später, am 5. September. Abbildung 7.11 zeigt, dass die beiden Sonden zunächst den Jupiter sowie den Saturn passierten und sich ihre Flugrouten danach trennten. Voyager 1 erreichte im Jahr 2004 die erste äußere Grenze des Sonnensystems, den sogenannten *Termination*

Kontrollmatrix	Generatormatrix
$\begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$

Abb. 7.10: Der erweiterte Golay-Code in der Variante, die von der NASA für die Raumsonden Voyager 1 und Voyager 2 ausgewählt wurde.

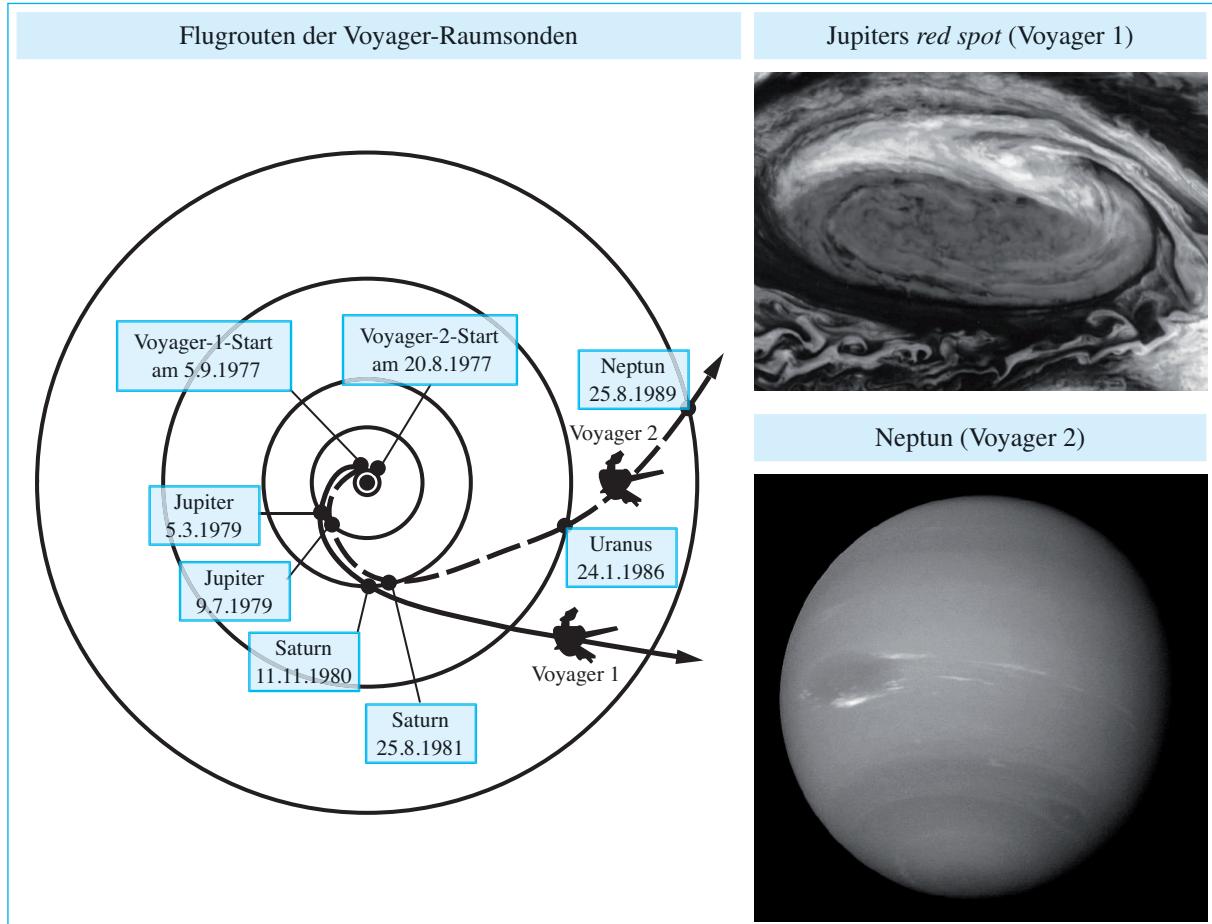
shock, und durchfliegt zurzeit den *Heliosheath*. Im Jahr 2015 wird sie die *Heliopause*, die äußere Grenze unseres Sonnensystems, erreichen.

Für Voyager 2 wurde eine Flugroute gewählt, die an zwei weiteren Planeten vorbeiführt. Am 24.1.1986 passierte die Sonde Uranus und am 25.8.1989 Neptun, den äußersten Planeten unseres Sonnensystems. Im Jahr 2007 erreichte Voyager 2 ebenfalls den Termination shock und befindet sich zurzeit, wie ihre Schwestersonde, im Heliosheath. Beide Sonden werden voraussichtlich bis zum Jahr 2025 wissenschaftliche Daten liefern. Danach werden sie als leblose Objekte weiter den interstellaren Raum durchqueren, einer unbestimmten Zukunft entgegen.



Abb. 7.12: Fotomontage der Voyager-Sonde (Bildquelle: NASA)

Rückblickend war die Voyager-Mission ein voller Erfolg. Die beiden Raumsonden haben Bilder von den äußeren Planeten geliefert, die alle davor gemachten Aufnahmen in ihrem Detaillierungsgrad um ein Vielfaches übertrafen. Aus der Sicht der Codierungstheorie ist besonders interessant, dass die Bilder, die Voyager 2 von Uranus und Neptun aufnahm, nicht mehr Golay-codiert übermittelt wurden. Um den Ausfall mehrerer technischer Komponenten zu kompensieren, wurde Voyager 2 von der Erde aus umprogrammiert und dabei unter anderem der Golay-Codierer durch einen moderneren Reed-Solomon-Codierer ausgetauscht. Dass ein Software-Update in diesem Umfang über eine so große Entfernung erfolgreich durchgeführt werden konnte, ist für sich gesehen ein spannendes Kapitel der Informatikgeschichte. Der Aufwand hatte sich wahrlich gelohnt: Wir halten heute detaillierte Aufnah-



**Abb. 7.11:** Flugrouten der Raumsonden Voyager 1 und Voyager 2 (Bildquelle: NASA)

men der beiden äußeren Planeten unseres Sonnensystems in Händen, die es ohne die gewagte Umprogrammierung nicht gegeben hätte.

### 7.3.3 Ternärer Golay-Code

Wir wollen an dieser Stelle erneut einen Blick auf den Auszug aus Golays Originalarbeit werfen, der in Abbildung 7.4 zu sehen ist. Neben der Matrix, die wir eben besprochen haben, ist dort eine zweite Matrix angegeben, die ebenfalls zu einem perfekten Code führt. Diesem liegt

Kontrollmatrix	Generatormatrix
$\begin{pmatrix} 1 & 1 & 1 & 2 & 2 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 2 & 1 & 0 & 2 & 0 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 1 & 2 & 0 & 0 & 1 & 0 & 0 \\ 1 & 2 & 0 & 1 & 2 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 2 & 2 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 2 & 2 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 2 & 1 & 0 & 2 \\ 0 & 0 & 0 & 1 & 0 & 0 & 2 & 1 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 1 & 0 & 2 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 2 & 2 & 1 & 1 \end{pmatrix}$

Abb. 7.13: Kontroll- und Generatormatrix des ternären [11,6]-Golay-Codes aus der Originalarbeit von 1964

das dreielementige Codealphabet

$$\Pi = \{0, 1, 2\}$$

zugrunde; es handelt sich also nicht um einen binären, sondern um einen *ternären Golay-Code*.

In Abbildung 7.13 ist die vollständige Kontrollmatrix zu sehen. Zusätzlich ist die Generatormatrix abgebildet, an deren Größe wir sofort die restlichen Codeparameter ablesen können. Die Matrix besteht aus 6 Zeilen und 11 Spalten, sodass der ternäre Golay-Code ein linearer [11,6]-Code ist. Die Code-Distanz lässt sich durch eine Analyse der Kontrollmatrix oder, technisch noch einfacher, durch eine Analyse der insgesamt 729 Codewörter ermitteln: Sie ist gleich 5 und der ternäre Golay-Code somit ein 2-fehlerkorrigierender Code.

Um rechnerisch zu überprüfen, dass der ternäre Golay-Code perfekt ist, müssen wir die Größe einer Hamming-Kugel mit dem Radius 2 berechnen. Nach (7.5) gilt:

$$\begin{aligned} |\mathbf{K}_2(\mathbf{v})| &= \sum_{i=0}^2 \binom{11}{i} 2^i \\ &= 1 + \binom{11}{1} \cdot 2 + \binom{11}{2} \cdot 4 \\ &= 1 + 22 + 220 \\ &= 243 \end{aligned}$$

Wegen  $n = 11$  und  $k = 6$  ist

$$|\mathbf{K}_2(\mathbf{v})| = 243 = 3^{11-6} = 3^{n-k}$$

Damit ist der ternäre Golay-Code im Sinne von Definition 7.2 perfekt.

## 7.4 Restfehlerwahrscheinlichkeit

In den vorangegangenen Abschnitten haben wir Kanalcodes mit sehr unterschiedlichen Code-Distanzen kennengelernt. Wir wissen, wie die Code-Distanz die Fähigkeit beeinflusst, Fehler zu erkennen bzw. zu korrigieren, und wir wissen auch, dass die Erhöhung der Distanz diese Fähigkeiten verbessert. Einen wichtigen Aspekt dürfen wir dabei nicht übersehen: Umfasst ein Code mindestens zwei Codewörter, und davon können wir in der Praxis immer ausgehen, so kann ein gesendetes Codewort auf der Übertragungsstrecke so ungünstig verfälscht werden, dass es zu einem anderen Codewort wird. In diesem Fall hat der Empfänger keine Möglichkeit, den Übertragungsfehler zu erkennen, geschweige denn, zu korrigieren. Die Frage nach der Häufigkeit solcher Ereignisse führt uns auf direktem Weg zum Begriff der *Restfehlerwahrscheinlichkeit*.

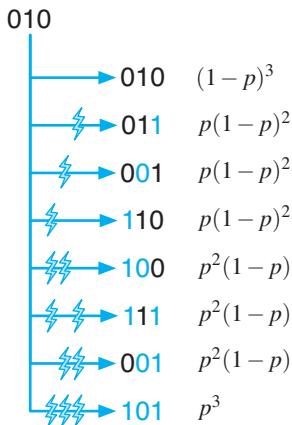
Behalten Sie stets im Gedächtnis, dass wir zwischen zwei verschiedenen Restfehlerwahrscheinlichkeiten unterscheiden müssen, je nachdem, ob wir die Fehlererkennungs- oder die Fehlerkorrektoreigenschaft eines Codes bewerten. In den folgenden Abschnitten werden wir beide Szenarien getrennt voneinander betrachten.

### 7.4.1 Restfehler bei der Fehlererkennung

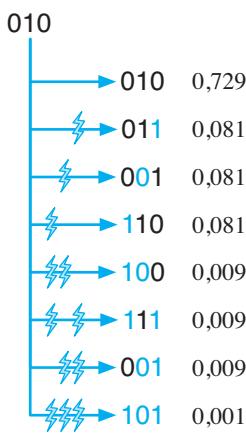
Die Restfehlerwahrscheinlichkeit eines *fehlererkennenden Codes* ist die Wahrscheinlichkeit, dass ein Codewort auf der Übertragungsstrecke verfälscht wurde und dieser Fehler auf der Empfängerseite nicht bemerkt werden kann. Um diesen Sachverhalt mathematisch zu modellieren, erinnern wir uns daran, wie der Empfänger einen Übertragungsfehler erkennt. Von der konkreten algorithmischen Implementierung abstrahiert, lässt sich das Vorgehen so beschreiben: Sobald eine Bitsequenz empfangen wird, prüft der Decoder, ob es sich dabei um ein Codewort handelt, und signalisiert genau dann einen Fehler, wenn dies nicht der Fall ist. Damit ist klar, wie sich die Restfehlerwahrscheinlichkeit formal charakterisieren lässt: Sie ist die Wahrscheinlichkeit, dass aus einem gesendeten Codewort durch das Kippen von einem oder mehreren Bits zufällig ein anderes Codewort entsteht.

In den Formeln, die wir nun entwickeln werden, bezeichnet  $p$  die Wahrscheinlichkeit, dass ein einzelnes Bit falsch übertragen wird, d. h., es wird eine gesendete 0 als 1 empfangen oder eine gesendete 1 als 0. Entsprechend ist  $(1 - p)$  die Wahrscheinlichkeit, dass ein einzelnes Bit

$$p^i(1-p)^{n-i} \text{ mit } n = 3$$



$$p^i(1-p)^{n-i} \text{ mit } n = 3 \text{ und } p = 0,1$$



**Abb. 7.14:** Mit der Wahrscheinlichkeit  $p$ , dass ein einzelnes Bit auf der Übertragungsstrecke verfälscht wird, lässt sich für jedes Bitpaket ausrechnen, mit welcher Wahrscheinlichkeit es den Empfänger erreicht.

korrekt den Empfänger erreicht. Wir nehmen in unserer Betrachtung an, dass die Übertragungsfehler unabhängig voneinander auftreten. Ob ein Bit verfälscht wird oder nicht, hängt also nicht von seiner Position ab. Genauso wenig spielt es eine Rolle, ob das Vorgängerbit verfälscht wurde oder nicht.

Legen wir dieses einfache Fehlermodell zugrunde, so ist die Wahrscheinlichkeit, dass ein Codewort der Länge  $n$  fehlerfrei übertragen wird, gleich

$$(1-p)^n$$

und die Wahrscheinlichkeit, dass von den  $n$  gesendeten Bits  $i$  verfälscht werden, gleich

$$p^i(1-p)^{n-i} \quad (7.7)$$

Abbildung 7.14 demonstriert diese Formel an einem konkreten Beispiel.

Mit  $p_{\text{rest}}(\mathbf{v})$  bezeichnen wir die Wahrscheinlichkeit, dass aus dem gesendeten Codewort  $\mathbf{v}$  durch das Kippen von einem oder mehreren Bits zufällig ein anderes Codewort entsteht.  $p_{\text{rest}}(\mathbf{v})$  lässt sich berechnen, indem die Empfangswahrscheinlichkeiten der Form (7.7) für alle Codewörter aufsummiert werden, die von  $\mathbf{v}$  verschieden sind:

$$p_{\text{rest}}(\mathbf{v}) = \sum_{\substack{\mathbf{v}' \in C \\ \mathbf{v} \neq \mathbf{v}'}} p^{\Delta(\mathbf{v}, \mathbf{v}')} (1-p)^{n-\Delta(\mathbf{v}, \mathbf{v}')} \quad (7.8)$$

Hierin ist  $\Delta(\mathbf{v}, \mathbf{v}')$  die Hamming-Distanz zwischen  $\mathbf{v}$  und  $\mathbf{v}'$ , d. h. die Anzahl der Positionen, an denen sich  $\mathbf{v}$  und  $\mathbf{v}'$  unterscheiden.

Um Formeln der Bauart (7.8) übersichtlich aufzuschreiben zu können, vereinbaren wir die folgende Abkürzung:

$$\beta_i^n := p^i(1-p)^{n-i}$$

Mit dieser liest sich (7.8) so:

$$p_{\text{rest}}(\mathbf{v}) = \sum_{\substack{\mathbf{v}' \in C \\ \mathbf{v} \neq \mathbf{v}'}} \beta_{\Delta(\mathbf{v}, \mathbf{v}')}^n \quad (7.9)$$

Unter der Annahme, dass alle Codewörter gleich häufig gesendet werden, können wir die Restfehlerwahrscheinlichkeit dann folgendermaßen beziffern:

$$\begin{aligned} p_{\text{rest}} &= \frac{1}{|C|} \sum_{\mathbf{v} \in C} p_{\text{rest}}(\mathbf{v}) \\ &= \frac{1}{|C|} \sum_{\mathbf{v} \in C} \sum_{\substack{\mathbf{v}' \in C \\ \mathbf{v} \neq \mathbf{v}'}} \beta_{\Delta(\mathbf{v}, \mathbf{v}')}^n \end{aligned} \quad (7.10)$$

Aus praktischer Sicht ist diese Formel reichlich unhandlich. Um die Restfehlerwahrscheinlichkeit auszurechnen, müssen wir für jedes Codewort die Distanzen zu allen anderen Codewörtern berechnen, mit der Fehlerwahrscheinlichkeit  $p$  verrechnen und aufsummieren; ein aufwendiges Unterfangen. Die Situation wendet sich zum Guten, wenn wir unsere Betrachtungen auf die wichtige Klasse der linearen Codes einschränken.

Um die gewünschte Vereinfachung zu erzielen, schreiben wir Formel (7.9) zunächst geringfügig um:

$$p_{\text{rest}}(\mathbf{v}) = \sum_{\substack{\mathbf{v} + \mathbf{e} \in C \\ \mathbf{e} \neq \mathbf{0}}} \beta_{\Delta(\mathbf{v}, \mathbf{v} + \mathbf{e})}^n \quad (7.11)$$

Im Unterschied zu Formel (7.9) wird das Codewort  $\mathbf{v}'$  nun als eine Summe der Form  $\mathbf{v} + \mathbf{e}$  ausgedrückt. Hierin ist  $\mathbf{e}$  der Fehlervektor, der die Veränderung des Codeworts auf der Übertragungsstrecke beschreibt und genau an jenen Bitpositionen eine 1 aufweist, an denen das ursprüngliche Codewort verfälscht wurde.

Die Distanz  $\Delta(\mathbf{v}, \mathbf{v} + \mathbf{e})$  ist mit dem Hamming-Gewicht von  $\mathbf{e}$  identisch,

$$\Delta(\mathbf{v}, \mathbf{v} + \mathbf{e}) = \omega(\mathbf{e}),$$

sodass wir (7.11) folgendermaßen umformulieren können:

$$p_{\text{rest}}(\mathbf{v}) = \sum_{\substack{\mathbf{v} + \mathbf{e} \in C \\ \mathbf{e} \neq \mathbf{0}}} \beta_{\omega(\mathbf{e})}^n \quad (7.12)$$

Als Nächstes greifen wir auf eine spezielle Eigenschaft linearer Codes zurück. Da die Codewörter einen Vektorraum bilden, ist die Addition eine abgeschlossene Operation, d. h., die Summe zweier Codewörter ist wiederum ein Codewort. Daraus folgt, dass alle Fehlervektoren  $\mathbf{e}$ , die wir in (7.12) berücksichtigen müssen, selbst Codewörter sind und umgekehrt jedes von  $\mathbf{0}$  verschiedene Codewort ein Fehlervektor sein kann. Folgerichtig können wir (7.12) auch so ausdrücken:

$$p_{\text{rest}}(\mathbf{v}) = \sum_{\substack{\mathbf{v}' \in C \\ \mathbf{v}' \neq \mathbf{0}}} \beta_{\omega(\mathbf{v}')}^n$$

Die erreichte Vereinfachung ist bedeutend: Sie zeigt, dass die Restfehlerwahrscheinlichkeit  $p_{\text{rest}}(\mathbf{v})$  für alle Codewörter  $\mathbf{v}$  gleich ist, und wir (7.10) deshalb folgendermaßen umschreiben können:

$$\begin{aligned} p_{\text{rest}} &= \frac{1}{|C|} \sum_{\mathbf{v} \in C} p_{\text{rest}}(\mathbf{v}) = \frac{1}{|C|} |C| p_{\text{rest}}(\mathbf{v}) = \sum_{\substack{\mathbf{v}' \in C \\ \mathbf{v}' \neq \mathbf{0}}} \beta_{\omega(\mathbf{v}')}^n \\ &= \sum_{\substack{\mathbf{v} \in C \\ \mathbf{v} \neq \mathbf{0}}} p^{\omega(\mathbf{v})} (1-p)^{n-\omega(\mathbf{v})} \end{aligned} \quad (7.13)$$

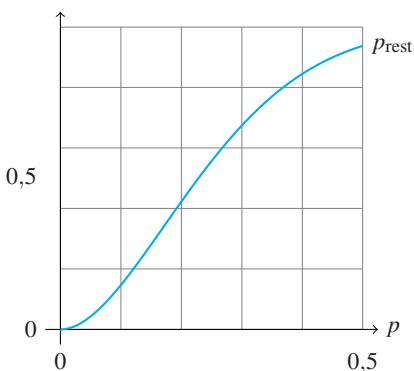
$\mathbf{v}$	
00000	
00011	$p^2(1-p)^3$
00101	$p^2(1-p)^3$
00110	$p^2(1-p)^3$
01001	$p^2(1-p)^3$
01010	$p^2(1-p)^3$
01100	$p^2(1-p)^3$
01111	$p^4(1-p)$
10001	$p^2(1-p)^3$
10010	$p^2(1-p)^3$
10100	$p^2(1-p)^3$
10111	$p^4(1-p)$
11000	$p^2(1-p)^3$
11011	$p^4(1-p)$
11101	$p^4(1-p)$
11110	$p^4(1-p)$

Restfehlerwahrscheinlichkeit  
↓

$$p_{\text{rest}} = 10p^2(1-p)^3 + 5p^4(1-p)$$

Abb. 7.15: Formel (7.13), angewendet auf den geraden Paritätscode der Länge 5

[5,4]-Paritätscode



**Abb. 7.16:** Restfehlerwahrscheinlichkeit des geraden ParitätsCodes der Länge 5

Im Falle von linearen Codes geht die Berechnung der Restfehlerwahrscheinlichkeit also wesentlich leichter von der Hand. Wir müssen lediglich für jedes von 0 verschiedene Codewort die Anzahl der Einsen zählen – diese ist das Hamming-Gewicht  $\omega(v)$  – und in Formel (7.13) einsetzen.

Abbildung 7.15 demonstriert die Berechnung am Beispiel des geraden ParitätsCodes der Länge 5. Als Ergebnis erhalten wir die Formel

$$p_{\text{rest}} = 10p^2(1-p)^3 + 5p^4(1-p), \quad (7.14)$$

in der die Bitfehlerwahrscheinlichkeit  $p$  als freie Variable übrig bleibt. Werden bei der Übertragung beispielsweise 1% aller Bits falsch übertragen, ist also

$$p = 0,01,$$

so beträgt die Restfehlerwahrscheinlichkeit

$$p_{\text{rest}} = 10 \cdot 0,01^2 \cdot 0,99^3 + 5 \cdot 0,01^4 \cdot 0,99 = 0,000970348$$

Das bedeutet, in grober Näherung, dass von 1000 übertragenen Codewörtern eines auf der Übertragungsstrecke so verfälscht wird, dass der Empfänger den Fehler nicht erkennt.

Abbildung 7.16 zeigt, wie sich die Restfehlerwahrscheinlichkeit (7.14) für steigende Werte von  $p$  entwickelt. Achten Sie darauf, die Grenzfälle richtig zu interpretieren. Ist  $p = 0$ , so erreichen alle Bits den Empfänger fehlerfrei. In diesem Fall wird kein einziges Bit verfälscht und die Restfehlerwahrscheinlichkeit ist 0. Ist

$$p = 0,5,$$

so werden 50 % alle Bits verfälscht. Dieser Wert ist nur theoretisch interessant, da der Empfänger dann eine perfekte Zufallssequenz empfängt. In diesem Fall bleibt ein Übertragungsfehler genau dann unentdeckt, wenn das gesendete Codewort  $v$  zufällig zu einem Codewort  $v'$  mit  $v \neq v'$  wird. Von den 16 möglichen Codewörtern des ParitätsCodes sind 15 von  $v$  verschieden, und da insgesamt 32 Bitsequenzen der Länge 5 existieren, muss die Restfehlerwahrscheinlichkeit  $\frac{15}{32}$  betragen. Genau diesen Wert erhalten wir, wenn der Wert  $p = 0,5$  in die oben hergeleitete Formel eingesetzt wird:

$$p_{\text{rest}} = 10 \cdot 0,5^2 \cdot 0,5^3 + 5 \cdot 0,5^4 \cdot 0,5 = 15 \cdot 0,5^5 = \frac{15}{2^5} = \frac{15}{32}$$

## 7.4.2 Restfehler bei der Fehlerkorrektur

Die Restfehlerwahrscheinlichkeit eines *fehlerkorrigierenden Codes* ist die Wahrscheinlichkeit, dass ein Codewort auf der Übertragungsstrecke verfälscht und durch den Korrekturalgorithmus des Empfängers fehlerhaft rekonstruiert wird. Für die Berechnung nehmen wir an, dass ein *Maximum-Likelihood-Decoder* zum Einsatz kommt, der für ein gesendetes Codewort  $v$  genau dann das richtige Ergebnis liefert, wenn die empfangene Bitsequenz innerhalb der von  $v$  aufgespannten Hamming-Kugel liegt. Ist der verwendete Code  $r$ -fehlerkorrigierend, so enthält die Hamming-Kugel  $K_r(v)$  sämtliche Bitsequenzen, die zu  $v$  eine Distanz kleiner oder gleich  $r$  aufweisen:

$$K_r(v) := \{w \in C \mid \Delta(v, w) \leq r\}$$

Aus Abschnitt 7.2.3 wissen wir, wie viele Bitsequenzen sich in einer Hamming-Kugel mit dem Radius  $r$  befinden. Bezeichnet  $n$  die Länge der Codewörter, so ist

$$|K_r(v)| = \sum_{i=0}^r \binom{n}{i}$$

Wie bisher bezeichnen wir mit  $p$  die Wahrscheinlichkeit, dass ein einzelnes Bit bei der Übertragung verfälscht wird, und mit  $(1 - p)$  die Wahrscheinlichkeit, dass ein einzelnes Bit korrekt den Empfänger erreicht.

Ist  $v$  das gesendete Codewort, so lässt sich die Wahrscheinlichkeit, dass eine Bitsequenz empfangen wird, die innerhalb der Hamming-Kugel  $K_r(v)$  liegt, folgendermaßen berechnen:

$$\sum_{i=0}^r \binom{n}{i} p^i (1-p)^{n-i}$$

Die Restfehlerwahrscheinlichkeit ist die Wahrscheinlichkeit, dass eine Bitsequenz außerhalb der Hamming-Kugel des gesendeten Codeworts empfangen wird. Nach dem eben Gesagten ergibt sich für sie die nachstehende Formel:

$$p_{\text{rest}} = 1 - \sum_{i=0}^r \binom{n}{i} p^i (1-p)^{n-i} \quad (7.15)$$

Für 1- und 2-fehlerkorrigierende Codes erhalten wir hieraus die folgenden spezialisierten Formeln:



Haben wir einen nicht perfekten Code vor uns, so lässt sich die in (7.15) angegebene Restfehlerwahrscheinlichkeit durch eine Modifikation des Decoders geringfügig unterbieten. Verantwortlich hierfür ist die Tatsache, dass wir jede empfangene Bitsequenz, die sich keiner Hamming-Kugel zuordnen lässt, als nicht korrigierbar angesehen haben. Modifizieren wir den Decoder so, dass er solche Bitsequenzen auf ein zufällig gewähltes Codewort abbildet, so werden wir in vereinzelten Fällen Glück haben und das richtige Codewort treffen. Dies führt dazu, dass die Restfehlerwahrscheinlichkeit geringfügig sinkt. Zu viel dürfen wir von dieser Optimierung natürlich nicht erwarten, da sie ausschließlich im Falle eines Zufallstreffers eine Verbesserung bringt. Auch bei nicht perfekten Codes wird die wahre Restfehlerwahrscheinlichkeit durch die Formel, die wir in diesem Abschnitt entwickelt haben, sehr präzise approximiert.

[10,2]-Wiederholungscode

$u$	$c(u)$
00	0000000000
01	0101010101
10	1010101010
11	1111111111

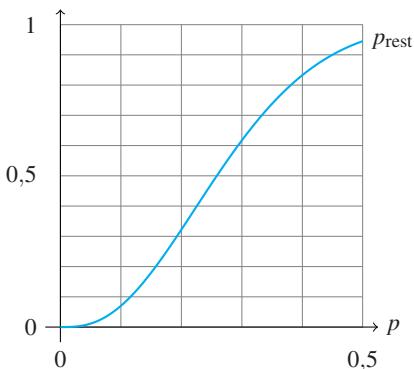


Abb. 7.17: Restfehlerwahrscheinlichkeit des [10,2]-Wiederholungscodes

■ 1-fehlerkorrigierende Codes

(👉  $r = 1$ )

$$p_{\text{rest}} = 1 - (1-p)^n - np(1-p)^{n-1} \quad (7.16)$$

■ 2-fehlerkorrigierende Codes

(👉  $r = 2$ )

$$\begin{aligned} p_{\text{rest}} &= 1 - (1-p)^n - np(1-p)^{n-1} \\ &\quad - \frac{n(n-1)}{2} p^2 (1-p)^{n-2} \end{aligned}$$

Als Beispiel betrachten wir den [10,2]-Wiederholungscode aus Abbildung 7.17. Alle Codewörter haben die Länge  $n = 10$  und die Code-Distanz beträgt 5. Die Hamming-Kugeln haben den Radius  $r = 2$ , sodass wir die Restfehlerwahrscheinlichkeit folgendermaßen berechnen können:

$$p_{\text{rest}} = 1 - (1-p)^{10} - 10p(1-p)^9 - 45p^2(1-p)^8 \quad (7.17)$$

Das in Abbildung 7.17 dargestellte Diagramm zeigt, wie sich die Restfehlerwahrscheinlichkeit zwischen  $p = 0$  und  $p = 0,5$  entwickelt. Für  $p = 0$  ist sie exakt 0, da in diesem Fall keine Übertragungsfehler auftreten. Bei  $p = 0,5$  erreicht den Empfänger ein perfektes Rauschen, und die Übertragung gelingt nur, wenn das ankommende Bitmuster zufällig innerhalb der korrekten Hamming-Kugel liegt. Von den insgesamt  $2^{10}$  möglichen Bitmustern führen dann 56 zu dem richtigen Ergebnis, sodass die Fehlerwahrscheinlichkeit

$$p_{\text{rest}} = 1 - \frac{56}{1024} = \frac{968}{1024}$$

betragen muss. Setzen wir den Wert  $p = 0,5$  in Gleichung (7.17) ein, so erhalten wir exakt diesen Wert:

$$\begin{aligned} p_{\text{rest}} &= 1 - (1-0,5)^{10} - 10 \cdot 0,5 \cdot (1-0,5)^9 - 45 \cdot 0,5^2 \cdot (1-0,5)^8 \\ &= 1 - 0,5^{10} - 10 \cdot 0,5^{10} - 45 \cdot 0,5^{10} = \frac{968}{1024} \end{aligned}$$

Als Nächstes Beispiel betrachten wir den [7,4]-Hamming-Code. Dieser Code ist 1-fehlerkorrigierend, sodass wir seine Restfehlerwahrscheinlichkeit über die Formel (7.16) berechnen können. Setzen wir den Wert  $n = 7$  ein, so ergibt dies die folgende Beziehung:

$$p_{\text{rest}} = 1 - (1-p)^7 - 7p(1-p)^6 \quad (7.18)$$

Werden bei der Übertragung 1% aller Bits verfälscht, ist also  $p = 0,01$ , so erhalten wir

$$p_{\text{rest}} = 1 - 0,99^7 - 7 \cdot 0,01 \cdot 0,99^6 \approx 0,002$$

als Ergebnis. Das bedeutet, dass von 500 gesendeten Codewörtern auf der Empfängerseite ungefähr eines fehlerhaft korrigiert wird. Der komplette Werteverlauf der Funktion (7.18) ist in Abbildung 7.18 zu sehen. Auch hier können wir die Funktionswerte für die Grenzfälle  $p = 0$  und  $p = 0,5$  ohne Kenntnis der Formel vorhersagen. Für  $p = 0$  ist die Restfehlerwahrscheinlichkeit, wie immer, gleich 0 und für  $p = 0,5$  ist die Wahrscheinlichkeit, dass ein gesendetes Codewort  $v$  nicht korrekt rekonstruiert wird, gleich der Wahrscheinlichkeit, dass eine zufällig erzeugte Sequenz außerhalb der Hamming-Kugel des gesendeten Codeworts liegt. Bei einer Codewortlänge von 7 Bit umfasst eine Hamming-Kugel mit dem Radius 1 genau 8 Bitvektoren, sodass sich die Restfehlerwahrscheinlichkeit wie folgt berechnet:

$$p_{\text{rest}} = 1 - \frac{8}{128} = \frac{15}{16}$$

Abschließend wollen wir einen Blick darauf werfen, wie gut die betrachteten Codes im Vergleich zu einer Übertragung abschneiden, bei der auf die Kanalcodierung komplett verzichtet und die Nachricht im Klartext übermittelt wird. Das bedeutet im Falle des Wiederholungscodes, dass anstelle der 10 Codewortbits ausschließlich die 2 Nachrichtenbits gesendet werden. Die Wahrscheinlichkeit  $p_{\text{error}}$ , dass der Empfänger die Nachricht nicht mehr fehlerfrei wiederherstellen kann, ist in diesem Fall mit der Wahrscheinlichkeit identisch, dass mindestens eines der beiden Nachrichtenbits falsch übertragen wird. Es ist dann

$$p_{\text{error}} = 1 - (1 - p)^2$$

Wie sich  $p_{\text{error}}$  im Vergleich zu der oben ermittelten Restfehlerwahrscheinlichkeit  $p_{\text{rest}}$  des Wiederholungscodes entwickelt, ist in Abbildung 7.19 (oben) aufgezeichnet. Für kleine Werte von  $p$  liegt  $p_{\text{rest}}$  deutlich unterhalb von  $p_{\text{error}}$ . Dieses Ergebnis ist kaum überraschend, schließlich ist es der Sinn und Zweck eines Kanalcodes, Übertragungsfehler zu kompensieren. Steigt die Fehlerwahrscheinlichkeit jedoch über ein gewisses Maß, so geschieht etwas Erstaunliches. Die Wahrscheinlichkeit, ein Datenpaket mithilfe des Wiederholungscodes fehlerfrei zu übertragen, ist auf einmal kleiner als die Wahrscheinlichkeit, die Nachricht im Klartext fehlerfrei zu übertragen.

Um den Grund zu erkennen, müssen wir uns daran erinnern, dass der Korrekturalgorithmus nur dann das gesendete Codewort  $v$  rekonstruieren kann, wenn eine Bitsequenz innerhalb der Hamming-Kugel  $K(v)$  empfangen wird. Im Übungsteil auf Seite 562 werden Sie zeigen, dass der [10,2]-Wiederholungscode die nachteilige Eigenschaft besitzt, dass die meisten Bitsequenzen zu keiner Hamming-Kugel gehören, sich die meisten Sequenzen also gewissermaßen im Niemandsland zwischen

[7,4]-Hamming-Code

$u$	$c(u)$
0000	0000000
0001	1101001
0010	0101010
0011	1000011
0100	1001100
0101	0100101
0110	1100110
0111	0001111
1000	1110000
1001	0011001
1010	1011010
1011	0110011
1100	0111100
1101	1010101
1110	0010110
1111	1111111

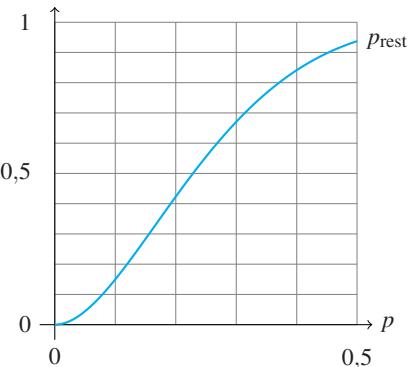
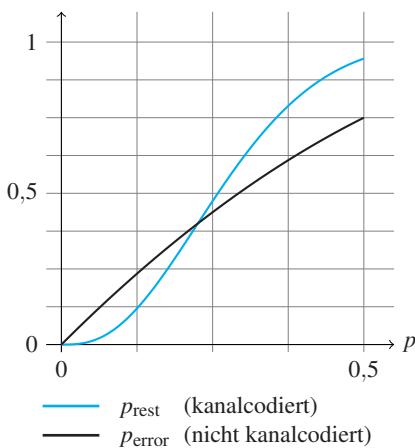
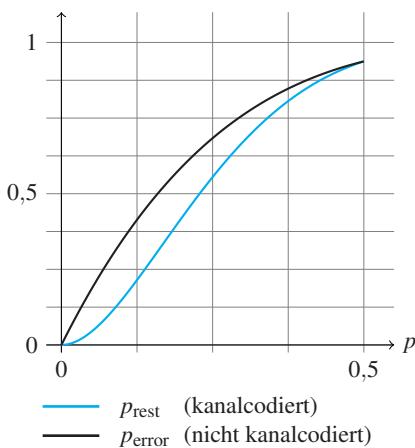


Abb. 7.18: Restfehlerwahrscheinlichkeit des [7,4]-Hamming-Codes

[10,2]-Wiederholungscode



[7,4]-Hamming-Code



**Abb. 7.19:** Der zusätzlich eingefügte Graph  $p_{\text{error}}$  beschreibt die Restfehlerwahrscheinlichkeit bei einer ungeschützten Übertragung.

den Kugeln befinden. Die langen Codewörter sorgen dafür, dass sehr schnell eine Bitsequenz entsteht, die keiner Hamming-Kugel zugeordnet ist und somit nicht korrigiert werden kann. Irgendwann wird die Wahrscheinlichkeit hierfür so groß, dass es tatsächlich vielversprechender ist, die Nachricht im Klartext zu übertragen.

Stellen wir die gleiche Überlegung für den [7,4]-Hamming-Code an, so erhalten wir die folgende Formel:

$$p_{\text{error}} = 1 - (1 - p)^4$$

Der Wert  $p_{\text{error}}$  ist die Wahrscheinlichkeit, dass von jeweils vier im Klartext gesendeten Nachrichtenbits mindestens eines fehlerhaft übertragen wurde. Abbildung 7.19 (unten) zeigt, dass die Situation jetzt eine bessere ist. Im Falle des [7,4]-Hamming-Codes übersteigt die Restfehlerwahrscheinlichkeit  $p_{\text{rest}}$  die Wahrscheinlichkeit  $p_{\text{error}}$  zu keinem Zeitpunkt. Der Grund hierfür ist schnell ausgemacht. Er geht auf die Eigenschaft der Hamming-Codes zurück, im Sinne von Definition 7.2 perfekt zu sein. Das bedeutet, dass sich jede Bitsequenz eindeutig einer Hamming-Kugel zuordnen lässt, d. h., die Kugeln liegen, bildlich gesprochen, lückenlos nebeneinander. Hierdurch erreicht der Code ein optimales Verhältnis zwischen der Codewortlänge auf der einen Seite und der Code-Distanz auf der anderen.

Auch die Tatsache, dass sich beide Graphen bei  $p = 0,5$  treffen, geht auf die Eigenschaft der Hamming-Codes zurück, perfekt zu sein. Wir wissen, dass bei einer Fehlerwahrscheinlichkeit von  $p = 0,5$  nur noch ein zufälliges Rauschen den Empfänger erreicht, und da die Hamming-Kugeln den Coderaum lückenlos ausfüllen, ist die Wahrscheinlichkeit, durch Zufall die richtige Hamming-Kugel zu treffen, gleich der Wahrscheinlichkeit, zufällig die richtige Nachricht zu erraten.

Alles in allem zeigen unsere Überlegungen, dass die Code-Distanz zwar ein wichtiges Merkmal für die Beurteilung der Praxistauglichkeit eines Codes ist, aber nicht die alleinige. Eine hohe Code-Distanz ist nur dann gewinnbringend, wenn sie nicht durch eine überbordende Verlängerung der Codewörter erreicht wurde. Das beste Verhältnis zwischen Code-Distanz und Codewortlänge wird von perfekten Codes erreicht. Das bedeutet, dass die Eigenschaft eines Codes, perfekt zu sein, mehr ist als eine Frage der mathematischen Ästhetik. Je stärker ein Code einem perfekten Code ähnelt, desto besser wird er sich bei steigenden Fehlerraten verhalten. Auf der anderen Seite wissen wir aus Abschnitt 7.2.3, dass perfekte Codes rar gesät sind. Zusammen unterstreicht dies erneut, warum z. B. die Hamming-Codes in der Informations- und Codierungstheorie eine so herausragende Rolle spielen.

## 7.5 Das Kanalcodierungstheorem

„What performance are you trying to achieve?  $10^{-15}$ ? You don't need sixty disk drives – you can get that performance with just two disk drives [...]. And if you want  $p_b = 10^{-18}$  or  $10^{-24}$  or anything, you can get there with two disk drives too!“

David J. C. MacKay [58]

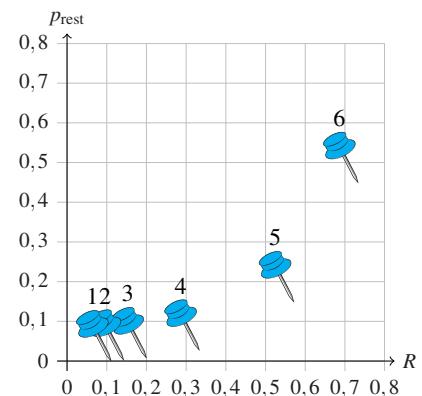
In diesem Abschnitt werden wir ein Ergebnis erarbeiten, das unbestritten zu den Perlen der Informations- und Codierungstheorie gehört. Gemeint ist das *Kanalcodierungstheorem* (*Noisy-channel coding theorem*), das in seiner Bedeutung auf der gleichen Stufe steht wie das Quellencodierungstheorem aus Abschnitt 5.3. Bewiesen wurde es von Claude de Shannon im Jahr 1948, in derselben Arbeit, in der wir auch die Herleitung des Quellencodierungstheorems finden [81]. Es ist eine seltene Ausnahme, dass in einer einzigen wissenschaftlichen Arbeit gleich zwei Theoreme mit solch weitreichender Bedeutung bewiesen wurden. Dies unterstreicht nicht nur, wie außergewöhnlich Shannons Meisterwerk ist; es wirft zugleich ein helles Licht auf Shannon selbst, dem wissenschaftliche Qualität zeitlebens wichtiger war als wissenschaftliche Quantität.

### 7.5.1 Inhaltliche Aussage

Womit befasst sich das Kanalcodierungstheorem? Blicken wir auf die Überlegungen aus den vorangegangenen Abschnitten zurück, so hat dort der Begriff der Restfehlerwahrscheinlichkeit eine immer wiederkehrende Rolle gespielt. Das Kanalcodierungstheorem stellt einen Zusammenhang zwischen der Restfehlerwahrscheinlichkeit und der *Coderate* eines Codes her. Letztere können wir uns als das Netto-Brutto-Verhältnis eines Kanalcodes vorstellen: Sie entspricht dem Verhältnis zwischen der Anzahl der Nachrichtenbits und der Anzahl der Bits, die ein Kanalcodierer daraus erzeugt. Für lineare  $[n, k]$ -Codes gilt also

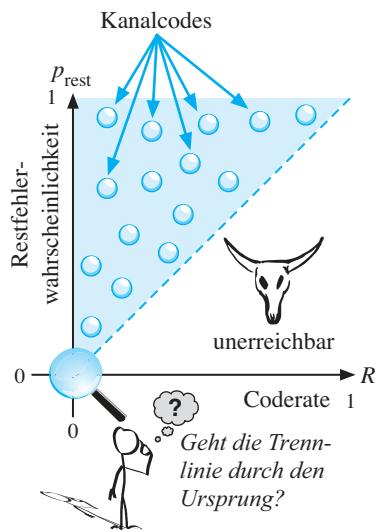
$$R = \frac{k}{n}$$

Der Begriff der Coderate lässt sich in naheliegender Weise auf beliebige Codierungen erweitern. Nimmt ein Kanalcodierer in  $T$  Zeiteinheiten  $K(T)$  Bits entgegen und erzeugt daraus  $N(T)$  Bits als Ausgabe, so können wir die Coderate als den Grenzwert definieren, dem der Quotient



1. [9,1]-Wiederholungscode
2. [7,1]-Wiederholungscode
3. [5,1]-Wiederholungscode
4. [3,1]-Wiederholungscode
5. [7,4]-Hamming-Code
6. [15,11]-Hamming-Code

**Abb. 7.20:** Die genannten Beispielcodes zeigen einen direkten Zusammenhang zwischen der Coderate  $R$  und der Restfehlerwahrscheinlichkeit  $p_{\text{rest}}$  auf. Die Codes mit der geringsten und der höchsten Coderate haben auch die geringste bzw. die höchste Restfehlerwahrscheinlichkeit.



**Abb. 7.21:** Jeder Punkt der zweidimensionalen Fläche entspricht einer bestimmten Kombination aus Coderate und Restfehlerwahrscheinlichkeit. Der Bereich rechts der Trennlinie ist nicht erreichbar. Dort ist die Coderate so hoch, dass kein Kanalcode die geforderte Restfehlerwahrscheinlichkeit erreichen kann.

dieser Größen für  $T \rightarrow \infty$  zustrebt:

$$R = \lim_{T \rightarrow \infty} \frac{K(T)}{N(T)}$$

Vergleichen wir typische Kanalcodes miteinander, so tritt ein wichtiger Zusammenhang hervor. Bei Codes, die mit einer geringeren Coderate übertragen, können wir eine tendenziell geringere Restfehlerwahrscheinlichkeit beobachten, und bei Codes, die eine höhere Coderate erreichen, müssen wir mit einer höheren Restfehlerwahrscheinlichkeit leben (Abbildung 7.20).

Dieser Zusammenhang ist alles andere als überraschend. Mit dem Wissen, das wir uns bis hierher angeeignet haben, ist klar, dass wir die Fehlerwahrscheinlichkeit auf ein beliebig kleines Maß verringern können, wenn wir dem Kanalcodierer erlauben, beliebig viele Prüfbits zu erzeugen. Ein Code der solches leistet, ist der Wiederholungscode. Die sukzessive Erhöhung der Wiederholungen führt dazu, dass die Restfehlerwahrscheinlichkeit kontinuierlich sinkt.

Shannon beschreibt dies in seiner Originalarbeit so:

*„It is clear, however, that by sending the information in a redundant form the probability of errors can be reduced. For example, by repeating the message many times and by a statistical study of the different received versions of the message the probability of errors could be made very small.“*

Claude Shannon [82]

Ordnen wir die Coderate und die Fehlerwahrscheinlichkeit, genau wie wir es in Abbildung 7.20 getan haben, zweidimensional an, so entsteht eine Grafik, deren schematischer Aufbau in Abbildung 7.21 zu sehen ist. Jeder Punkt symbolisiert einen Kanalcode, dessen  $x$ -Position durch die Coderate und dessen  $y$ -Position durch die Restfehlerwahrscheinlichkeit definiert wird. Die eingezeichnete Trennlinie markiert die Grenze zwischen zwei wichtigen Regionen. Oberhalb der Trennlinie befinden sich jene Kombinationen, die durch einen realen Code erreicht werden können. Die Kombinationen unterhalb der Trennlinie sind dagegen unerreichbar. In dieser Region ist die Coderate so hoch, dass zu wenige Prüfbits vorhanden sind, um die geforderte Restfehlerwahrscheinlichkeit zu erreichen.

Shannons nächster Satz ist gleichsam bedeutend:

„One would expect, however, that to make this probability of errors approach zero, the redundancy of the encoding must increase indefinitely, and the rate of transmission therefore approach zero.“

Was Shannon hier beschreibt, scheint auf den ersten Blick die einzige vernünftige Schlussfolgerung zu sein. Es ist klar, dass eine niedrige Restfehlerwahrscheinlichkeit nicht umsonst zu haben ist, und es sieht so aus, als müssten wir den Preis dafür in einer stetig wachsenden Anzahl an Prüfbits zahlen. Mit jedem zusätzlichen Prüfbit sinkt die Coderate, sodass die Restfehlerwahrscheinlichkeit nur dann gegen 0 streben kann, wenn die Coderate gleichermaßen gegen 0 strebt. Doch dürfen wir unserer Intuition, die die Trennlinie vor unserem geistigen Auge im Koordinatenursprung enden lässt, tatsächlich trauen? Ist es wirklich wahr, dass die sukzessive Absenkung der Fehlerwahrscheinlichkeit mit einer kontinuierlichen Verringerung der Coderate einhergehen muss?

Shannons Antwort kommt prompt:

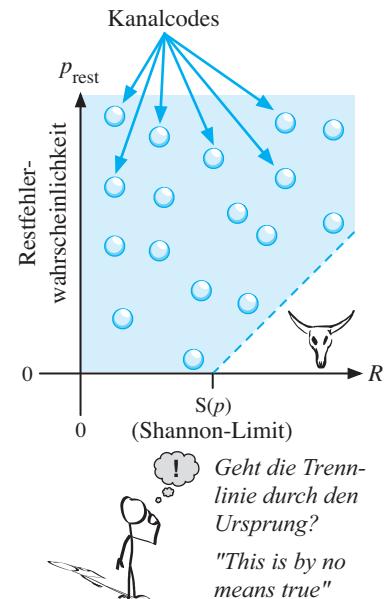
„This is by no means true.“

Abbildung 7.22 zeigt, was in der Nähe des Ursprungs wirklich geschieht. Was Sie dort sehen, ist eine bildliche Darstellung des Kanalcodierungstheorems. Anders als es unsere Intuition suggeriert, schneidet die Trennlinie die  $x$ -Achse keinesfalls im Koordinatenursprung, sondern in einem Punkt, dessen Abszisse fast immer größer als 0 ist. In der Literatur wird dieser Wert gerne als die *Kanalkapazität* bezeichnet. Wir nennen diesen Wert das *Shannon-Limit*, um ihn von der Kanalkapazität, die wir in Abschnitt 3.5.1 definiert haben, sauber zu trennen.

Wir wollen an dieser Stelle etwas vorgreifen und angeben, wie sich das Shannon-Limit  $S(p)$  berechnen lässt. Legen wir einen binären Kanal zugrunde, der ein eingespeistes Bit mit der Wahrscheinlichkeit  $p$  falsch überträgt, so gilt die folgende Formel:

$$S(p) = 1 - \left( p \log_2 \frac{1}{p} + (1-p) \log_2 \frac{1}{1-p} \right) \quad (7.19)$$

Das Shannon-Limit  $S(p)$  hängt lediglich von der Fehlerrate  $p$  des Übertragungskanals ab und ist für  $p \neq \frac{1}{2}$  von 0 verschieden. Dies hat erstaunliche Konsequenzen: Wählen wir eine Coderate  $R$  mit  $R < S(p)$ , so gibt es eine Kanalcodierung, die mit der gewählten Coderate überträgt und gleichzeitig eine beliebig vorgegebene Zuverlässigkeit erreicht. Die Restfehlerwahrscheinlichkeit zu senken bedeutet damit nicht zwangsläufig, die Coderate zu erhöhen. Bleiben wir mit der Coderate unterhalb



**Abb. 7.22:** Shannons Entdeckung ist fulminant. Er fand heraus, dass die Trennlinie die  $x$ -Achse in einem Punkt schneidet, dessen Abszisse im Allgemeinen rechts des Ursprungs liegt. Dies hat eine erstaunliche Konsequenz: Bleibt die Coderate unterhalb des *Shannon-Limits*, so ist die Übertragung mit einer beliebig geringen Fehlerwahrscheinlichkeit möglich.

des Shannon-Limits, so lässt sich die Restfehlerwahrscheinlichkeit alleine dadurch senken, dass eine geeigneteren Codierung gewählt wird. Ein faszinierendes Ergebnis!



### Satz 7.7 (Kanalcodierungstheorem, Shannon, 1948)

Gegeben sei ein binärer Übertragungskanal mit dem Shannon-Limit  $S(p)$ . Dann gilt für jedes  $\varepsilon > 0$ :

- Es existiert eine Kanalcodierung mit einer Coderate größer als  $S(p) - \varepsilon$  und einer Restfehlerwahrscheinlichkeit kleiner als  $\varepsilon$ .
- Mit Coderaten oberhalb von  $S(p)$  ist keine beliebig zuverlässige Übertragung mehr möglich.

Bevor wir den Beweis des Kanalcodierungstheorems skizzieren, wollen wir uns noch ein wenig konkreter mit dessen Aussage befassen. Nehmen wir an, dass ein Bit mit einer Fehlerwahrscheinlichkeit von  $p = 0,05$  falsch übertragen wird, dann erhalten wir für  $S(p)$  den folgenden Wert:

$$S(0,05) = 1 - \left( 0,05 \cdot \log_2 20 + (1 - 0,05) \cdot \log_2 \frac{1}{1 - 0,05} \right) \approx 0,7136$$

Der [7,4]-Hamming-Code weist eine Coderate auf, die mit  $\frac{4}{7}$  unterhalb des Shannon-Limits liegt. Mit dem Kanalcodierungstheorem hat Shannon gezeigt, dass eine Codierung existiert, die mit der gleichen Coderate wie der [7,4]-Hamming-Code arbeitet, aber beispielsweise nur die halbe Restfehlerwahrscheinlichkeit aufweist. Mit einer anderen Codierung lässt sich die Restfehlerwahrscheinlichkeit sogar auf ein Viertel reduzieren, mit wieder einer anderen auf ein Achtel und so fort. Shannons Theorem garantiert, dass für jedes  $\varepsilon$ , egal wie klein wir diese Größe auch wählen, eine Codierung existiert, die bei gleichbleibender Coderate mit einer Restfehlerwahrscheinlichkeit  $< \varepsilon$  übertragen kann.

Für den [15,11]-Hamming-Code läuft die Argumentation dagegen ins Leere, da dessen Coderate mit  $R = \frac{11}{15}$  knapp oberhalb des Shannon-Limits liegt. Es ist zwar nicht ausgeschlossen, dass sich durch die Wahl einer anderen Codierung die Restfehlerwahrscheinlichkeit bei gleichbleibender Coderate weiter absenken lässt, eine beliebig tiefe Absenkung, wie es unterhalb des Shannon-Limits möglich ist, kann dagegen nicht gelingen. All dies ist eine unmittelbare Folge aus dem Kanalcodierungstheorem!

Wir wollen uns die Funktion  $S(p)$  noch etwas intensiver ansehen. Blicken wir auf den in Abbildung 7.23 dargestellten Werteverlauf, so wird schnell klar, dass wir die Funktion eigentlich schon kennen. Sie sieht aus wie die vertikale Spiegelung der Funktion, die auf Seite 320 in Abbildung 5.6 zu sehen ist. Und tatsächlich: Ein Blick auf die Formel (7.19) zeigt, dass wir  $S(p)$  in der Form

$$S(p) = 1 - H_2(p) \quad (7.20)$$

darstellen können, wobei  $H_2(p)$  die Entropiefunktion einer gedächtnislosen Bernoulli-Quelle ist.

Zwei Aspekte machen die Funktion  $S(p)$  besonders interessant. Zum einen weist sie bei  $p = \frac{1}{2}$  eine Nullstelle auf, und zum anderen beginnt die Kanalkapazität für  $p \geq \frac{1}{2}$  größer zu werden, obwohl die Fehlerwahrscheinlichkeit weiter steigt. Um das Verhalten der Funktion zu verstehen, wollen wir uns überlegen, wie sich die verschiedenen Werte von  $p$  auf die übertragenen Daten auswirken. Abbildung 7.24 demonstriert dies auf grafische Weise.

Eine Fehlerwahrscheinlichkeit von  $p = \frac{1}{2}$  bedeutet, dass im Schnitt jedes zweite Pixel einen falschen Wert annimmt. Den Empfänger erreicht in diesem Fall ein zufälliges Rauschen, das keinerlei Information in sich trägt. Über einen solchen – und nur über einen solchen – Kanal ist keine Informationsübertragung möglich.

Dass die Kanalkapazität für Fehlerwahrscheinlichkeiten größer als  $\frac{1}{2}$  wieder ansteigt, ist ebenfalls leicht zu erklären. Betrachten wir beispielsweise den extremen Fall  $p = 1$ , so wird ausnahmslos jedes Bit auf der Übertragungsstrecke verfälscht, d. h., jede 0 erreicht den Empfänger als eine 1 und jede 1 als eine 0. Um die gesendete Bitfolge eins zu eins zu rekonstruieren, bräuchte der Empfänger lediglich die ankommenden Bits zu invertieren, und damit wäre ein solcher Kanal genauso brauchbar wie ein Kanal mit  $p = 0$ . In der Realität kommen Übertragungskanäle mit  $p > \frac{1}{2}$  nicht vor und sind lediglich in mathematischer Hinsicht von Interesse.

## 7.5.2 Beweisskizze

Nachdem wir uns mit der inhaltlichen Aussage des Kanalcodierungstheorems vertraut gemacht haben, wollen wir dessen Beweis skizzieren. Der Ausgangspunkt für unsere Überlegungen ist das in Abbildung 7.25 dargestellte Übertragungsszenario. Unsere Betrachtung beginnt mit dem Datenstrom am Ausgang des Quellencodierers. Wir neh-

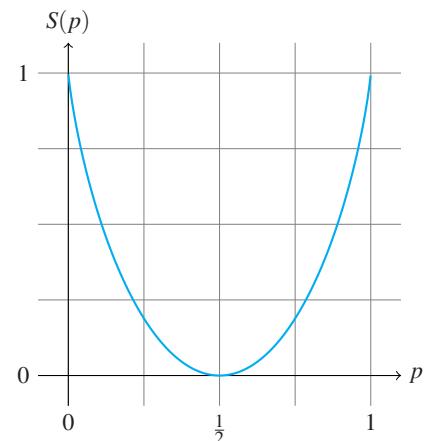
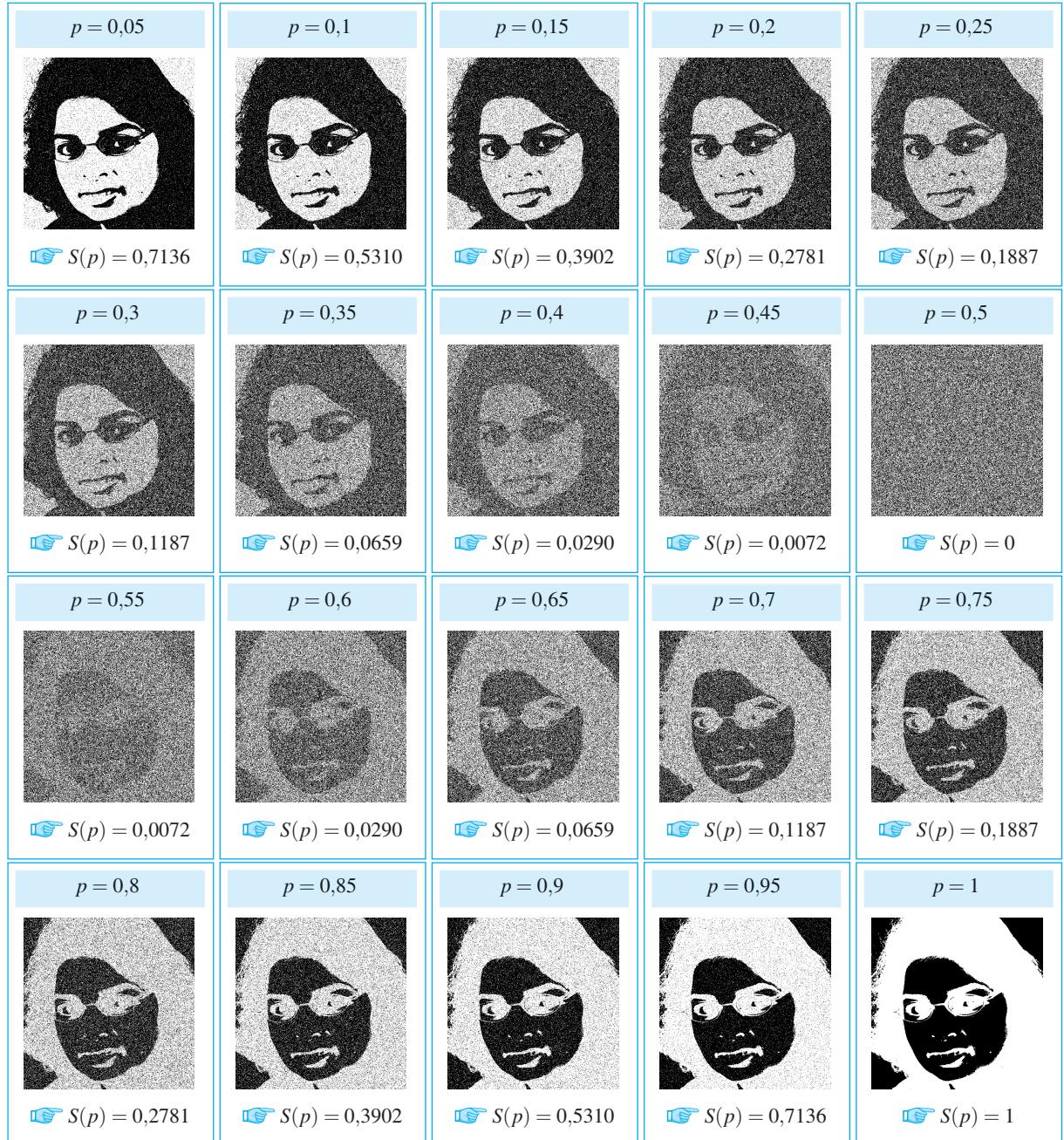
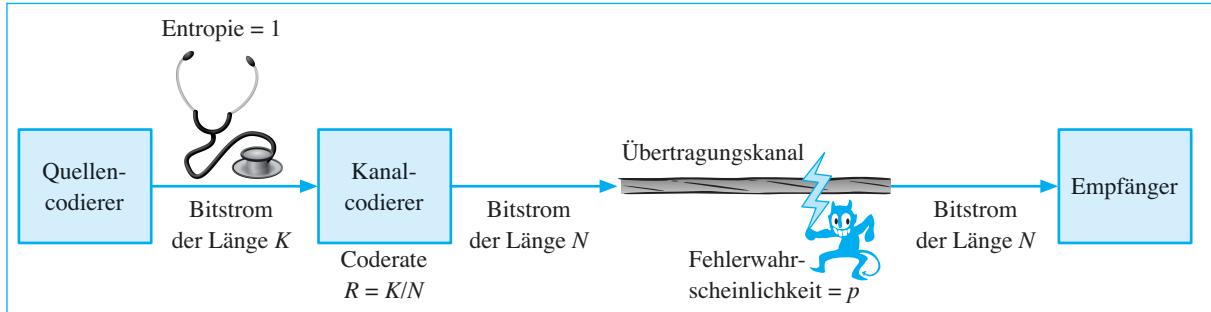


Abb. 7.23: Shannon-Limit eines Übertragungskanals mit der Bitfehlerwahrscheinlichkeit  $p$



**Abb. 7.24:** Das Shannon-Limit  $S(p)$ , berechnet für verschiedene verrauschte Kanäle. Die Kanäle mit einer Fehlerwahrscheinlichkeit  $p \geq \frac{1}{2}$  sind mathematische Spielereien und haben keine praktische Bedeutung.



**Abb. 7.25:** Ausgangsszenario für die Beweisskizze des Kanalcodierungstheorems

men an, dass eine optimale Kompression durchgeführt wird, die jegliche Redundanz entfernt. Hieraus folgt, dass wir am Ausgang des Quellencodierers einen Bitstrom mit der Entropie 1 beobachten können. In diesem Bitstrom sind genauso viele Nullen wie Einsen enthalten, und die Bits sind völlig zufällig verteilt.

Der Ausgang des Quellencodierers ist mit dem Eingang des Kanalcodierers verbunden. Werden über einen gewissen Zeitraum  $K$  Bits in den Kanalcodierer eingespeist, so gibt dieser in dem gleichen Zeitraum  $N$  Bits aus. Der Quotient

$$R = \frac{K}{N}$$

ist, über einen hinreichend langen Zeitraum betrachtet, die Coderate des eingesetzten Kanalcodes. Verwenden wir beispielsweise einen [7,4]-Hamming-Codierer, so werden jeweils 4 Bit des Eingabedatenstroms auf 7 Bit abgebildet; die Coderate beträgt dann  $\frac{4}{7}$ .

Um unsere Überlegungen an dieser Stelle so einfach wie möglich zu halten, nehmen wir an, dass der Übertragungskanal die Bits direkt transportieren kann, d. h., keine nochmalige Umcodierung erforderlich ist. In der Praxis ist dies nicht immer der Fall. Ein Beispiel für einen Übertragungskanal, der eine Umcodierung erfordert, ist der Morse-Kanal, mit dem wir uns ausführlich in Abschnitt 3.5.2 beschäftigt haben. Die binären Morse-Codewörter lassen sich über diesen Kanal nicht direkt übertragen und müssen zunächst in eine Folge von langen und kurzen Impulsen übersetzt werden. Anders als Shannon, der das Kanalcodierungstheorem in einer noch allgemeineren Form bewiesen hat, blenden wir diese Szenarien zum Zwecke der Vereinfachung aus.

Was wir nicht ausblenden dürfen, ist die Tatsache, dass manche Bits falsch übertragen werden. Die Wahrscheinlichkeit, dass ein Bit auf der

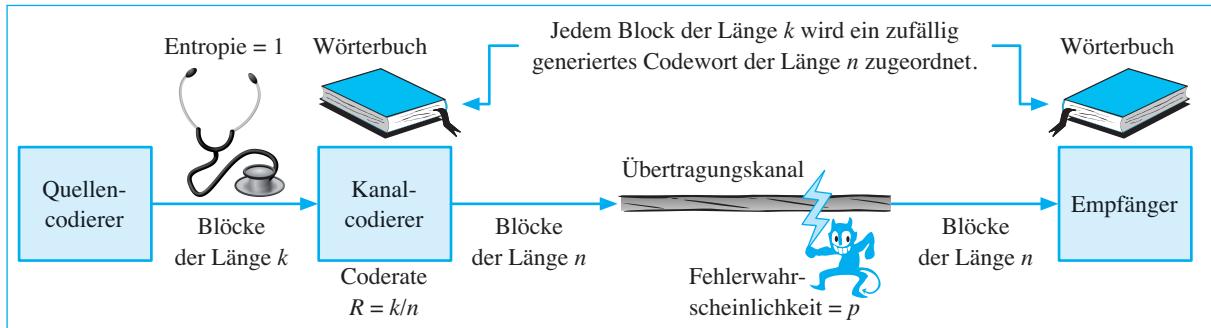


Abb. 7.26: Kanalcodierer mit einer zufallsgenerierten Codetabelle

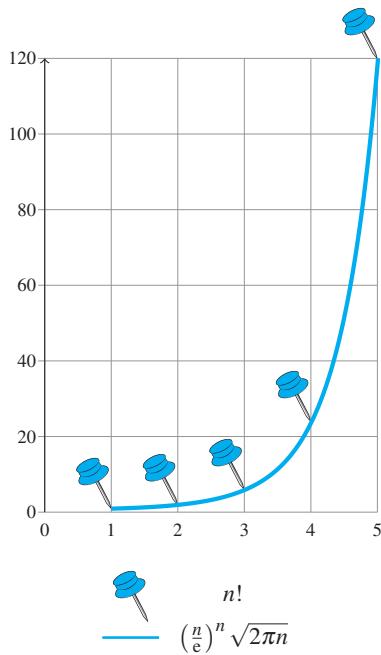


Abb. 7.27: Bei genauerer Betrachtung entpuppt sich die *Stirling-Formel* als ein faszinierendes mathematisches Gebilde. Sie kombiniert die Zahl  $n$  so geschickt mit der eulerschen Zahl  $e$  und der Kreiszahl  $\pi$ , dass der Funktionswert für wachsende Werte von  $n$  eine immer bessere Approximation für die Fakultät  $n!$  ergibt.

Übertragungsstrecke verfälscht wird, bezeichnen wir, wie bisher, mit  $p$ . Der Fall  $p = 0$  beschreibt demnach einen Kanal, der sämtliche Bits korrekt überträgt, und der Fall  $p = \frac{1}{2}$  einen Kanal, der an seinem Ausgang eine zufällige Bitsequenz erzeugt.

### Coderaten unterhalb des Shannon-Limits

Wir werden nun zeigen, dass mit Coderaten unterhalb des Shannon-Limits eine beliebig zuverlässige Nutzung des Übertragungskanals möglich ist. Hierzu betrachten wir eine Codierung, die zunächst trivial erscheint, aber für immer größere Blocklängen zu dem gewünschten Ergebnis führt. Wir benutzen einen Kanalcodierer, der den eingehenden Bitstrom in Blöcke der Länge  $k$  aufteilt und diese auf zufällig generierte Codewörter der Länge  $n$  abbildet. Die Zuordnung wird in einem Wörterbuch vermerkt, das vor der Übertragung an den Sender und den Empfänger verteilt wird (Abbildung 7.26).

Speisen wir gedanklich eine Bitsequenz der Länge  $n$  in den Kanal ein, so wird diese nur im Fall  $p = 0$  in ihrem ursprünglichen Zustand den Empfänger erreichen. In allen anderen Fällen müssen wir damit rechnen, dass in der empfangenen Sequenz einige Bits einen anderen Wert angenommen haben. Das Kanalrauschen führt also zu einer Streuung einer eingespeisten Sequenz: Für jede typische Sequenz, die sich am Eingang des Kanals beobachten lässt, gibt es eine Schar typischer Sequenzen, die wir am Ausgang des Kanals beobachten können.

Dies führt uns zu der Frage, auf wie viele typische Sequenzen eine eingespeiste Bitsequenz gestreut wird? Wird eine Bitsequenz  $v$  der Länge  $n$  gesendet, so wissen wir, dass die empfangene Sequenz  $w$  ungefähr  $np$

gekippte Bits enthalten wird. Ferner wissen wir aus den Überlegungen, die wir im Zusammenhang mit dem Quellencodierungstheorem ange stellt haben, dass das Gesagte ohne das Wörtchen „ungefähr“ falsch wäre. Für viele konkrete Werte von  $n$  ist  $np$  gar keine ganze Zahl, sodass die Anzahl der gekippten Bits eben nur ungefähr dieser Zahl entsprechen kann. Das bedeutet, dass wir, genau wie im Falle des Quellencodierungstheorems, Nachrichten in unsere Betrachtung miteinbeziehen müssen, in denen die Fehlerwahrscheinlichkeit geringfügig von  $p$  abweicht. Wir lösen das Problem, indem wir eine empfangene Sequenz der Länge  $n$  als typisch erachten, wenn sie für eine vorgegebene Kon stante  $\delta$  in der folgenden Menge liegt:

$$S(n, \mathbf{v}) := \left\{ \mathbf{w} \in \mathbb{Z}_2^n \mid (p - \delta)n < \Delta(\mathbf{v}, \mathbf{w}) < (p + \delta)n \right\} \quad (7.21)$$

Die Konstante  $\delta$  dürfen wir beliebig klein wählen. Nach dem Gesetz der großen Zahlen strebt die Wahrscheinlichkeit, dass der Sender ein Codewort  $\mathbf{v}$  sendet und der Empfänger eine Sequenz aus  $S(n, \mathbf{v})$  erhält, für  $n \rightarrow \infty$  gegen 1.

Die Anzahl der Möglichkeiten, eine Folge von  $n$  Bits an  $np$  Positionen zu verändern, können wir über den Binomialkoeffizienten berechnen. Hierüber erhalten wir die Abschätzung

$$|S(n, \mathbf{v})| \approx \binom{n}{np},$$

die für steigende Werte von  $n$  immer genauer wird. Jetzt müssen wir den Binomialkoeffizienten nur noch geschickt umformen. Hierzu ersetzen wir diesen zunächst durch die folgende Definition:

$$\binom{n}{np} := \frac{n!}{(np)! \cdot (n(1-p))!} \quad (7.22)$$

Jetzt greifen wir einem Ergebnis vor, das Sie im Übungsteil auf Seite 563 erarbeiten werden. Dort werden Sie zeigen, dass sich unter Zu hilfenahme der Stirling-Formel

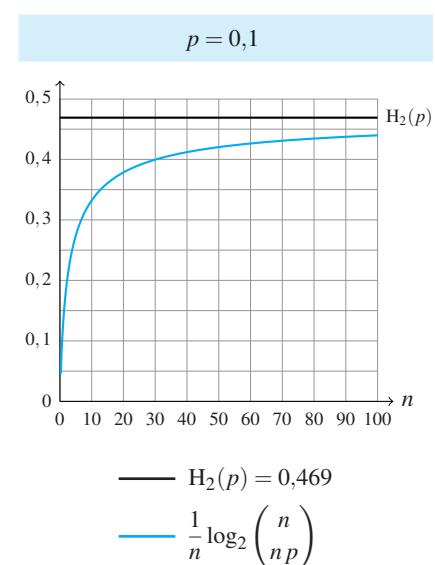
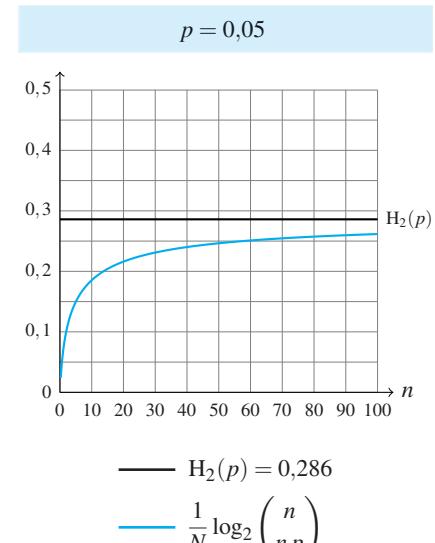
$$n! \approx \left(\frac{n}{e}\right)^n \sqrt{2\pi n} \quad (\text{Abb. 7.27})$$

die folgende Beziehung herleiten lässt:

$$\lim_{n \rightarrow \infty} \frac{1}{n} \log_2 \binom{n}{np} = H_2(p) \quad (\text{Abb. 7.28})$$

Aus dieser Formel folgt, dass wir für steigende Werte von  $n$  einen immer kleineren Fehler machen, wenn wir für  $\log_2 |S(n, \mathbf{v})|$  die Abschätzung

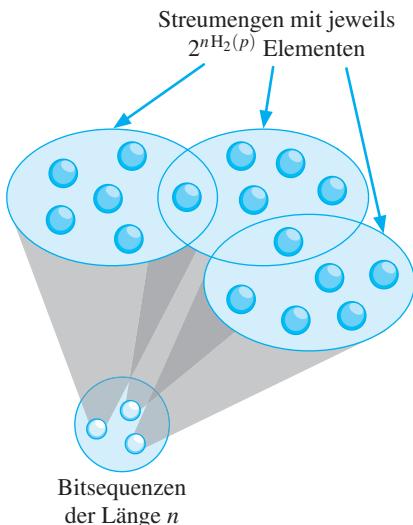
$$\log_2 |S(n, \mathbf{v})| \approx n H_2(p)$$



**Abb. 7.28:** Für kontinuierlich wachsende Werte von  $n$  rückt der Wert

$$\frac{1}{n} \log_2 \binom{n}{np}$$

immer näher an die Entropie  $H_2(p)$  heran.



**Abb. 7.29:** Für wachsende Werte von  $n$  strebt die Wahrscheinlichkeit gegen 1, dass die Bitsequenzen, die am Eingang des Übertragungskanals beobachtet werden können, auf jeweils eine von  $2^{nH_2(p)}$  typischen Bitsequenzen abgebildet werden. Die Zielmengen bezeichnen wir als *Streumengen*.

und für  $|S(n, v)|$  die Abschätzung

$$|S(n, v)| \approx 2^{nH_2(p)}$$

verwenden.

Damit haben wir ein wichtiges Etappenziel erreicht. Wir wissen jetzt, dass eine in den Kanal eingespeiste Bitsequenz der Länge  $n$  für steigende Werte von  $n$  mit einer an Sicherheit grenzenden Wahrscheinlichkeit auf der Übertragungsstrecke in eine von  $2^{nH_2(p)}$  typischen Bitsequenzen verfälscht wird (Abbildung 7.29). Die Menge  $S(n, v)$ , in die ein eingespeistes Codewort  $v$  der Länge  $n$  typischerweise abgebildet wird, bezeichnen wir als die *Streumenge* von  $v$ .

Im Folgenden sei  $v$ , wie bisher, das gesendete Codewort und  $w$  die empfangene Bitsequenz. Sobald  $w$  eintrifft, versucht der Empfänger  $v$  zu rekonstruieren, indem er alle Streumengen bestimmt, die  $w$  enthalten. Das Einzige, was er hierfür benötigt, ist das Wörterbuch, das ihm bereits vorliegt. Anschließend unterscheidet er drei Fälle (vgl. Abbildung 7.30):

- 1. Fall: Es gibt genau eine Streumenge  $S(n, v')$  mit  $w \in S(n, v')$ .

Dies ist der erhoffte Fall. Da es genau eine Streumenge  $S(n, v')$  gibt, in der  $w$  enthalten ist, geht der Empfänger davon aus, dass  $v'$  auch das gesendete Codewort war. Der Decoder gibt  $v'$  aus und bearbeitet die nächste ankommende Bitsequenz.

Natürlich bedeutet die Existenz einer eindeutig bestimmten Streumenge nicht, dass  $v'$  wirklich das gesendete Codewort  $v$  ist. Die gefundene Streumenge kann jedoch nur dann die falsche sein, wenn  $v$  auf der Übertragungsstrecke ungewöhnlich stark verfälscht wurde, und wir wissen bereits, dass die Wahrscheinlichkeit hierfür für wachsende Werte von  $n$  immer kleiner wird. Mit anderen Worten: Für  $n \rightarrow \infty$  geht die Wahrscheinlichkeit gegen 1, dass die empfangene Sequenz  $w$  innerhalb der richtigen Streumenge bleibt.

- 2. Fall: Es wird für  $w$  keine passende Streumenge gefunden.

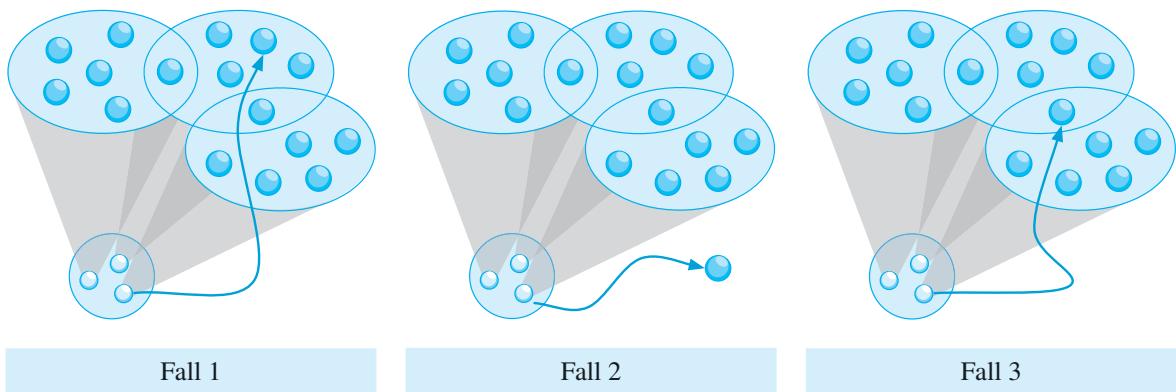
In diesem Fall scheitert die Decodierung. Da keine Streumenge gefunden werden konnte, hat der Decoder keinerlei Anhaltspunkte über die Gestalt des ursprünglich gesendeten Codeworts. Wir wissen aber schon, dass auch dieses Fehlerszenario ebenfalls immer unwahrscheinlicher wird, wenn wir die Codewortlänge  $n$  sukzessive erhöhen. Für  $n \rightarrow \infty$  geht die Wahrscheinlichkeit gegen 1, dass das Codewort  $v$  auf eine Sequenz gestreut wird, die zu seiner eigenen Streumenge  $S(n, v)$  gehört, und damit strebt die Wahrscheinlichkeit gegen 0, dass den Empfänger eine Bitsequenz erreicht, die überhaupt keiner Streumenge angehört.

Auf der Empfängerseite schlägt die Decodierung fehl, wenn das gesendete Codewort auf der Übertragungsstrecke ...

... zu einer Bitsequenz wird,  
die zu einer falschen  
Streumenge gehört.

... zu einer Bitsequenz wird,  
die zu gar keiner Streumenge  
gehört.

... zu einer Bitsequenz wird,  
die zu mehreren  
Streumengen gehört.



**Abb. 7.30:** Drei Fehlerszenarien muss der Empfänger unterscheiden.

- 3. Fall: Es werden mehrere zu  $w$  passende Streumengen gefunden.

Dieser Fall ist der eigentlich interessante: Sobald der Decoder mehrere Streumengen findet, kann er sich nicht eindeutig festlegen; er hat keinerlei Anhaltspunkte darüber, welche der gefundenen Streumengen die richtige ist. Auf den ersten Blick sieht es so aus, als würden wir nun den Preis dafür zahlen, dass wir mit der zufälligen Verteilung der Codewörter eine sehr primitive Zuordnungsvorschrift gewählt haben. Wenn wir die Codewörter willkürlich verteilen, so können wir doch sicher nicht erwarten, dass die Streumengen sauber voneinander getrennt sind, oder etwa doch? Eine genauere Betrachtung der Situation wird uns eine erstaunliche Antwort bescheren. Sehen wir also genauer hin!

Wieder sei  $v$  das gesendete Codewort und  $w$  die empfangene Bitsequenz. Ferner sei  $v'$  ein beliebiges Codewort mit  $v \neq v'$ . Um die Wahrscheinlichkeit zu berechnen, dass die Bitsequenz  $w$  einer einzigen Streumenge angehört, überlegen wir uns zunächst, wie groß die Wahrscheinlichkeit ist, dass  $w$  innerhalb der Streumenge von  $v'$  liegt. Da die Codewörter zufällig verteilt sind, ist die Wahrscheinlichkeit genauso groß wie die Wahrscheinlichkeit, aus einer Menge von  $2^n$  Bitmustern zufällig

eines von  $2^{nH_2(p)}$  besonderen Bitmustern auszuwählen:

$$p(\mathbf{w} \in S(n, \mathbf{v}')) = \frac{2^{nH_2(p)}}{2^n} = 2^{n(H_2(p)-1)}$$

Damit gilt für die Wahrscheinlichkeit, dass  $\mathbf{w}$  außerhalb von  $S(\mathbf{v}')$  liegt:

$$p(\mathbf{w} \notin S(n, \mathbf{v}')) = 1 - 2^{n(H_2(p)-1)}$$

Insgesamt gibt es  $2^k - 1$  verschiedene Streumengen, in denen  $\mathbf{w}$  nicht enthalten sein darf. Damit können wir die Wahrscheinlichkeit, dass  $\mathbf{w}$  in mehr als einer Streumenge liegt und die Decodierung deshalb fehlschlägt, folgendermaßen berechnen:

$$p_{\text{rest}} = 1 - \left(1 - 2^{n(H_2(p)-1)}\right)^{2^k-1}$$

Bringen wir über die Beziehung  $k = Rn$  die Coderate mit ins Spiel, so liest sich diese Formel so:

$$p_{\text{rest}} = 1 - \left(1 - 2^{n(H_2(p)-1)}\right)^{2^{Rn}-1} \quad (7.23)$$

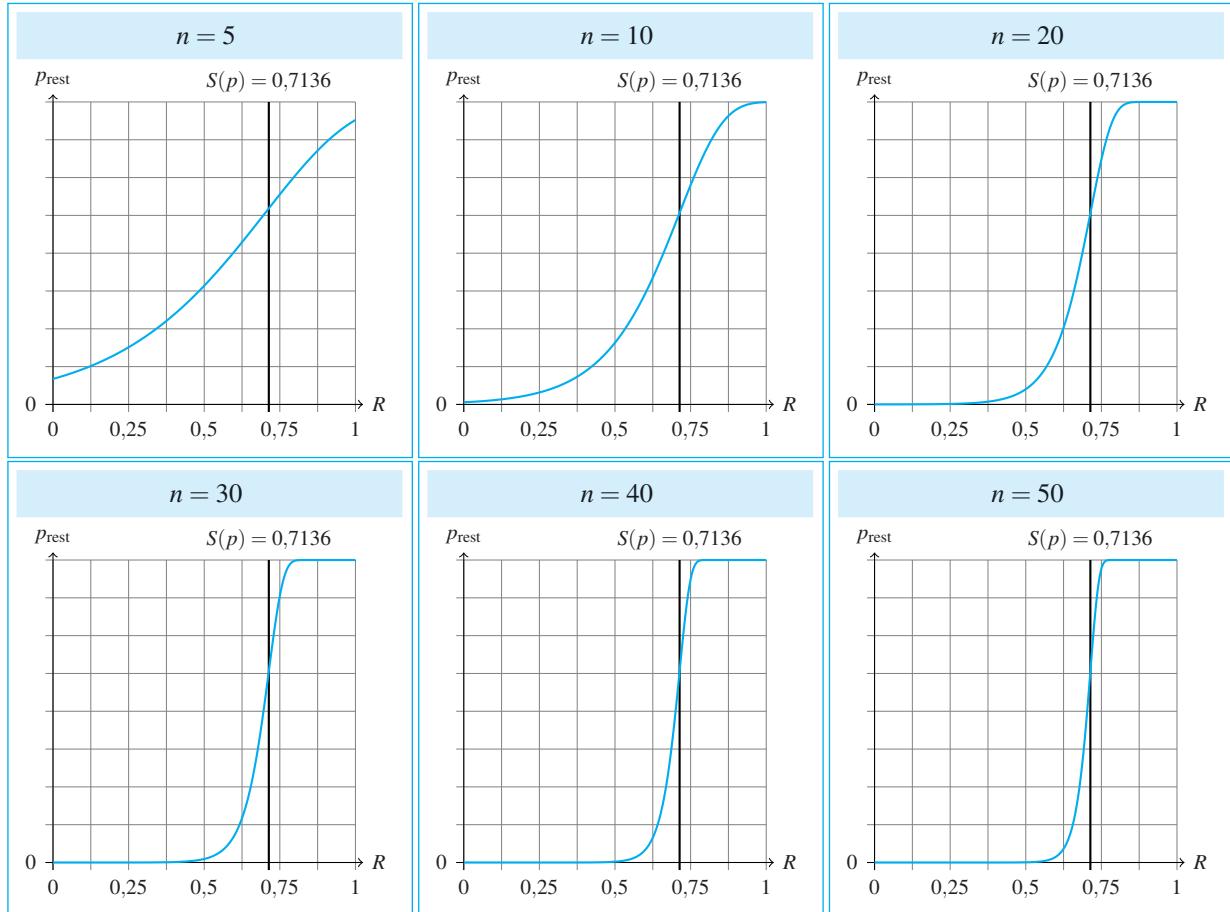
Ein Blick auf die Abbildungen 7.31 und 7.32 offenbart Erstaunliches. Setzen wir für  $n$  immer größere Werte ein, so konvergiert die Restfehlerwahrscheinlichkeit  $p_{\text{rest}}$  links des Shannon-Limits gegen 0 und rechts des Shannon-Limits gegen 1:

$$\lim_{n \rightarrow \infty} p_{\text{rest}} = 0 \quad \text{für } R = S(p) - \varepsilon \quad (7.24)$$

$$\lim_{n \rightarrow \infty} p_{\text{rest}} = 1 \quad \text{für } R = S(p) + \varepsilon \quad (7.25)$$

Die Grenzwertbeziehung (7.24) zeigt, dass unsere Codierung tatsächlich ihren Zweck erfüllt. Bleiben wir unterhalb des Shannon-Limits, so können wir die Restfehlerwahrscheinlichkeit durch eine Verlängerung der Codewörter beliebig senken. Ein wahrhaft bemerkenswertes Resultat!

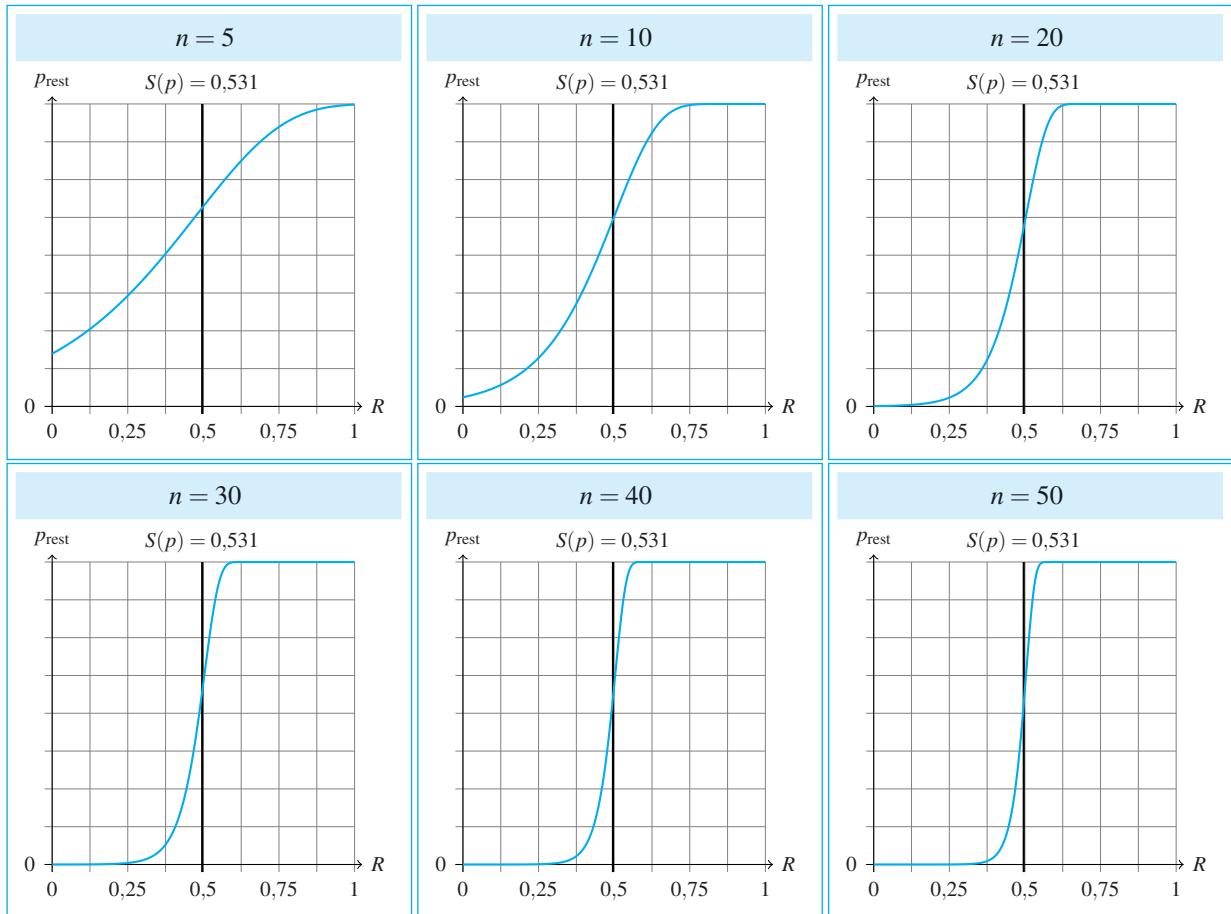
Gleichermaßen macht die Grenzwertbeziehung (7.25) klar, dass die gewählte Codierung versagt, sobald wir das Shannon-Limit auch nur minimal überschreiten. In diesem Fall führt die Verlängerung der Codewörter dazu, dass die Restfehlerwahrscheinlichkeit ansteigt. Natürlich ist damit noch nicht bewiesen, dass jenseits des Shannon-Limits keine zuverlässige Übertragung möglich ist, schließlich haben wir in den angestellten Überlegungen einen sehr speziellen Kanalcodierer benutzt. Theoretisch wäre es denkbar, dass lediglich die Methode der zufallsgenerierten Codewörter in diesem Fall versagt und durch eine geschicktere, zielgerichtete Wahl der Codewörter die Coderate über das Shannon-Limit hinaus gesteigert werden kann.



**Abb. 7.31:** Restfehlerwahrscheinlichkeit des Zufallscodierers in Abhängigkeit der Coderate  $R$  und der Codewortlänge  $n$ , berechnet für einen Übertragungskanal mit der Fehlerwahrscheinlichkeit  $p = 0,05$

### Coderaten oberhalb des Shannon-Limits

Wir wollen uns mit einem sehr allgemein gehaltenen Argument davon überzeugen, dass diese Hoffnung vergebens ist. Der Schlüssel hierfür ist die Erkenntnis, dass gar nicht die Verteilung der Streumengen das Problem ist – der Zufallscodierer verteilt diese Mengen hinreichend gut –, sondern die Anzahl der Bitsequenzen, die in den Streumengen enthalten sind. Um dies einzusehen, stellen wir uns einen beliebigen Kanalcodierer vor und beobachten diesen längere Zeit. Die Anzahl der Bits, die er in dem Beobachtungszeitraum entgegennimmt, bezeichnen wir mit  $K$

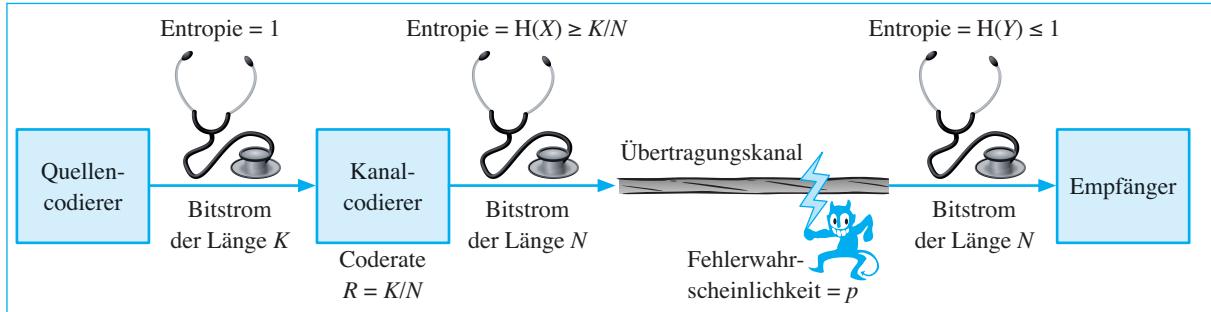


**Abb. 7.32:** Restfehlerwahrscheinlichkeit des Zufallscodierers in Abhängigkeit der Coderate  $R$  und der Codewortlänge  $n$ , berechnet für einen Übertragungskanal mit der Fehlerwahrscheinlichkeit  $p = 0,1$

und die Anzahl der Bits, die er daraus produziert, mit  $N$ . Wir nehmen an, dass der Kanalcodierer mit einer Coderate arbeitet, die das Shannon-Limit übersteigt. Es gilt also:

$$\frac{K}{N} = S(p) + \varepsilon = (1 - H_2(p)) + \varepsilon \quad (7.26)$$

Betrachten wir den Datenstrom am Ausgang des Kanalcodierers, oder, was dasselbe ist, am Eingang des Übertragungskanals, so können wir diesem eine Entropie  $H(X)$  zuordnen. Genauso können wir auch den Ausgang der Übertragungsstrecke betrachten und dem beobachtbaren



**Abb. 7.33:**  $H(X)$  ist die Entropie am Eingang des Übertragungskanals und  $H(Y)$  die Entropie am Ausgang.

Datenstrom ebenfalls eine Entropie zuordnen, die wir mit  $H(Y)$  bezeichnen (Abbildung 7.33).

Jetzt greifen wir auf ein Ergebnis zurück, das wir in Abschnitt 5.3 im Zusammenhang mit dem Quellencodierungstheorem erzielt haben. Dort haben wir die Menge  $T_\delta(N)$  definiert, die alle Sequenzen der Länge  $N$  enthält, die von einer Datenquelle  $X$  typischerweise emittiert werden. Die Anzahl typischer Sequenzen kennen wir ebenfalls aus Abschnitt 5.3. Es gilt die folgende Beziehung:

$$\lim_{N \rightarrow \infty} |T_\delta(N)| = 2^{N H(X)}$$

Lassen wir  $N$  kontinuierlich wachsen, so wird es immer wahrscheinlicher, dass eine Sequenz aus der Menge  $T_\delta(N)$  in den Übertragungskanal eingespeist wird. Deshalb machen wir für immer größere Werte von  $N$  einen immer kleineren Fehler, wenn wir unsere Betrachtung auf die Elemente aus  $T_\delta(N)$  beschränken.

Wir wissen also, dass in dem Beobachtungszeitraum  $2^{N H(X)}$  typische Sequenzen am Eingang des Kanalcodierers auftreten können und jede dieser Sequenzen am Ausgang des Übertragungskanals auf eine von  $2^{N H_2(p)}$  typischen Sequenzen gestreut wird. Wir wissen ferner, dass am Ausgang des Übertragungskanals

$$2^{N H(Y)} \quad (7.27)$$

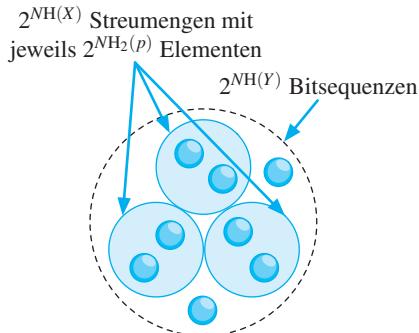
typische Sequenzen beobachtet werden können. Um eine zuverlässige Übertragung sicherzustellen, müssen also ungefähr

$$2^{N H(X)} \cdot 2^{N H_2(p)} \quad (7.28)$$

Bitsequenzen am Ausgang des Übertragungskanals beobachtbar sein, um die Streumengen hinreichend sauber voneinander zu trennen. Eine

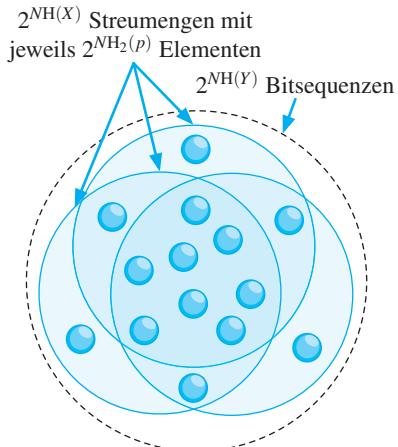
## Unterhalb des Shannon-Limits

$$R = \frac{K}{N} < S(p)$$



## Oberhalb des Shannon-Limits

$$R = \frac{K}{N} > S(p)$$



**Abb. 7.34:** Liegt die Coderate oberhalb des Shannon-Limits, so sind die 2<sup>NH(Y)</sup> zur Verfügung stehenden Bitmuster zu wenige, um eine saubere Trennung der Streumengen zu gewährleisten. Die sukzessive Verlängerung der Codewörter drängt die Mengen unablässig dichter zusammen, sodass es immer wahrscheinlicher wird, eine zufällig ausgewählte Bitsequenz in mehreren Streumengen vorzufinden.

einfache Überlegung zeigt aber, dass genau dies nicht der Fall ist. Wir müssen hierfür lediglich beobachten, wie sich das Verhältnis zwischen (7.27) und (7.28) für  $N \rightarrow \infty$  entwickelt. Bevor wir den Quotienten

$$\frac{2^{NH(Y)}}{2^{NH(X)} \cdot 2^{NH_2(p)}} \quad (7.29)$$

ausrechnen, machen wir uns zunächst Gedanken über die Größen  $H(X)$  und  $H(Y)$ . Für letztere gilt die Abschätzung

$$H(Y) \leq 1, \quad (7.30)$$

da wir es mit einem binären Datenstrom zu tun haben. Um  $H(X)$  abzuschätzen, erinnern wir uns zunächst daran, dass der Kanalcodierer mit einem Datenstrom gespeist wird, der optimal komprimiert wurde. Das bedeutet, dass jede entgegengenommene 0 oder 1 genau ein Bit an Information in sich trägt. Der Kanalcodierer produziert einen Ausgabedatenstrom, dessen Informationsgehalt nicht kleiner sein kann als der Informationsgehalt des Eingabedatenstroms; die  $N$  Ausgabebits enthalten also mindestens so viel Information wie die  $K$  Eingabebits. Dieser Zusammenhang beschert uns die folgende Abschätzung:

$$H(X) \geq \frac{K}{N} \quad (7.31)$$

Jetzt können wir den Quotienten (7.29) folgendermaßen umformen:

$$\begin{aligned} \frac{2^{NH(Y)}}{2^{NH(X)} \cdot 2^{NH_2(p)}} &\stackrel{(7.30)}{\leq} \frac{2^N}{2^{NH(X)} \cdot 2^{NH_2(p)}} \\ &\stackrel{(7.31)}{\leq} \frac{2^N}{2^K \cdot 2^{NH_2(p)}} = 2^{N(1-H_2(p))-K} \end{aligned} \quad (7.32)$$

Aus (7.26) folgt

$$N(1 - H_2(p)) - K = -N\varepsilon,$$

womit wir (7.32) zu

$$\frac{2^{NH(Y)}}{2^{NH(X)} \cdot 2^{NH_2(p)}} \leq 2^{-N\varepsilon}$$

umformen können. Der Quotient konvergiert für  $N \rightarrow \infty$  gegen 0, und zwar unabhängig davon, wie klein wir  $\varepsilon$  wählen. Das bedeutet, dass die zur Verfügung stehenden Sequenzen so wenige sind, dass sich die Mengen, auf die die eingehenden Sequenzen gestreut werden, in einem immer stärkeren Maße überlagern. Mit anderen Worten: Die 2<sup>NH(Y)</sup> zur Verfügung stehenden Sequenzen bieten zu wenig Platz, um eine saubere Trennung der Streumengen zu gewährleisten (Abbildung 7.34).

Jetzt können wir noch besser verstehen, warum der von uns konstruierte Kanalcodierer für Coderaten unterhalb des Shannon-Limits funktioniert, für Coderaten oberhalb dagegen nicht. Bei einer Coderate  $R$  mit  $R < S(p)$  sind ausreichend viele Sequenzen vorhanden, um die Streumengen zu trennen, und die zufällige Wahl der Codewörter sorgt dafür, dass sich die Streumengen für wachsende Werte von  $n$  so gleichmäßig verteilen, dass die Wahrscheinlichkeit gegen 0 geht, eine zufällig ausgewählte Binärsequenz in der Schnittmenge zweier oder mehrerer Streumengen vorzufinden. Bei einer Coderate  $R$  mit  $R > S(p)$  werden die Streumengen zwar immer noch gleichmäßig verteilt, allerdings führt der zunehmende Platzmangel dazu, dass sie sich immer stärker überlappen. Die Trennung der Streumengen wird durch eine Verlängerung der Codewörter daher nicht mehr verbessert, sondern verschlechtert.

Vielleicht ist Ihnen aufgefallen, dass das Kanalcodierungstheorem lediglich eine Aussage für Coderaten unterhalb und oberhalb des Shannon-Limits macht, aber keine Aussage für den Fall, dass die Coderate exakt dem Shannon-Limit entspricht. Der Grund hierfür ist, dass unser Kanalcodierer, der mit zufällig generierten Codewörtern arbeitet, bei dieser Coderate nicht mehr zuverlässig funktioniert. Dies wird ersichtlich, wenn wir die Beziehung  $R = S(p)$  in die Formel (7.23) einsetzen. Wir erhalten dann für die Restfehlerwahrscheinlichkeit die folgende Beziehung:

$$p_{\text{rest}} = 1 - \left(1 - 2^{n(H_2(p)-1)}\right)^{2^n R - 1} \stackrel{(7.20)}{=} 1 - \left(1 - 2^{-nS(p)}\right)^{2^n S(p) - 1}$$

Im Übungsteil auf Seite 564 werden Sie zeigen, dass wir in diesem Fall die folgende Grenzwertbeziehung erhalten:

$$\lim_{n \rightarrow \infty} p_{\text{rest}} = 1 - \frac{1}{e} \approx 0,63 \quad (7.33)$$

Die Restfehlerwahrscheinlichkeit unseres Kanalcodierers stagniert bei einem Wert  $> 0$ , sodass die Verlängerung der Codewörter keine Verbesserung mit sich bringt. Auf der anderen Seite läuft unser Argument ins Leere, dass sich die Streumengen bei dieser Coderate immer stärker zusammendrängen. Betrachten wir den Quotienten (7.29) für den Fall  $R = S(p)$ , so gilt  $K = N(1 - H_2(p))$  und wir erhalten:

$$\frac{2^{N\text{H}(Y)}}{2^{N\text{H}(X)} \cdot 2^{N\text{H}_2(p)}} \leq \frac{2^N}{2^K \cdot 2^{N\text{H}_2(p)}} = \frac{2^N}{2^{N(1-H_2(p))} \cdot 2^{N\text{H}_2(p)}} = 1$$



Wir sind in diesem Abschnitt zu der Erkenntnis gelangt, dass sich mit einem Kanalcodierer, der mit zufällig generierten Codewörtern arbeitet, eine beliebig geringe Restfehlerwahrscheinlichkeit erreichen lässt. Dies wirft die Frage auf, warum wir uns in den vorangegangenen Abschnitten überhaupt die Mühe gemacht haben, so viele verschiedene und mitunter auch recht komplizierte Kanalcodes zu besprechen. Können wir Shannons Beweisidee nicht einfach in die Praxis tragen und mit einem Kanalcodierer, der nach dem geschilderten Prinzip arbeitet, sämtliche Probleme lösen? Die Antwort lautet Nein! Damit der Encoder und der Decoder eine Nachricht codieren bzw. decodieren können, müssen die Codewörter bekannt sein, und dies wäre bei einem zufallsgenerierten Code nur dann möglich, wenn sämtliche Codewörter in einem großen Wörterbuch vorgehalten werden. Die Größe des Wörterbuchs wächst jedoch exponentiell mit der Codewortlänge, sodass wir in der Praxis keine Chance haben, diese hinreichend groß zu wählen. Aber genau dies müssten wir tun, um mit einem zufallsgenerierten Code sinnvolle Ergebnisse zu erzielen.

## 7.6 Übungsaufgaben

### Aufgabe 7.1



**Webcode  
7033**

Auf Seite 513 wurde erwähnt, dass keine nichttrivialen binären MDS-Codes existieren. Fast im gleichen Atemzug haben wir uns davon überzeugt, dass die Reed-Solomon-Codes zu den MDS-Codes gehören. RS-Codes lassen sich über beliebigen endlichen Körpern definieren und somit auch über dem Körper  $\mathbb{Z}_2$ . Reed-Solomon-Codes über  $\mathbb{Z}_2$  sind aber Binärcodes und dies scheint uns einen herben Widerspruch zu beschreiben. Können Sie die augenscheinlich paradoxe Situation aufklären?

### Aufgabe 7.2



**Webcode  
7165**

Welche der folgenden Aussagen sind richtig? Welche sind falsch?

- a) „Ein  $k$ -fehlerkorrigierender Code ist höchstens  $2k$ -fehlererkennend.“
- b) „Die Code-Distanz eines perfekten Codes ist immer ungerade.“
- c) „Ein perfekter  $k$ -fehlerkorrigierender Code ist höchstens  $2k$ -fehlererkennend.“

### Aufgabe 7.3

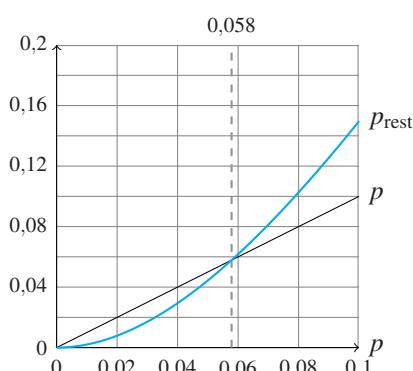


**Webcode  
7247**

Auf Seite 538 haben wir für den [7,4]-Hamming-Code die folgende Formel für die Berechnung der Restfehlerwahrscheinlichkeit hergeleitet:

$$p_{\text{rest}} = 1 - (1-p)^7 - 7p(1-p)^6$$

Wie die untenstehende Grafik zeigt, liegt die Restfehlerwahrscheinlichkeit für  $p > 0,058$  sogar oberhalb von  $p$ . Bedeutet dies, dass die Hamming-Codierung bei Fehlerraten oberhalb dieser Grenze keinen Sinn mehr macht und wir die Daten besser im Klartext übertragen?



In dieser Aufgabe wollen wir die auf Seite 537 angegebene Formel

$$p_{\text{rest}} = 1 - \sum_{i=0}^r \binom{n}{i} p^i (1-p)^{n-i}$$

auf den Prüfstand stellen. Hierfür wurde das nachstehend gezeigte Bild mithilfe dreier verschiedener Wiederholungscodes über einen verrauschten Kanal übertragen und in den empfangenen Bildern die Pixelfehler gezählt.

### Aufgabe 7.4



**Webcode  
7356**

[5,1]-Wiederholungscode

$$\text{Coderate } R = \frac{1}{5}$$



Fehlerwahrscheinlichkeit:  $p = 0,1$



$$p_{\text{rest}} = 0,00855 \quad 0,89 \% \text{ Pixelfehler}$$

[7,1]-Wiederholungscode

$$\text{Coderate } R = \frac{1}{7}$$



Fehlerwahrscheinlichkeit:  $p = 0,1$



$$p_{\text{rest}} = 0,00272 \quad 0,26 \% \text{ Pixelfehler}$$

[9,1]-Wiederholungscode

$$\text{Coderate } R = \frac{1}{9}$$



Fehlerwahrscheinlichkeit:  $p = 0,1$



$$p_{\text{rest}} = 0,00089 \quad 0,09 \% \text{ Pixelfehler}$$

Der experimentell bestimmte Wert und der theoretisch vorhergesagte liegen sehr nahe zusammen. Offenbar funktioniert unsere Formel!

Als Nächstes wiederholen wir unser Experiment mit drei verschiedenen Hamming-Codes:

[15,11]-Hamming-Code

[7,4]-Hamming-Code

[3,1]-Hamming-Code

$$\text{Coderate } R = \frac{11}{15}$$

$$\text{Coderate } R = \frac{4}{7}$$

$$\text{Coderate } R = \frac{1}{3}$$



Fehlerwahrscheinlichkeit:  $p = 0,1$

Fehlerwahrscheinlichkeit:  $p = 0,1$

Fehlerwahrscheinlichkeit:  $p = 0,1$



$p_{\text{rest}} = 0,4510 \quad 10,44 \% \text{ Pixelfehler}$



$p_{\text{rest}} = 0,1496 \quad 6,79 \% \text{ Pixelfehler}$



$p_{\text{rest}} = 0,0280 \quad 2,79 \% \text{ Pixelfehler}$

- a) Das Ergebnis erstaunt. Für den [15,11]-Hamming-Code erhalten wir eine Restfehlerwahrscheinlichkeit von ca. 45 %, aber nur ca. 10 % verfälschte Pixel. Ist unsere Formel etwa falsch?
- b) Warum lässt sich bei den Wiederholungscodes keine signifikante Abweichung beobachten?
- c) Warum zeigt der [3,1]-Hamming-Code, im Gegensatz zu den anderen beiden Hamming-Codes, ebenfalls keine signifikante Abweichung?

In dieser Aufgabe wollen wir die Restfehlerwahrscheinlichkeit des [7,4]-Hamming-Codes für den Fall bestimmen, dass dieser nicht, wie üblich, zur Fehlerkorrektur, sondern lediglich zur Fehlererkennung verwendet wird.

**Aufgabe 7.5**
**Webcode**  
**7461**

- a) In der nachstehenden Tabelle sind alle Codewörter des [7,4]-Hamming-Codes aufgelistet, und für zwei Bitmuster ist bereits vermerkt, wie stark sie die Restfehlerwahrscheinlichkeit beeinflussen. Führen Sie eine entsprechende Berechnung für die anderen Codewörter durch.

<b>v</b>	<b>v</b>
0000000	1110000
1101001	$(1-p)^3 p^4$
0101010	$(1-p)^4 p^3$
1000011	_____
1001100	_____
0100101	_____
1100110	_____
0001111	_____

- b) Stellen Sie mithilfe der ausgefüllten Tabelle die Formel zur Berechnung der Restfehlerwahrscheinlichkeit auf.

- c) Berechnen Sie die Restfehlerwahrscheinlichkeit für die folgenden Werte:

$p = 0,01$	$p = 0,05$	$p = 0,1$	$p = 0,2$	$p = 0,3$	$p = 0,4$	$p = 0,5$
_____	_____	_____	_____	_____	_____	_____

- d) Hätten Sie das Ergebnis für  $p = 0,5$  auch ohne die Formel vorhersagen können?

Auf Seite 536 haben wir die folgende Formel hergeleitet, mit der sich die Restfehlerwahrscheinlichkeit des geraden Paritätscodes der Länge 5 berechnen lässt:

$$p_{\text{rest}} = 10p^2(1-p)^3 + 5p^4(1-p)$$

Wir wollen die Berechnung für den ungeraden Paritätscode der gleichen Länge wiederholen und beginnen, wie auf Seite 535, mit dem Aufstellen einer Wahrscheinlichkeitstabelle:

**Aufgabe 7.6**
**Webcode**  
**7577**

<b>v</b>		<b>v</b>	
00001	$(1-p)^4 p$	10000	$(1-p)^4 p$
00010	$(1-p)^4 p$	10011	$(1-p)^2 p^3$
00100	$(1-p)^4 p$	10101	$(1-p)^2 p^3$
00111	$(1-p)^2 p^3$	10110	$(1-p)^2 p^3$
01000	$(1-p)^4 p$	11001	$(1-p)^2 p^3$
01011	$(1-p)^2 p^3$	11010	$(1-p)^2 p^3$
01101	$(1-p)^2 p^3$	11100	$(1-p)^2 p^3$
01110	$(1-p)^2 p^3$	11111	$p^5$

Damit ergibt sich für die Restfehlerwahrscheinlichkeit die folgende Formel:

$$p_{\text{rest}} = 5p(1-p)^4 + 10p^3(1-p)^2 + p^5$$

Für  $p = 0,01$  erhalten wir mit

$$p_{\text{rest}} = 0,0480396016$$

ein anderes Ergebnis als jenes, das wir für den geraden Paritätscode ermittelt haben. Kann es wirklich sein, dass die beiden Paritäscodes unterschiedliche Restfehlerwahrscheinlichkeiten aufweisen?

### Aufgabe 7.7



**Webcode  
7619**

Auf Seite 453 haben Sie den Hadamard-Code kennengelernt, mit dem die Raumsonde Mariner 9 das aufgenommene Bildmaterial zur Erde übertragen hat.

a) Rekapitulieren Sie für den Mariner-Code die folgenden Parameter:

Codewortlänge  $n =$

Code-Distanz  $\Delta C =$

Radius  $r$  einer Hamming-Kugel =

b) Gehen Sie davon aus, dass 5 % aller Bits bei der Übertragung verfälscht werden. Berechnen Sie daraus die Restfehlerwahrscheinlichkeit des Mariner-Codes.

### Aufgabe 7.8



**Webcode  
7745**

Der [10,2]-Wiederholungscode besteht aus den folgenden Codewörtern:

{0000000000, 0101010101, 1010101010, 1111111111}

- a) Wie groß ist der Radius der Hamming-Kugeln, wenn der Code für die Fehlerkorrektur eingesetzt wird?
- b) Schreiben Sie ein Programm, das genau diejenigen Bitsequenzen berechnet, die zu keiner Hamming-Kugel gehören.



- c) Wie groß ist der Prozentsatz dieser nicht zuordenbaren Bitsequenzen?

Abschnitt 7.5 haben wir mit dem folgenden Zitat begonnen, das dem Klassiker *Information Theory, Inference, and Learning Algorithms* von David MacKay entnommen ist [58]:

**Aufgabe 7.9**  
  
**Webcode**  
**7813**

„What performance are you trying to achieve?  $10^{-15}$ ? You don't need sixty disk drives – you can get that performance with just two disk drives [...]. And if you want  $p_b = 10^{-18}$  or  $10^{-24}$  or anything, you can get there with two disk drives too!“

In MacKay's Buch geht dieses Zitat mit dem folgenden Satz weiter:

„Strictly, the above statements might not be quite right [...].“

Was könnte der Autor damit gemeint haben?

Zeigen Sie unter Zuhilfenahme der *Stirling-Formel*

$$n! \approx \left(\frac{n}{e}\right)^n \sqrt{2\pi n}$$

die folgende Beziehung:

$$\lim_{n \rightarrow \infty} \frac{1}{n} \log_2 \binom{n}{np} = H_2(p)$$

**Aufgabe 7.10**  
  
**Webcode**  
**7865**

**Aufgabe 7.11** Die in diesem Abschnitt formulierten Beziehungen**Webcode  
7956**

$$\lim_{n \rightarrow \infty} p_{\text{rest}} = 0 \quad \text{für } R = S(p) - \varepsilon \quad (\text{Formel (7.24) auf Seite 552})$$

$$\lim_{n \rightarrow \infty} p_{\text{rest}} = 1 \quad \text{für } R = S(p) + \varepsilon \quad (\text{Formel (7.25) auf Seite 552})$$

$$\lim_{n \rightarrow \infty} p_{\text{rest}} = 1 - \frac{1}{e} \quad \text{für } R = S(p) \quad (\text{Formel (7.33) auf Seite 557})$$

sind eine unmittelbare Folge aus den folgenden Grenzwertgleichungen:

$$\lim_{x \rightarrow \infty} (1 - 2^{-ax})^{2^{(a-\varepsilon)x}} = 1 \quad (a > 0, \varepsilon > 0) \quad (7.34)$$

$$\lim_{x \rightarrow \infty} (1 - 2^{-ax})^{2^{(a+\varepsilon)x}} = 0 \quad (a > 0, \varepsilon > 0) \quad (7.35)$$

$$\lim_{x \rightarrow \infty} (1 - 2^{-ax})^{2^{ax}} = \frac{1}{e} \quad (a > 0) \quad (7.36)$$

Beweisen Sie die Grenzwertgleichungen (7.34), (7.35) und (7.36). Erinnern Sie sich hierfür an die Regel von L'Hospital, die Sie in der Mathematikvorlesung kennengelernt haben.

**Aufgabe 7.12****Webcode  
7983**

Im Zusammenhang mit fehlerkorrigierenden Codes sind wir in den allermeisten Fällen davon ausgegangen, dass die Korrektur mithilfe von *Maximum-Likelihood-Decodern* erfolgt. Wird eine Bitsequenz  $w$  empfangen, so wählt ein solcher Decoder für die Korrektur dasjenige Codewort  $v$  aus, das zu  $w$  die geringste Hamming-Distanz aufweist.

Im Beweis des Kanalcodierungstheorems hat Claude Shannon einen Decoder benutzt, der anders funktioniert. Empfängt dieser eine Sequenz  $w$  der Länge  $n$ , so werden zunächst alle Streumengen  $S(n, v)$  mit  $w \in S(n, v)$  bestimmt. Gibt es genau eine Streumenge, so wird das zugehörige Codewort  $v$  ausgegeben. Ansonsten wird ein Fehler signalisiert.

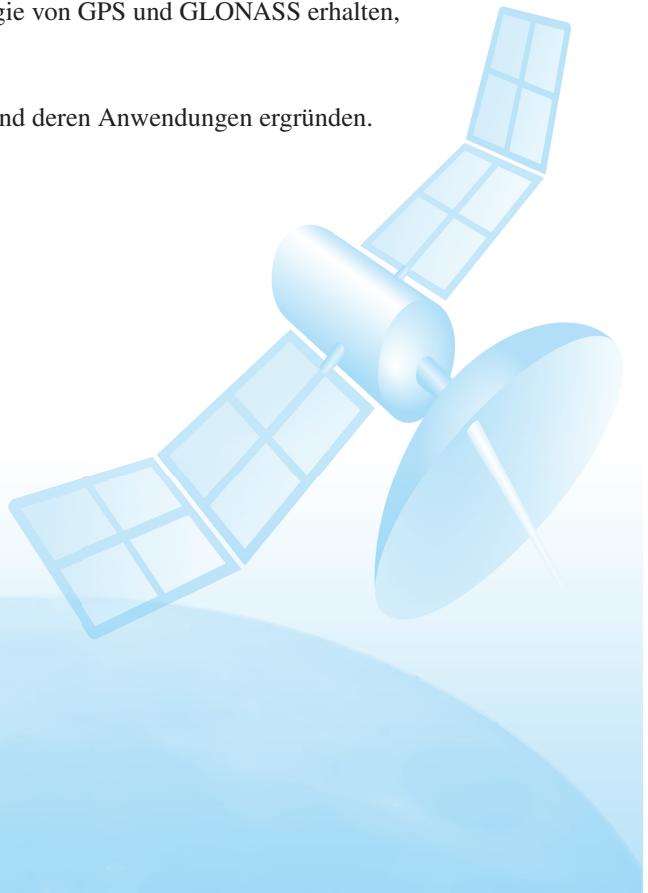
- Wir nehmen an, dass eine Nachricht korrekt übertragen wurde. Ein Maximum-Likelihood-Decoder handelt in diesem Fall, wie Sie es wahrscheinlich erwarten: Er gibt die empfangene Bitsequenz unverändert aus. Der Shannon-Decoder verfährt jedoch anders. Obwohl die Nachricht korrekt übertragen wurde, gibt er fast ausnahmslos ein anderes, und somit falsches Codewort aus. Begründen Sie, warum der Decoder so handelt.
- Macht die Tatsache, dass korrekt übertragene Nachrichten auf der Empfängerseite verfälscht werden, den Shannon-Decoder unbrauchbar?

# 8 Leitungscodierung und Modulation

---

In diesem Kapitel werden Sie ...

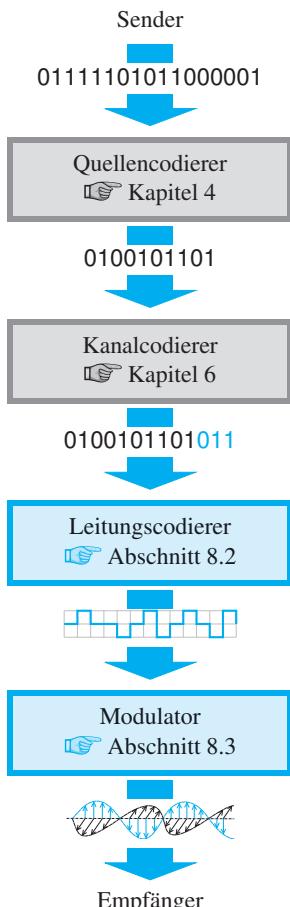
- die Aufgabe der Leitungscodierung verstehen,
- erfahren, wie sich Signale digital modulieren lassen,
- die gängigen Multiplexverfahren kennenlernen,
- einen Einblick in die Technologie von GPS und GLONASS erhalten,
- Pseudozufallsfolgen erzeugen,
- Gold-Folgen, Kasami-Folgen und deren Anwendungen ergründen.



## 8.1 Motivation

*„Nehmt aus der Welt die Elektrizität, und das Licht verschwindet; nehmst aus der Welt den lichttragenden Äther, und die elektrischen und magnetischen Kräfte können nicht mehr den Raum überschreiten.“*

Heinrich Hertz [41]



**Abb. 8.1:** Der Leitungscodierer bereitet den kanalcodierten Bitstrom für die Übertragung vor. Anschließend werden in einem nachgeschalteten Modulator die physikalischen Signale erzeugt.

Lassen wir die zurückliegenden Kapitel vor unserem geistigen Auge Revue passieren, so wird deutlich, dass wir dort fast ausschließlich auf einer logischen Ebene argumentiert haben. Nahezu alle Überlegungen basierten auf der unausgesprochenen Annahme, dass wir mit den Symbolen der Quellen- und Codealphabete frei agieren können, und entsprechend wenig Gedanken haben wir uns darüber gemacht, wie die erzeugten Symbolsequenzen die physikalische Übertragungsstrecke überwinden können. In diesem Kapitel nehmen wir uns dieses Umstands an und rücken die physikalischen Aspekte der Datenkommunikation in den Vordergrund.

In Abschnitt 8.2 nähern wir uns der physikalischen Ebene, indem wir die wohlvertraute Kette, bestehend aus einem Quellencodierer und einem Kanalcodierer, um ein weiteres Glied, einen sogenannten *Leitungscodierer*, ergänzen (Abbildung 8.1). Die erzeugten Signale sind an dieser Stelle immer noch digital, allerdings werden jetzt wichtige physikalische Parameter mitberücksichtigt, die bei der Kanalcodierung noch keine Rolle gespielt haben.

Den eigentlichen Sprung auf die physikalische Ebene vollziehen wir in Abschnitt 8.3. Dort beantworten wir die Frage, wie sich ein leitungscodierter Bitstrom so modulieren lässt, dass er von einem realen Übertragungsmedium transportiert werden kann.

Anschließend wenden wir uns in Abschnitt 8.4 der Frage zu, wie ein Übertragungsmedium von mehreren Kommunikationsteilnehmern gemeinsam genutzt werden kann. Neben den klassischen Verfahren, die auf der räumlichen, zeitlichen oder spektralen Teilnehmerseparation basieren, beschäftigen wir uns in Abschnitt 8.4.4 ausführlich mit dem Codemultiplexverfahren. Dieses ist nicht nur wegen seiner trickreichen Grundidee der intellektuelle Höhepunkt dieses Kapitels. Sein nichttriviales mathematisches Fundament führt uns auf direktem Wege zurück in die Theorie, und so werden wir uns am Ende dieses Kapitels auf der gleichen logischen Ebene wiederfinden, auf der wir unsere Reise begonnen haben. Dort schließt sich der Kreis.

## 8.2 Leitungscodierungen

### NRZ-Codierung

Die erste Leitungscodierung, die wir uns genauer ansehen, ist die *NRZ-Codierung*. NRZ ist die Abkürzung für den zunächst wenig aussagekräftigen Bezeichner *Non-Return-to-Zero*, dessen Bedeutung erst weiter unten klar werden wird, wenn wir die RZ-Codierungen besprechen.

Die Arbeitsweise eines NRZ-Encoders ist nahezu trivial: Er übersetzt den binären Bitstrom des Kanalcodierers eins zu eins in ein digitales Signal mit zwei Pegeln, + und -. Wird die 1 auf den +'-Pegel und die 0 auf den ,-'Pegel abgebildet, so sprechen wir von einer *positiven Logik*. Andernfalls liegt eine *negative Logik* vor (Abbildung 8.2).

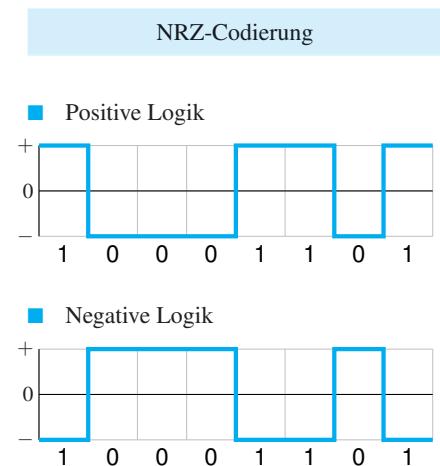
Eingesetzt wird die NRZ-Codierung beispielsweise an der seriellen RS232-Schnittstelle, die in den 60er-Jahren eingeführt wurde und bis heute von allen gängigen Computersystemen unterstützt wird. An dieser Schnittstelle wird der ,+'-Pegel durch eine Spannung im Bereich von +3 V bis +15 V und der ,-'-Pegel durch eine Spannung im Bereich von -3 V bis -15 V dargestellt. Für die Codierung wird eine negative Logik verwendet, d. h., die binäre 1 wird durch eine negative Spannung dargestellt. Im Folgenden werden wir die Eigenschaften der NRZ-Codierung genauer analysieren und dabei mehrere Kriterien formulieren, die bei der Konstruktion und der Bewertung von Leitungs-codes eine wichtige Rolle spielen.

### Bewertungskriterien

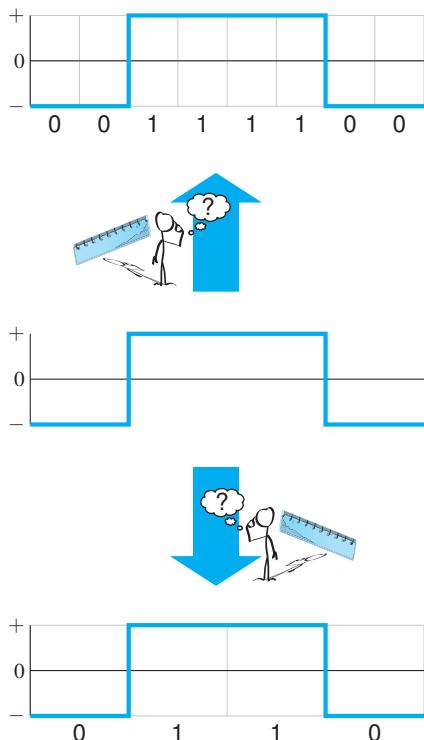
#### Selbsttaktung

Bei der Spezifikation des RS232-Protokolls wurde bewusst darauf verzichtet, für die Übertragung eine feste Sendegeschwindigkeit vorzuschreiben. Dies ermöglicht die Kommunikation mit unterschiedlichen Bitraten und offenbart gleichsam das erste Problem der NRZ-Codierung. Stellt sich an der Schnittstelle beispielsweise der in Abbildung 8.3 skizzierte Signalverlauf ein, so hat der Empfänger keine Möglichkeit, die Länge eines Bits auszumessen, aber genau dies wäre nötig, um aus den eingehenden Daten auf die Sendegeschwindigkeit zu schließen. Wir sagen, der NRZ-Code ist nicht *selbsttaktend*.

Tatsächlich ist es gar nicht so einfach, die Eigenschaft der Selbsttaktung formal zu definieren. Weiter unten werden wir einen selbsttaktenden Code kennenlernen, der sich durch das Senden einer speziell



**Abb. 8.2:** Die NRZ-Codierung bildet einen binären Datenstrom eins zu eins auf ein digitales Signal mit zwei Pegeln ab.



**Abb. 8.3:** Die NRZ-Codierung ist nicht selbttaktend. Aus dem in der Mitte dargestellten Signalverlauf kann das Taktsignal nicht eindeutig zurückgewonnen werden.

konstruierten Bitsequenz gewissermaßen austricksen lässt. Diese Sequenz verschiebt die Signalflanken derart, dass das Taktsignal, ähnlich wie bei der NRZ-Codierung, nicht mehr eindeutig ausgemessen werden kann. Um das Kriterium der Selbttaktung zu erfüllen, wird deshalb auch gar nicht gefordert, dass die Taktrückgewinnung für ausnahmslos alle Signalverläufe funktioniert. Es reicht aus, dass sie für jene gelingt, die von typischen Bitmustern erzeugt werden.

Als typische Bitmuster gelten dabei insbesondere lange Ketten, die ausschließlich das Symbol 0 oder ausschließlich das Symbol 1 enthalten. Erzeugt bereits eines dieser Bitmuster einen Signalverlauf, aus dem das Taktsignal nicht zurückgewonnen werden kann, so gilt der Code per se als nicht selbttaktend. Bei der NRZ-Codierung ist dies sogar für beide Bitmuster der Fall. Sie erzeugen einen konstanten Signalpegel, der keinerlei Rückschluss auf den Takt erlaubt.

### ■ Resynchronisation

In der Praxis werden Codes, die eine Taktrückgewinnung ermöglichen, selbst dann gerne eingesetzt, wenn der Empfänger die Sendegeschwindigkeit kennt und das Taktsignal daher gar nicht ausmessen muss. Der Grund hierfür ist, dass selbttaktende Codes immer auch *resynchronisierend* sind. Ganz allgemein stellt sich das Problem der Resynchronisation folgendermaßen dar: Bleibt das Sendedignal über einen langen Zeitraum konstant, so steigt die Wahrscheinlichkeit, dass der Empfänger die Synchronisation verliert und die Anzahl der empfangenen Bits falsch berechnet. Da die regelmäßigen Signalwechsel, die selbttaktende Codes für die implizite Übermittlung des Taktsignals vollziehen, zu einer automatischen Resynchronisation führen, ist jeder selbttaktende Code auch resynchronisierend.

In Abschnitt 8.2.3.1 werden wir auf die geschilderte Problematik erneut zu sprechen kommen. Dort werden wir zwei alternative Möglichkeiten präsentieren, um das Problem der Resynchronisation zu lösen.

### ■ Gleichstromfreiheit

Eine Leitungscodierung heißt *gleichstromfrei*, wenn das Integral über die Signalkurve nach oben und nach unten beschränkt ist. Etwa bildlicher lässt sich dies auch so formulieren: Beobachten wir den Signalverlauf eines gleichstromfreien Codes über einen längeren Zeitraum, so können wir in etwa genauso viele Ausschläge nach oben zählen wie nach unten.

Die NRZ-Codierung produziert nur dann einen gleichstromfreien Signalverlauf, wenn der Bitstrom genauso viele Nullen wie Einsen ent-

hält. Werden über einen längeren Zeitraum deutlich mehr Nullen als Einsen oder deutlich mehr Einsen als Nullen übertragen, so stellt sich ein einseitiger Stromfluss zwischen Sender und Empfänger ein. Dieser Gleichstromanteil hat mehrere Nachteile. Wird beispielsweise ein elektrischer Leiter verwendet, so lassen sich die Gleichstromanteile nur schwer über lange Strecken transportieren, da der elektrische Widerstand proportional mit der Leitungslänge wächst. Genauso problematisch ist der Einsatz von Transformatoren, die in der Praxis auf vielen Übertragungsstrecken zum Einsatz kommen. Diese können Gleichstromanteile gar nicht oder nur in sehr geringem Umfang übertragen.

Nachdem wir nun ein gutes Gespür dafür haben, welche Kriterien bei der Beurteilung einer Leitungscodierung eine Rolle spielen, ist es an der Zeit, nach Alternativen zu der in vielerlei Hinsicht verbesserungsbedürftigen NRZ-Codierung zu suchen.

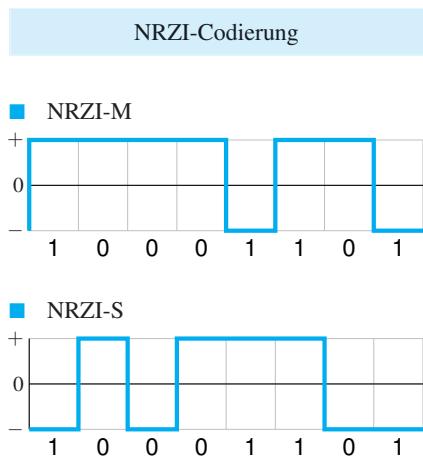
Wie beginnen in Abschnitt 8.2.1 mit der Vorstellung verschiedener Bitcodierungen, zu denen auch die oben diskutierte NRZ-Codierung gehört. Anschließend werden wir in Abschnitt 8.2.2 mehrere Alternativen diskutieren, die größere Bitblöcke gemeinsam codieren und damit eine noch stärkere Kontrolle über den generierten Signalverlauf ausüben.

## 8.2.1 Bitcodierungen

Von einer Bitcodierung sprechen wir immer dann, wenn sich jedes Segment der Signalkurve eindeutig einem gewissen Bit des Datenstroms zuordnen lässt, der in den Leitungscodierer eingespeist wurde. Die Signalkurve, die für ein einzelnes Bit erzeugt wird, muss dabei aber nicht die gleiche sein. So werden wir im nächsten Abschnitt die NRZI-Codierung kennenlernen, die den Signalpegel in Abhängigkeit des zuvor generierten Pegels variiert.

### NRZI-Codierung

Die Abkürzung NRZI steht für *Non-Return-to-Zero Inverted* und bedeutet, dass den Bits 0 und 1, anders als bei der NRZ-Codierung, keine festen Signalpegel mehr zugeordnet sind. Stattdessen wird eines der Bits durch einen Signalwechsel und das andere durch das Halten des aktuellen Signalpegels beschrieben.



**Abb. 8.4:** Die NRZI-Codierung benutzt keine feste Signalzuordnung. Als Unterscheidungsmerkmal dient die Information, ob ein Signal wechselt oder seinen aktuellen Pegel behält.

Die Signalverläufe in Abbildung 8.4 zeigen, in welchen Varianten die NRZI-Codierung in der Praxis vorkommt. Bei der NRZI-M-Codierung entspricht der Flankenwechsel einer Eins (*Mark*) und bei der NRZI-S-Codierung einer Null (*Space*). In der Literatur werden für die NRZI-M- und die NRZI-S-Codierung auch gerne die Abkürzungen NRZ-M und NRZ-S verwendet. Achten Sie in diesen Fällen darauf, die Codierungen nicht mit den beiden NRZ-Varianten zu verwechseln, die wir in Abbildung 8.2 kennengelernt haben.

Bei der Interpretation der Signalverläufe darf nicht vergessen werden, dass der Pegel innerhalb eines Takts von dem Pegel des Vorgängertakts abhängt. Um überhaupt ein Signal zeichnen zu können, sind wir in Abbildung 8.4 davon ausgegangen, dass den dargestellten Pegeln ein negativer Pegel vorausgegangen ist. Unter der Annahme eines positiven Vorgängerpegels hätten wir die beiden Signalverläufe invertiert einzeichnen müssen.

Beurteilen wir die NRZI-Codierung im Hinblick auf die oben aufgestellten Kriterien, so schneidet sie im Vergleich zur NRZ-Codierung nur unwesentlich besser ab. Genau wie die NRZ-Codierung ist auch die NRZI-Codierung weder gleichstromfrei noch selbsttaktend. Zwar führen bei der NRZI-M-Codierung konsekutiv gesendete Einsen jetzt zu permanenten Signalwechseln, bei konsekutiv gesendeten Nullen bleiben allerdings jegliche Pegeländerung aus. Die NRZI-S-Codierung verhält sich analog. Hier führen konsekutiv gesendete Einsen zu einem Ausbleiben der Signalwechsel.

Eine Anwendung der NRZI-Codierung haben wir bereits in Abschnitt 6.4.5.4 kennengelernt, als wir uns mit dem Aufbau optischer Speichermedien beschäftigt haben. Dort haben wir gesehen, dass eine 1 auf der Oberfläche einer CD, DVD oder BD durch einen Wechsel von *Pit* nach *Land* oder einen Wechsel von *Land* nach *Pit* dargestellt wird. Entsprechend wird eine 0 codiert, indem das Höhenprofil des Vorgängerborts beibehalten wird. Dies ist exakt das Prinzip, das der NRZI-M-Codierung zugrunde liegt. Folglich sind auch die optischen Datenspeicher von dem Problem der NRZI-M-Codierung betroffen, dass konsekutive Nullen keine Flanken verursachen. Abhilfe schafft dort die EFM-Codierung, die wir ebenfalls in Abschnitt 6.4.5.4 besprochen haben. Diese stellt sicher, dass spätestens nach 10 konsekutiven Nullen eine Eins folgt und sich das Höhenprofil damit nach maximal 10 Schritten ändert.

Auch in anderen Bereichen der Computertechnik wird gerne auf die NRZI-Codierung zurückgegriffen. Beispiele sind die Datenkommuni-

kation über die USB-Schnittstelle oder die Datenspeicherung auf magnetischen Medien.

### AMI-Codierung

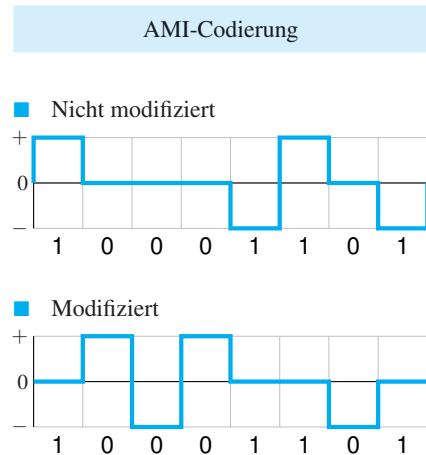
Eine Codierung, die im Gegensatz zu den bisher betrachteten das Kriterium der Gleichstromfreiheit erfüllt, ist die AMI-Codierung (Abbildung 8.5). Die Abkürzung AMI steht für *Alternate Mark Inversion* und beschreibt treffend die charakteristische Eigenschaft dieser Codierung: Eines der Bits – normalerweise die 1 – wird mit einem alternierenden Signalpegel codiert, der kontinuierlich zwischen  $+^\circ$  und  $-^\circ$  hin und her wechselt. Der obere Signalverlauf in Abbildung 8.5 demonstriert die AMI-Codierung am Beispiel einer konkreten Bitsequenz. Die erste 1 wird durch einen positiven Ausschlag codiert, die zweite durch einen negativen und so fort. Der untere Signalverlauf entspricht der *modifizierten AMI-Codierung*. Bei dieser Variante wird eine 0 durch einen alternierenden Pegelausschlag und eine 1 durch den Nullpegel codiert.

Im Gegensatz zu den bisher betrachteten Beispielen verwendet die AMI-Codierung drei Signalpegel ( $0^\circ$ ,  $+^\circ$  und  $-^\circ$ ) und gehört damit zu den *ternären* Codierungen. Es ist ein besonderes Merkmal der AMI-Codierung, dass die drei Signalpegel nicht unabhängig voneinander auf dem Bus erscheinen können: Es gibt zu jedem Zeitpunkt immer nur zwei Möglichkeiten. Das bedeutet, dass die AMI-Codierung zwar drei Pegelzustände unterscheidet, in jedem Takt aber trotzdem nur ein einziges Bit codiert. In der Literatur werden solche Codierungen häufig als *pseudoternäre Codierungen* bezeichnet.

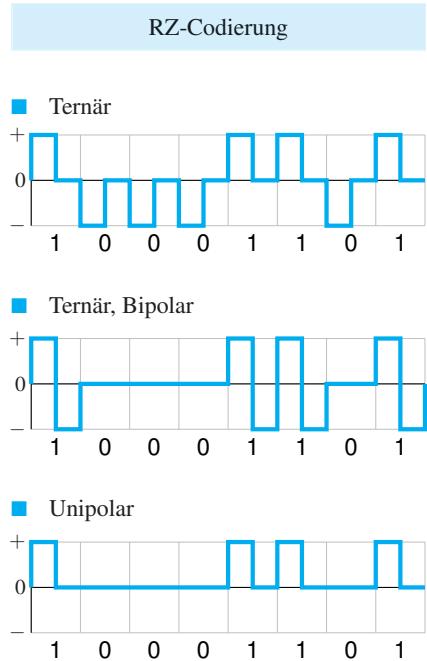
Auch der AMI-Code ist nicht selbttaktend, da konsekutiv gesendete Nullen zu keinem Signalwechsel führen. Dennoch ist die ternäre Codierung an dieser Stelle vorteilhaft. Sobald nämlich eine 1, oder bei der modifizierten Variante eine 0, übertragen wird, entsteht ein Pegelausschlag, der für die exakte Ausmessung der Taktlänge verwendet werden kann. Verantwortlich hierfür ist die Tatsache, dass der Plus- oder Minuspegel bei der AMI-Codierung immer genau für einen Takt gehalten wird und danach entweder auf den Nullpegel oder den komplementären Pegel wechselt.

### RZ-Codierung

Ein weiterer Vertreter aus der Gruppe der ternären Codierungen ist die *RZ-Codierung*, deren Signalverlauf in Abbildung 8.6 (oben) dargestellt



**Abb. 8.5:** Die AMI-Codierung ist gleichstromfrei. Die alternierenden Pegelausschläge sorgen dafür, dass das Integral unter der Signalkurve beschränkt bleibt.



**Abb. 8.6:** Die RZ-Codierung ist ein selbsttaktender Code. Erkauft wird diese Eigenschaft durch eine Verdoppelung der Frequenz.

ist. Hier werden die Bitwerte 0 und 1 durch einen Pegelausschlag codiert, der über eine halbe Taktperiode konstant bleibt und danach auf den Nullpegel zurückfällt. Legen wir eine positive Logik zugrunde, so wird eine 1 durch einen Ausschlag in die positive Richtung und eine 0 durch einen Ausschlag in die negative Richtung dargestellt. Bei der negativen Logik ist die Zuordnung umgekehrt.

Die RZ-Codierung ermöglicht eine sehr einfache Taktekonstruktion. Da der Signalpegel, unabhängig von dem codierten Bit, in der Mitte einer Taktpause auf den Nullpegel zurückfällt, lässt sich die Länge eines Takts zu jeder Zeit problemlos ausmessen. Diese spezielle Eigenschaft des Signalpegels zeichnet auch für den Namen dieser Codierung verantwortlich: RZ ist die Abkürzung für *Return to Zero*. An dieser Stelle erklären sich auch die Namen der NRZ- und NRZI-Codierungen, die wir weiter oben eingeführt haben. Es ist der fehlende Flankenwechsel in der Mitte der Taktpause, der für den Namen *Non-Return-to-Zero* bzw. *Non-Return-to-Zero-Inverted* steht.

Gleichstromfrei ist die RZ-Codierung nicht. Werden über einen längeren Zeitraum mehr Nullen als Einsen oder mehr Einsen als Nullen gesendet, so fließt ein Gleichstrom in eine der beiden Richtungen.

Eine Variante, die das Problem der Gleichstromfreiheit löst, ist die *bipolare RZ-Codierung*. Diese stellt eine 1, wie in der Mitte von Abbildung 8.6 zu sehen ist, durch eine Signalkurve dar, die in der ersten Takthälfte einen positiven und in der zweiten Takthälfte einen negativen Ausschlag zeigt. Bleibt das Signal während einer vollständigen Taktpause neutral, so wird eine binäre 0 codiert. Hierdurch verliert die bipolare RZ-Codierung die Eigenschaft, selbsttaktend zu sein, da längere Nullerketten keine Signalwechsel mehr verursachen.

Eine weitere Variante ist die *unipolare RZ-Codierung*, die ausschließlich die beiden Pegel „0“ und „+“ verwendet (Abbildung 8.6 unten). Den Signalverlauf der unipolaren RZ-Codierung können wir ganz einfach aus dem Signalverlauf der anderen beiden Varianten konstruieren, indem wir alle negativen Pegel auf die Nulllinie verschieben. Die unipolare Codierung hat den Vorteil, dass sie mit zwei Signalpegeln auskommt, vereint aber ansonsten die Nachteile der anderen beiden. Sie ist weder selbsttaktend noch gleichstromfrei.

### Manchester-Codierung

Die oben geführte Diskussion hat gezeigt, dass die RZ-Codierung in ihrer ursprünglichen Form selbsttaktend, aber nicht gleichstromfrei ist,

und sich die bipolare Variante genau gegensätzlich verhält. Indem wir die Abbildungsvorschrift ein wenig abwandeln, können wir die bipolare RZ-Codierung in eine selbsttaktende Codierung überführen, ohne die Eigenschaft der Gleichstromfreiheit aufgeben zu müssen.

Dies gelingt, indem die beiden Bitwerte 0 und 1 durch ein Signal codiert werden, das in beiden Takthälften einen jeweils entgegengesetzten Wert annimmt. Welches Bit codiert wird, hängt davon ab, ob die Flanke in der Mitte eines Takts steigend oder fallend ist. Wird eine steigende Flanke, wie in Abbildung 8.7 (oben), mit der 1 und eine fallende Flanke mit der 0 assoziiert, so entsteht die *Manchester-Codierung*, wie sie in der IEEE-Spezifikation 802.3 für die Übertragung im 10-Mbit/s-Ethernet-Protokoll festgelegt ist. Die umgekehrte Zuordnung ergibt die *Biphase-L-Codierung*, die in der Literatur gerne auch als *Manchester-II-Codierung* bezeichnet wird (Abbildung 8.7 Mitte).

Bei Token-Ring-Netzwerken wird die Manchester-Codierung in einer differenziellen Variante benutzt, die auf eine eindeutige Zuordnung zwischen den beiden Flankenrichtungen und den Bitwerten 0 und 1 verzichtet (Abbildung 8.7 unten). Welches Bit codiert wird, hängt hier davon ab, ob die Signalflanke im Vergleich zum Vorgängertakt die Richtung ändert oder gleich bleibt.

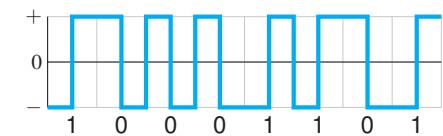
Beachten Sie, dass wir den Signalverlauf der differenziellen Manchester-Codierung, genau wie im Fall der NRZI-Codierungen, nur dann angeben können, wenn wir wissen, ob der ersten eingezeichneten Flanke eine positive oder eine negative Flanke vorausgegangen ist. Dass wir den eingezeichneten Signalverlauf mit einer fallenden Flanke begonnen haben, zeigt, dass wir von einer steigenden Vorgängerflanke ausgegangen sind. Unter der Annahme einer fallenden Vorgängerflanke hätten wir den Signalverlauf invertiert einzeichnen müssen.

## Biphase-Mark- und Biphase-Space-Codierung

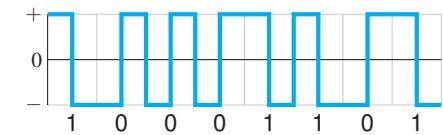
Zum Schluss wollen wir unser Augenmerk auf zwei Codierungen richten, die eng mit dem differenziellen Manchester-Code verwandt sind: die *Biphase-M-Codierung* und die *Biphase-S-Codierung*. Ihre Signalverläufe weisen die Besonderheit auf, dass wir den garantierten Flankenwechsel nicht wie bei der differenziellen Manchester-Codierung in der Mitte, sondern am Anfang einer Taktfalte beobachten können. Über das codierte Bit bestimmt der Signalverlauf in der Mitte eines Takts. Findet dort ein Pegelwechsel statt, so wird bei der Biphase-M-Codierung eine 1 (*Mark*) signalisiert. Bleibt das Signal dagegen über

### Manchester-Codierung

■ Manchester (nach IEEE 802.3)



■ Manchester II



■ Differential Manchester

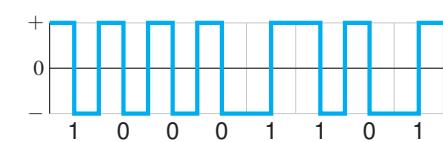
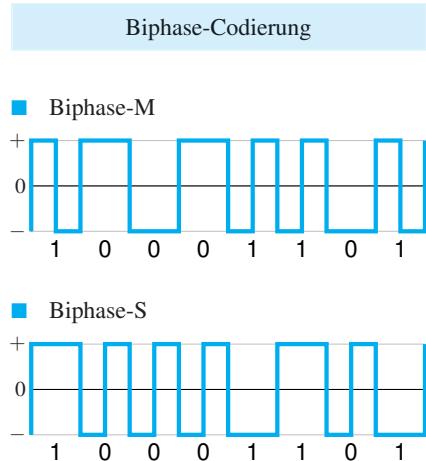


Abb. 8.7: Die Manchester-Codierung ist sowohl gleichstromfrei als auch selbsttaktend.



Auch wenn die Manchester-Codierung moderner wirkt als ihre Kontrahenten, ist sie vergleichsweise alt. Ihre historische Spur führt uns in die Pionierzeit der Computertechnik zurück, als Ende der 40er-Jahre an der Victoria University of Manchester einer der ersten großen Rechenanlagen der Welt in Betrieb genommen wurde: die legendäre *Manchester Mark I*.

Für die Zwischenspeicherung von Daten war der Rechnerkoloss mit einem primitiven Vorgänger der heutigen Festplatte ausgestattet: der *Magnetic Drum*. Die Codierung, die damals für die Speicherung der Daten auf diesem Medium zum Einsatz kam, wurde von den Ingenieuren ad hoc als Manchester-Codierung bezeichnet, und genauso heißt sie noch heute.



**Abb. 8.8:** Anders als bei den Manchester-Codierungen tritt bei der Biphase-M- und der Biphase-S-Codierung ein garantierter Signalwechsel nicht in der Mitte, sondern am Anfang eines Takts auf.

die volle Taktphase konstant, so entspricht dies einer 0. Bei der Biphase-S-Codierung wird die umgekehrte Zuordnung gewählt. Dort signalisiert ein zusätzlicher Flankenwechsel eine 0 (*Space*). Eingesetzt werden diese Codierungen unter anderem im Audiobereich, z. B. an der AES-3-Schnittstelle.

Abbildung 8.9 zeigt, wie eng diese Codierungen mit der differenziellen Manchester-Codierung zusammenhängen. Verschieben wir ein Biphase-S-codiertes Signal um einen halben Takt nach links, so entsteht exakt der Signalverlauf, den die differenzielle Manchester-Codierung hervorgebracht hätte.

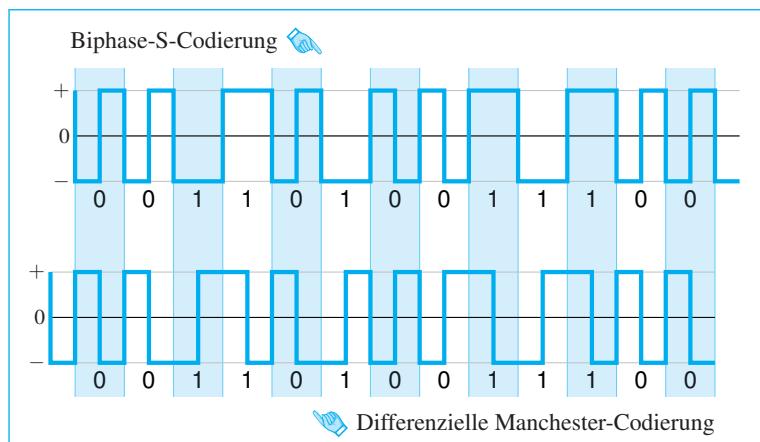
## 8.2.2 Blockcodes

In diesem Abschnitt betrachten wir Leitungscodierungen, die nicht einzelne Bits, sondern Bitblöcke auf einen bestimmten Signalverlauf abbilden. Für die Klassifikation solcher Codes hat sich das Namensschema

$$kX/nY$$

etabliert, das Auskunft über die Blockgrößen sowie das Quellen- und Codealphabet gibt. Beispielsweise bedeutet 2B/1Q, dass

- jeweils zwei 2B/1Q
- binäre Symbole 2B/1Q
- durch einen 2B/1Q



**Abb. 8.9:** Versetzen wir ein Biphase-S-codiertes Signal um einen halben Takt nach links, so entsteht der Signalverlauf des differenziellen Manchester-Codes.

- quaternären Signalpegel dargestellt werden.

2B/1Q

### 2B/1Q-Codierung

Ein Blockcode, der diesem Schema entspricht, ist in Abbildung 8.10 zu sehen. Auch wenn die Zuordnung der vier Bitkombinationen zu den vier Signalpeglern auf den ersten Blick willkürlich wirkt, wurde sie mit Bedacht gewählt. Wechseln wir nämlich von einem Signalpegel zu einem seiner Nachbarpegel, so ändert sich in der codierten Bitkombination immer genau 1 Bit. Die gewählte Anordnung erinnert an den Gray-Code und trägt zur Reduzierung der fehlerhaft übertragenen Bits bei: Wird ein Signalpegel auf der Empfängerseite falsch decodiert und versehentlich mit einem seiner Nachbarpegel assoziiert, so wirkt sich der Fehler immer nur auf ein einzelnes Bit aus.

Für die Bewertung einer blockweise arbeitenden Leitungscodierung wird neben den uns bereits bekannten Kriterien der Selbsttaktung und der Gleichstromfreiheit auch gerne die Coderate betrachtet. Für einen  $kX/nY$ -Leitungscode lässt sie sich sehr einfach berechnen: Sie ist der Quotient

$$\frac{k}{n},$$

der für die oben betrachtete 2B/1Q-Codierung beispielsweise den Wert 2 ergibt. Dass wir eine Coderate größer als 1 erzielen können, ist nur auf den ersten Blick ungewöhnlich: Sie ist deshalb so groß, weil wir binäre Signale auf quaternäre Signale abbilden und somit mehrere Bits hinter einem einzigen Signalpegel verstecken können.

Eine Leitungscodierung mit der Coderate  $\frac{4}{5}$  ist in Abbildung 8.11 zu sehen. Hierbei handelt es sich um eine 4B/5B-Leitungscodierung, die jeweils 4 Bits auf einen Block der Länge 5 abbildet. Ein Blick auf die Codewörter macht klar, dass die Biterweiterung zum Zwecke der Resynchronisation eingeführt wurde: Sie stellt sicher, dass der ,-'-Pegel für höchstens 3 Zeiteinheiten und der ,+'-Pegel für höchstens 8 Zeiteinheiten konstant bleiben kann. Gleichstromfrei ist die Codierung nicht. Beispielsweise wird aus einem Bitstrom, der ausschließlich aus Nullen besteht, ein Signal erzeugt, das viermal häufiger den ,+'-Pegel annimmt als den ,-'-Pegel.

Eingesetzt wird die 4B/5B-Leitungscodierung unter anderem an der 100-Mbit/s-Ethernet-Schnittstelle. Die heute üblicherweise verwendeten Gigabit-Schnittstellen arbeiten mit Leitungscodes, die noch komplizierter sind. Unter anderem kommt dort eine 8B/10B-Codierung zum Einsatz, die durch ein ausgeklügeltes Balancierungsschema nicht nur selbsttaktend, sondern auch gleichstromfrei ist. Eine andere ist die 64B/66B-Codierung, die jeweils 64 Bits auf einmal bearbeitet. Sie ist ebenfalls selbsttaktend und gleichstromfrei.

### Zuordnung

$00 \mapsto --$   $01 \mapsto -$   $10 \mapsto ++$   $11 \mapsto +$

### Signalverlauf

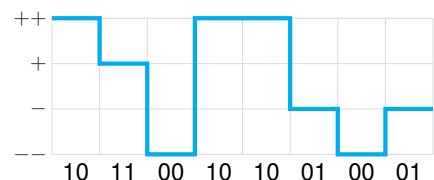


Abb. 8.10: Die dargestellte 2B/1Q-Codierung bildet jeweils zwei Bits auf einen quaternären Pegel ab.

### 4B/5B-Leitungscodierung

0000	$\mapsto$	++++-
0001	$\mapsto$	-+---+
0010	$\mapsto$	+--++-
0011	$\mapsto$	+--+--
0100	$\mapsto$	-++-+-
0101	$\mapsto$	-++-++
0110	$\mapsto$	-+++-+
0111	$\mapsto$	-+++-+
0000	$\mapsto$	+-+-+-
0001	$\mapsto$	+-+-++
0010	$\mapsto$	+-++-+
0011	$\mapsto$	+-+++-
0100	$\mapsto$	+++-+-
0101	$\mapsto$	+++-++
0110	$\mapsto$	+++-+-
0111	$\mapsto$	+++-+-

Abb. 8.11: Die gezeigte 4B/5B-Leitungscodierung kommt unter anderem bei der Übertragung an der 100-MBit/s-Ethernet-Schnittstelle zum Einsatz. Sie weist eine Coderate von  $\frac{4}{5}$  auf, ist selbsttaktend, aber nicht gleichstromfrei.

	Tabelle 0		Tabelle 1		Tabelle 2		Tabelle 3	
Binär	Ternär	Folgetabelle	Ternär	Folgetabelle	Ternär	Folgetabelle	Ternär	Folgetabelle
0000	+0+	2	0–0	0	0–0	1	0–0	2
0001	0–+	0	0–+	1	0–+	2	0–+	3
0010	+-0	0	+-0	1	+-0	2	+-0	3
0011	00+	1	00+	2	00+	3	--0	1
0100	-+0	0	-+0	1	-+0	2	-+0	3
0101	0++	2	-00	0	-00	1	-00	2
0110	-++	1	-++	2	--+	1	--+	2
0111	-0+	0	-0+	1	-0+	2	-0+	3
1000	+00	1	+00	2	+00	3	0--	1
1001	++-	1	++-	2	++-	3	----	0
1010	++-	1	++-	2	++-	1	++-	2
1011	+0-	0	+0-	1	+0-	2	+0-	3
1100	+++	3	-+-	0	-+-	1	-+-	2
1101	0+0	1	0+0	2	0+0	3	-0-	1
1110	0+-	0	0+-	1	0+-	2	0+-	3
1111	++0	2	00-	0	00-	1	00-	2

**Tab. 8.1:** Die MMS43-Codierung verwendet zur Ausbalancierung des Gleichstromanteils vier verschiedene Codetabellen.

### 8.2.2.1 MMS43-Codierung

Wie ein gleichstromfreier Signalverlauf durch das geschickte Ausbalancieren der Codewörter erzeugt werden kann, wollen wir am Beispiel der MMS43-Codierung herausarbeiten (*Modified Monitored Sum 43*). Dabei handelt es sich um eine 4B/3T-Codierung, die vor allem durch ihre Verwendung im digitalen ISDN-Netz bekannt geworden ist. Die Codierung bildet jeweils 4 Bits auf 3 Ternärpegel ab, wobei für jede der 16 möglichen Bitkombinationen 4 verschiedene Codewörter zur Verfügung stehen. Tabelle 8.1 gibt Auskunft darüber, um welche Codewörter es sich im Einzelnen handelt.

Das erste Codewort wird immer aus Tabelle 0 ausgelesen; danach geht es in jener Tabelle weiter, die neben dem Codewort als Folgetabelle eingetragen ist. Das bedeutet, dass wir, obwohl vier verschiedene Codewörter zur Verfügung stehen, keine freie Wahl haben: Das Codewort, auf das eine Bitsequenz abgebildet wird, ist zu jeder Zeit eindeutig bestimmt.

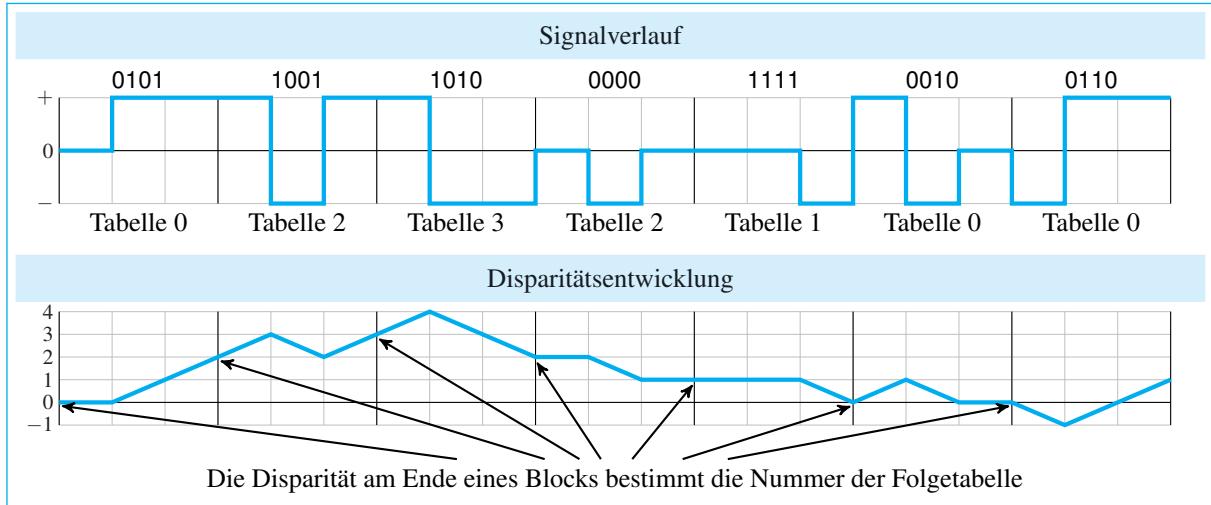


Abb. 8.12: Die MMS43-Codierung bildet jeweils 4 Bits auf 3 Ternärpegel ab.

Abbildung 8.12 demonstriert die Codierung anhand eines konkreten Beispiels. Unter der Signalkurve ist für jeden Teilstück die Tabelle vermerkt, aus der der zu produzierende Signalverlauf ausgelesen wurde. Die zweite in Abbildung 8.12 gezeigte Kurve gibt Auskunft über die momentane *Disparität*. Sie ist das Integral über die Signalkurve und entspricht der Differenz zwischen der Anzahl der positiven Pegel und der Anzahl der negativen Pegel.

Um zu verdeutlichen, in welchem Umfang die Codewörter zu einer Verschiebung der Paritätskurve beitragen, sind die Zellen in Tabelle 8.1 unterschiedlich eingefärbt. Die hell eingefärbten Zellen enthalten Codewörter, in denen die Anzahl der positiven und der negativen Pegel übereinstimmen; diese Codewörter führen zu keiner nachhaltigen Verschiebung der Disparität. Die dunkel eingefärbten Zellen enthalten die Codewörter mit dem größten Missverhältnis zwischen den positiven und den negativen Signalpegeln. Diese Codewörter können die Disparitätskurve um bis zu 3 Zähler nach oben oder unten verändern.

Obwohl in den meisten Codewörtern mehr positive oder mehr negative Pegel vorkommen, stellt die Codierungsvorschrift sicher, dass die Disparität, unabhängig von der verarbeiteten Bitsequenz, niemals kleiner als  $-1$  und niemals größer als  $4$  werden kann. Beziehen wir ausschließlich die Codewortgrenzen in die Betrachtung ein, so wird die Disparität niemals kleiner als  $0$  und niemals größer als  $3$ . Warum dies so ist, wird sofort klar, wenn wir Tabelle 8.1 ein wenig umsortieren und anstellen:

	Tabelle 0		Tabelle 1		Tabelle 2		Tabelle 3	
Binär	Ternär	Disparität	Ternär	Disparität	Ternär	Disparität	Ternär	Disparität
0001	0-+	±0	0-+	±0	0-+	±0	0-+	±0
0111	-0+	±0	-0+	±0	-0+	±0	-0+	±0
0100	-+0	±0	-+0	±0	-+0	±0	-+0	±0
0010	+ -0	±0	+ -0	±0	+ -0	±0	+ -0	±0
1011	+0-	±0	+0-	±0	+0-	±0	+0-	±0
1110	0+-	±0	0+-	±0	0+-	±0	0+-	±0
1001	++-	+1	++-	+1	++-	+1	---	-3
0011	00+	+1	00+	+1	00+	+1	--0	-2
1101	0+0	+1	0+0	+1	0+0	+1	-0-	-2
1000	+00	+1	+00	+1	+00	+1	0--	-1
0110	-++	+1	-++	+1	--+	-1	--+	-1
1010	++-	+1	++-	+1	---	-1	---	-1
1111	++0	+2	00-	-1	00-	-1	00-	-1
0000	+0+	+2	0-0	-1	0-0	-1	0-0	-1
0101	0++	+2	-00	-1	-00	-1	-00	-1
1100	+++	+3	-+-	-1	-+-	-1	-+-	-1

**Tab. 8.2:** Die Disparität der einzelnen Codewörter deckt die Bedeutung der Codetabellen auf. Ist  $d$  die aktuelle Disparität des bisher produzierten Datenstroms, so wird das nächste Codewort aus Tabelle  $d$  ausgelesen.

le der Folgetabelle die Disparität der einzelnen Codewörter eintragen. Das Ergebnis ist in Tabelle 8.2 zu sehen. Es zeigt, dass sich die Folgetabelle direkt berechnen lässt, indem die Disparität des Codeworts ganz einfach auf die aktuelle Tabellenummer addiert wird. Damit ist auch klar, welche Bedeutung den einzelnen Tabellen zukommt und warum wir die Nummerierung mit 0 begonnen haben: Die MMS43-Codierung ist so gestaltet, dass die Disparität an einer Blockgrenze stets der Tabellenummer entspricht, aus der das Folgecodewort ausgelesen wird.

### 8.2.2.2 RLL-Codierungen

Eine wichtige Unterklasse der Leitungscodierungen bilden die *RLL-Codierungen*. Die Abkürzung RLL steht für *Run Length Limited* („lauf-längenbeschränkt“) und bedeutet, dass die erzeugten Bitfolgen Einschränkungen in Bezug auf die darin enthaltenen Einser- und Nullerketten unterliegen. Für die Klassifikation von RLL-Codes wird gerne

auf das Namensschema

$(d, k)$ -RLL

zurückgegriffen, wobei die beiden Parameter  $d$  und  $k$  das Folgende bedeuten:

- Zwischen zwei Einsen gibt es mindestens  $d$  Nullen.
- Zwischen zwei Einsen gibt es höchstens  $k$  Nullen.

Eine wichtige RLL-Codierung kennen wir bereits aus Abschnitt 6.4.5.4. Gemeint ist die *EFM-Codierung*, die für die Datenspeicherung auf CDs, DVDs und BDs verwendet wird. Wir erinnern uns: Die EFM-Codewörter sind so beschaffen, dass zwei Einsen des erzeugten Datenstroms durch mindestens zwei und höchstens zehn Nullen getrennt sind. In der gerade eingeführten Nomenklatur ist die EFM-Codierung somit eine  $(2,10)$ -RLL-Codierung.

Dass wir im Zusammenhang mit den RLL-Codewörtern die Symbole 0 und 1 verwenden und nicht, wie bei den anderen bisher vorgestellten Codierungen, direkt auf die Signalpegel  $+$  und  $-$  zurückgreifen, ist kein Zufall. In der Praxis werden RLL-codierte Bitsequenzen in den meisten Fällen weiterverarbeitet und daher erst im Nachgang, zumeist mit einem NRZI-Encoder, in einen Signalverlauf übersetzt. Bei den optischen Speichermedien konnten wir genau dies beobachten. In Abschnitt 6.4.5.4 haben wir gesehen, dass eine 1 des EFM-codierten Datenstroms durch einen Wechsel von *Pit* nach *Land* oder einen Wechsel von *Land* nach *Pit* auf dem physikalischen Medium repräsentiert wird, und ein konstant bleibendes Höhenprofil eine 0 bedeutet.

## Digitale Frequenzmodulation

Eine sehr einfache lauflängenbeschränkte Codierung ist die  $(0,1)$ -RLL-Codierung in Abbildung 8.13. Ihre Codewörter sind so aufgebaut, dass der Encoder zwar beliebig lange Einserketten, aber niemals mehr als eine Null hintereinander erzeugen kann. Kombinieren wir diese Codierung mit einem nachgeschalteten NRZI-Encoder, so entsteht ein Signalverlauf, der in der Literatur als *digitale Frequenzmodulation* beschrieben wird.

Ein Blick auf die Signalkurve in Abbildung 8.14 zeigt, dass der Name treffend gewählt ist: Wir können dort ein kontinuierlich wechselndes Signal beobachten, das im Falle einer Eins doppelt so schnell schwingt

$(0,1)$ -RLL-Codierung

### Zuordnung

$$\begin{array}{rcl} 0 & \mapsto & 10 \\ 1 & \mapsto & 11 \end{array}$$

### Signalverlauf

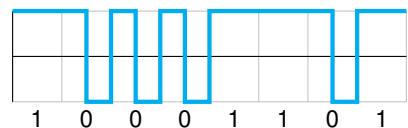


Abb. 8.13: Die  $(0,1)$ -RLL-Codierung stellt sicher, dass zwischen zwei Einsen höchstens eine Null auftritt.

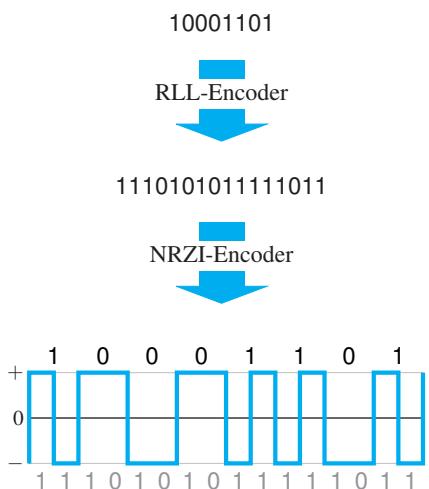


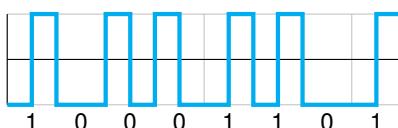
Abb. 8.14: In Kombination mit einem NRZI-Encoder entsteht eine Codierung, die gerne als *digitale Frequenzmodulation*, kurz FM, bezeichnet wird. Sie ist mit der Biphase-M-Codierung aus Abbildung 8.8 identisch.

## (1,3)-RLL-Codierung

## ■ Zuordnung

$$\begin{array}{l} 0 \mapsto x0 \\ 1 \mapsto 01 \end{array}$$

## ■ Signalverlauf



**Abb. 8.15:** Die modifizierte Frequenzmodulation (MFM)

wie im Falle einer Null. Wenn Ihnen der Signalverlauf bekannt kommt, dann haben Sie gut aufgepasst! Ein Blick zurück auf Abbildung 8.8 zeigt, dass wir diese Art der Codierung schon längst kennen. Die digitale Frequenzmodulation ist mit der Biphase-M-Codierung identisch, die wir auf Seite 574 besprochen haben.

Früher wurde die digitale Frequenzmodulation gerne für die Datenspeicherung auf Magnetbändern und Disketten verwendet. Dort wurde sie später durch die *modifizierte Frequenzmodulation* (MFM) ersetzt, die auch bei den ersten Festplattenlaufwerken zum Einsatz kam.

### Modifizierte digitale Frequenzmodulation

In der oben eingeführten Nomenklatur verbirgt sich hinter der modifizierten digitalen Frequenzmodulation eine (1,3)-RLL-Codierung, d. h., zwei Einsen sind durch mindestens eine, aber durch höchstens drei Nullen voneinander getrennt. Dies gelingt, indem für das Bit 0 zwei Codewörter vorgehalten werden, die sich in dem ersten Bit ( $x$ ) unterscheiden. Ist der Vorgängerpegel positiv, so wird  $x$  auf 0 und andernfalls auf 1 gesetzt. Kurzum:  $x$  ist das Komplement des Vorgägerbits. Abbildung 8.15 demonstriert an einem konkreten Beispiel, welchen Signalverlauf die MFM-Codierung produziert.

Nicht selten werden RLL-Codierungen in einer tabellarischen Form angegeben, wie sie exemplarisch in Abbildung 8.16 zu sehen ist. Auf den ersten Blick wirkt die Darstellung gewöhnungsbedürftig, und dies hat einen triftigen Grund. Da die angegebenen Codewörter unterschiedlich lang sind, handelt es sich nicht um klassische Blockcodierungen,

(1,7)-RLL-Codierung	(2,7)-RLL-Codierung
0000 $\mapsto$ 101000	000 $\mapsto$ 000100
0001 $\mapsto$ 100000	0010 $\mapsto$ 00100100
0010 $\mapsto$ 101001	0011 $\mapsto$ 00001000
0011 $\mapsto$ 101010	010 $\mapsto$ 100100
01 $\mapsto$ 100	011 $\mapsto$ 001000
1000 $\mapsto$ 001000	10 $\mapsto$ 0100
1001 $\mapsto$ 010000	11 $\mapsto$ 1000
1010 $\mapsto$ 001001	
1011 $\mapsto$ 001010	
11 $\mapsto$ 010	

**Abb. 8.16:** Zwei weitere RLL-Codierungen. Die linke bildet jeweils 4 Bits auf 6 Bits und die rechte jeweils 12 Bits auf 24 Bits ab.

die jedem Eingabeblock immer den gleichen Ausgabeblock zuweisen. Beispielsweise bildet die (1,7)-RLL-Codierung die Bitsequenz 0110 manchmal auf 100001 und manchmal auf 100010 ab. Die zweite Variante wird immer dann gewählt, wenn die Eingabesequenz mit 01 fortgesetzt wird.

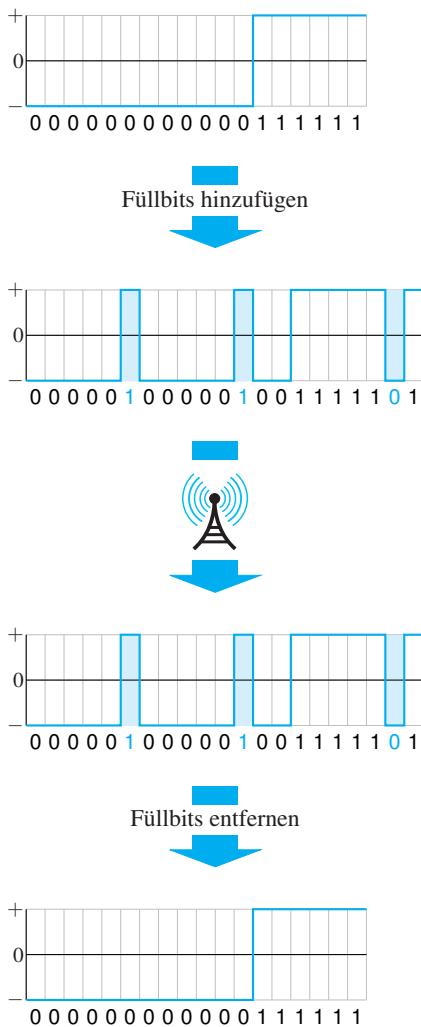
Dass die (1,7)-RLL-Codierung und die (2,7)-RLL-Codierung aus Abbildung 8.16 dennoch als Blockcodierungen bezeichnet werden, hat ebenfalls einen triftigen Grund: Beide erfüllen die für Blockcodierungen typische Eigenschaft, für jedes Eingabebit immer gleich viele Ausgabebits zu produzieren.

### GCR-Codierung

In der linken Spalte in Abbildung 8.17 ist eine (0,2)-RLL-Codierung zu sehen, die jeweils 4 Bits auf 5 Bits erweitert. In der allgemeinen Nomenklatur der Leitungscodes handelt es sich somit um eine 4B/5B-Codierung. Sie wurde von IBM eingeführt und gehört zu der größeren Familie der GCR-Codierungen. Diese wurden in den früheren Tagen der Computertechnik, genau wie die oben besprochenen FM- und MFM-

IBM-GCR	6520-GCR	Commodore-GCR
0000 $\mapsto$ 11001	0000 $\mapsto$ 11001	0000 $\mapsto$ 01010
0001 $\mapsto$ 11011	0001 $\mapsto$ 11011	0001 $\mapsto$ 01011
0010 $\mapsto$ 10010	0010 $\mapsto$ 10010	0010 $\mapsto$ 10010
0011 $\mapsto$ 10011	0011 $\mapsto$ 10011	0011 $\mapsto$ 10011
0100 $\mapsto$ 11101	0100 $\mapsto$ 11101	0100 $\mapsto$ 01110
0101 $\mapsto$ 10101	0101 $\mapsto$ 10101	0101 $\mapsto$ 01111
0110 $\mapsto$ 10110	0110 $\mapsto$ 10110	0110 $\mapsto$ 10110
0111 $\mapsto$ 10111	0111 $\mapsto$ 10111	0111 $\mapsto$ 10111
1000 $\mapsto$ 11010	1000 $\mapsto$ 11010	1000 $\mapsto$ 01001
1001 $\mapsto$ 01001	1001 $\mapsto$ 01001	1001 $\mapsto$ 11001
1010 $\mapsto$ 01010	1010 $\mapsto$ 01010	1010 $\mapsto$ 11010
1011 $\mapsto$ 01011	1011 $\mapsto$ 01011	1011 $\mapsto$ 11011
1100 $\mapsto$ 11110	1100 $\mapsto$ 11110	1100 $\mapsto$ 01101
1101 $\mapsto$ 01101	1101 $\mapsto$ 01101	1101 $\mapsto$ 11101
1110 $\mapsto$ 01110	1110 $\mapsto$ 01110	1110 $\mapsto$ 11110
1111 $\mapsto$ 01111	1111 $\mapsto$ 01111	1111 $\mapsto$ 10101

Abb. 8.17: Drei Varianten des *Group Coded Recordings*, kurz GCR (Bildquelle: [11])



**Abb. 8.18:** Die Bit-Stuffing-Technik löst das Problem der Resynchronisation, indem gleichförmige Bitketten künstlich unterbrochen werden. Auf der Empfängerseite werden die zusätzlich eingefügten Füllbits wieder aus dem Datenstrom entfernt.

Verfahren, für die Datenspeicherung auf Magnetbändern und Disketten verwendet. Die meisten GCR-Codierungen sind für die Codierung von Byteströmen ausgelegt und bilden die 8 Bits eines Bytes entweder direkt auf ein RLL-Codewort ab oder verarbeiten diese, wie unsere Beispieldcodierung, in Form von Viererblöcken (*Nibbles*).

GRC-Codierungen gibt es viele, und in der mittleren und der rechten Spalte in Abbildung 8.17 sind zwei weitere Beispiele zu sehen. Die 6520-GCR-Codierung wurde für die Datenspeicherung auf Magnetbändern und die Commodore-Codierung für die Datenspeicherung auf Disketten verwendet. Auch das Floppy-Laufwerk des in den Anfängen der Heimcomputerära weit verbreiteten Commodore 64 benutzte zur Speicherung der Daten die Codierung aus der rechten Spalte. Eine praktische Relevanz haben die gezeigten Beispiele heute nicht mehr, und so ist das Interesse an ihnen vor allem historischer Natur.

## 8.2.3 Externe Resynchronisation

In den vorangegangenen Abschnitt haben wir etliche Leitungscodierungen kennengelernt, die selbsttaktend sind, aber auch solche, die diese Eigenschaft nicht erfüllen. Kann ein Code, der keine Taktrückgewinnung erlaubt, überhaupt sinnvoll eingesetzt werden? Die Antwort ist Ja, denn die Selbsttaktung ist nur eine von mehreren Möglichkeiten, um das Problem der Resynchronisation zu lösen. Eine Alternative besteht darin, die Taktung bei der Codierung zunächst zu ignorieren und den generierten Signalverlauf in einer nachgelagerten Verarbeitungsstufe um künstliche Flankenwechsel anzureichern. Zwei Techniken, die für diesen Zweck geeignet sind, wollen wir uns genauer ansehen.

### 8.2.3.1 Bit Stuffing

Die Idee des *Bit Stuffing*, das in manchen deutschen Texten etwas ungelenk als *Bitstopfen* bezeichnet wird, ist bestechend einfach: Sobald ein leitungscodiertes Signal  $n$  gleiche Bits aufweist, wird zur Resynchronisation ein komplementäres Bit – ein sogenanntes *Fillbit* oder *Stopfbit* – eingefügt und dieses auf der Empfängerseite ganz einfach wieder entfernt. Hierzu muss der Empfänger lediglich den eingehenden Datenstrom überwachen und jedes Mal, wenn  $n$  gleiche Bits empfangen wurden, das nachfolgende Bit löschen (Abbildung 8.18). Bei vielen Schnittstellen-Controllern ist die Bit-Stuffing-Logik direkt in Hardware implementiert, sodass bei der Programmierung nicht darauf geachtet werden muss.

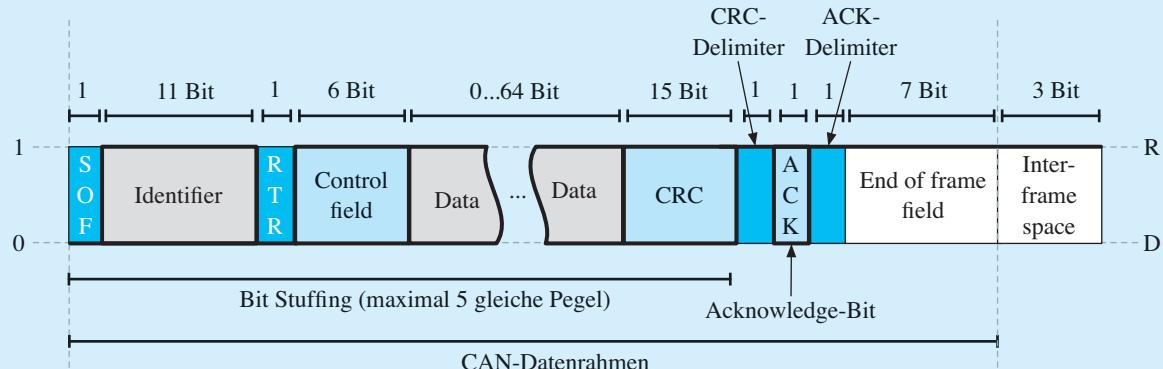


Eingesetzt wird die Bit-Stuffing-Technik unter anderem auf dem CAN-Bus, der in nahezu allen heute gebauten Kraftfahrzeugen für die Vernetzung der Steuergeräte sorgt. Möchte ein Busteilnehmer Informationen versenden, so schreibt er einen sogenannten *Datenrahmen* auf den Bus, dessen logischer Aufbau weiter unten skizziert ist.

Eingeleitet wird ein Datenrahmen durch das *Start-of-Frame-Bit* (SOF), das einer logischen 0 entspricht. In der CAN-Bus-Nomenklatur wird die 0 als dominanter Pegel (D) und die 1 als rezessiver Pegel (R) bezeichnet. Danach folgen, im Standardformat, 11 Bits, die gleichzeitig die ID und die Priorität des Rahmens beschreiben. Das sich anschließende RTR-Bit (*Remote Transmission Request Bit*) entscheidet, ob es sich um einen Datenrahmen (RTR = 0) oder einen Anfragerahmen handelt (RTR = 1). Danach folgen ein Kontrollfeld, in dem unter anderem die Rahmenlänge festgelegt wird, sowie die eigentlichen Nutzdaten. Für die Absicherung der Nutzdaten kommt die zyklische Redundanzprüfung zum Einsatz, mit der wir aus Abschnitt 6.4.3 gut vertraut sind. Hinter den CRC-Prüfbits folgt ein spezielles *Acknowledge-Bit*, das von zwei rezessiven *Delimiter-Bits* eingegrenzt wird und ei-

ne primitive Empfangskontrolle implementiert. Abgeschlossen wird ein Datenrahmen durch 7 rezessive Bits, die keine Information in sich tragen. Danach muss der Bus für mindestens drei weitere Takte frei sein, bis ein neuer Datenrahmen gesendet werden darf.

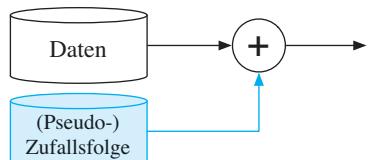
Betrachten wir den logischen Aufbau eines Datenrahmens genauer, so fällt auf, dass die Bit-Stuffing-Technik nur in bestimmten Teilabschnitten verwendet wird. Ausgenommen sind z. B. das *End-of-Frame-Field* sowie alle Bereiche, die zu keinem Rahmen gehören und den Bus-Idle-Zustand repräsentieren. Ferner sieht das CAN-Bus-Protokoll sogenannte *Fehlerrahmen* als weiteren Telegrammtyp vor, die ebenfalls von der Bit-Stuffing-Regel ausgenommen sind. Der Grund hierfür ist einfach: Würde auf die Ausnahmen verzichtet werden, so könnte ein Steuergerät weder entscheiden, ob der Bus frei ist und somit beschrieben werden darf, noch könnte er das Ende eines Datenrahmens detektieren. Auch die Fehlerrahmen wären nicht mehr länger als solche erkennbar. Die Bit-Stuffing-Technik wird bei der Kommunikation von Steuergeräten somit nicht nur zur Resynchronisation eingesetzt. Sie wird implizit dazu verwendet, um wichtige logische Informationen auf dem Bus zu übermitteln.



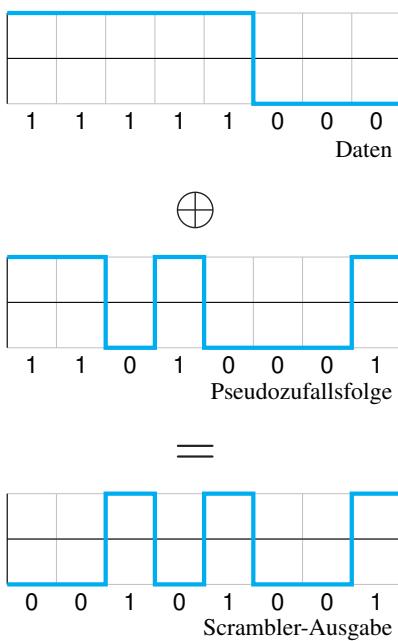
Die Bit-Stuffing-Technik ist eine effektive, aber nicht in jeder Hinsicht günstige Lösung. Zum einen führen die künstlich eingefügten Füllbits zu einer Verringerung der Coderate. Zum anderen lässt sich die Anzahl der übertragenen Füllbits nicht vorhersagen, sodass die Bitrate, mit der die Nettodaten übertragen werden, über die Zeit schwanken kann. Unter Umständen wird es hierdurch schwieriger, die empfangenen Daten innerhalb der Empfängerhardware synchron weiterzuverarbeiten.

### Scrambler

■ Aufbau



■ Signalverlauf



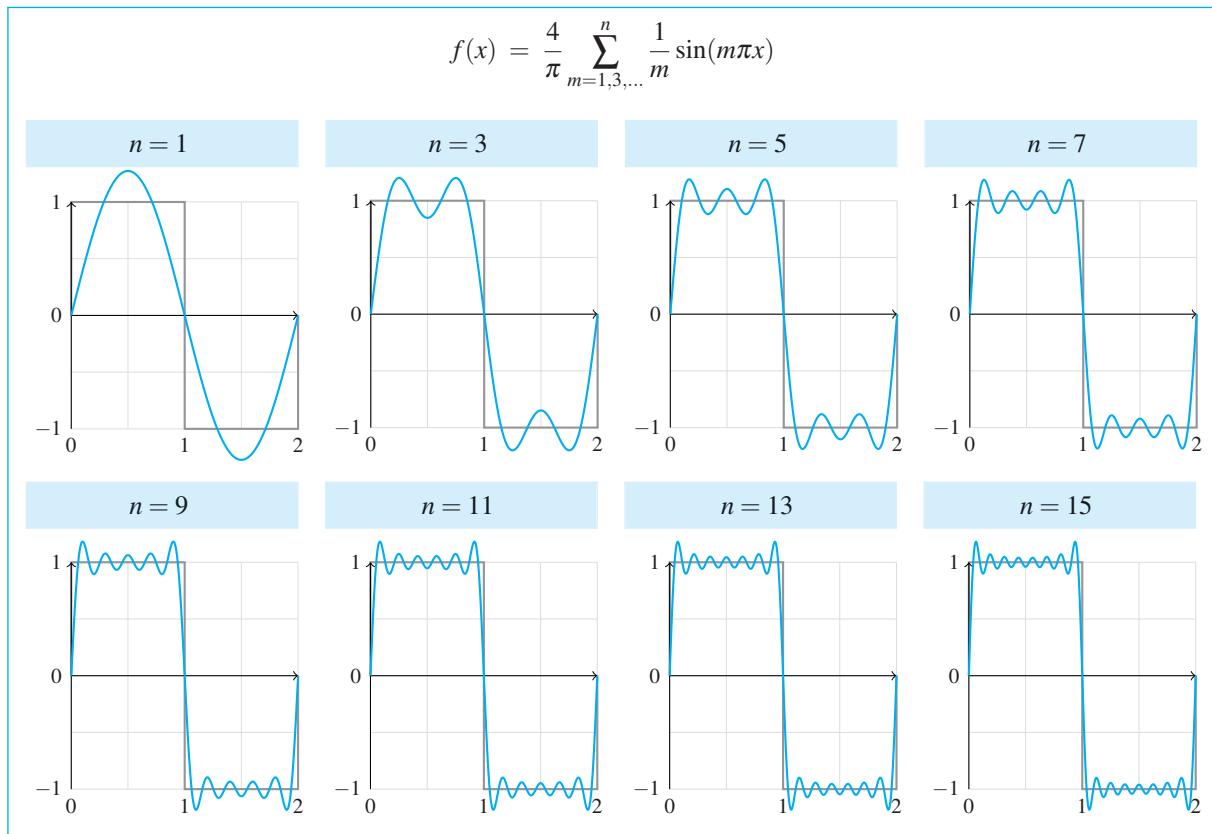
**Abb. 8.19:** Ein Scrambler verknüpft den eingespeisten Bitstrom mit einer zufällig oder pseudozufällig generierten Folge. Der entstehende Signalverlauf ist in den allermeisten Fällen genauso ungeordnet wie die Zufallsfolge selbst.

#### 8.2.3.2 Scrambler

Scrambler lösen das Problem der Resynchronisation, indem sie den Bitstrom mit einer zufällig generierten Bitfolge verknüpfen, die sowohl dem Empfänger als auch dem Sender bekannt ist (Abbildung 8.19). Als Ergebnis erhalten wir in den allermeisten Fällen eine Bitfolge, die genauso ungeordnet ist wie die Zufallsfolge selbst, und somit genug Flankenwechsel enthält, um die Rekonstruktion des Taktsignals zu gewährleisten. Dass wir hier nur von den „allermeisten Fällen“ und nicht von allen Fällen sprechen, hat seinen Grund. Anders als bei der Übertragung mithilfe der Bit-Stuffing-Technik, die in allen Fällen funktioniert, gibt es für jeden Scrambler ganz bestimmte Eingabesequenzen, die einen gleichförmigen Signalpegel erzeugen. Dies sind genau jene Sequenzen, die die gleiche Struktur aufweisen wie das Scrambler-Signal und die künstlich eingebrachte Unordnung beseitigen. Die Wahrscheinlichkeit, dass eine zu übertragende Bitsequenz das Scrambler-Signal auslöst, ist jedoch so klein, dass dieses Problem in den meisten Anwendungsfällen gefahrlos ignoriert werden darf.

Häufig dienen Scrambler noch einem anderen Zweck. Wird die Folge, mit der die Nutzdaten verknüpft werden, geheim gehalten, so führt dies zu einer impliziten Verschlüsselung der Sendedaten. Wird dabei, wie eben geschildert, eine echte Zufallsfolge benutzt, so entspricht dies einer Verschlüsselung mit einem *One-Time-Pad* [29]. Eine solche Folge ist optimal geschützt und kann ohne die Kenntnis des One-Time-Pads nicht entschlüsselt werden.

Eines haben wir dabei allerdings noch nicht bedacht: Wenn wir für die Verknüpfung eine echte Zufallsfolge verwenden wollen, muss diese vorab an den Empfänger übertragen werden, aber genau dies ist, ohne die Folge bereits übermittelt zu haben, gar nicht möglich. Die meisten Scrambler, die in der Praxis verwendet werden, arbeiten daher nicht mit echten, sondern mit sogenannten *Pseudozufallsfolgen*. Diese weisen immer noch eine hinreichend große Unordnung zwischen den Nullen und Einsen auf, lassen sich aber vergleichsweise einfach mit rückgekoppelten Schieberegistern erzeugen. Wie solche Schaltungen konkret aussehen, werden wir weiter unten, in Abschnitt 8.5.2, besprechen. Dort kommen wir im Zusammenhang mit der CDMA-Technik im Detail auf die Erzeugung von Pseudozufallsfolgen zurück.



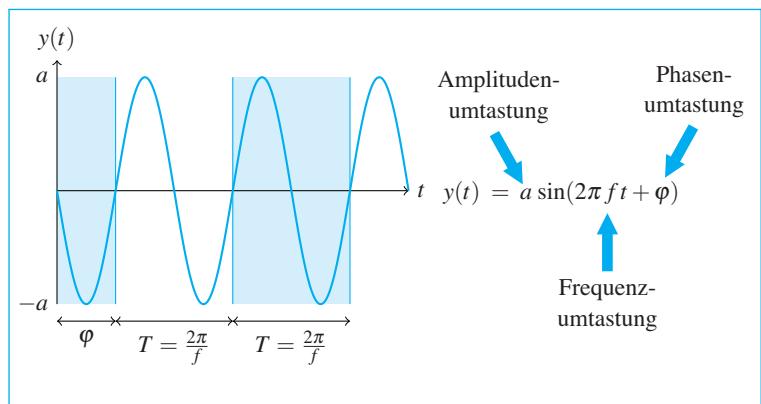
**Abb. 8.20:** Fourierreihenentwicklung am Beispiel der NRZ-codierten Bitsequenz 10

## 8.3 Modulationsverfahren

Mit der Leitungscodierung haben wir einen ersten Schritt in Richtung der physikalischen Datenübertragung unternommen. Wir haben wichtige Parameter wie die Gleichstromfreiheit in die Signalverläufe eingearbeitet und sind damit gut vorbereitet, um den finalen Schritt zu vollziehen. Dieser besteht darin, das immer noch digitale Signal so in ein analoges zu wandeln, dass es über ein physikalisches Medium transportiert werden kann.

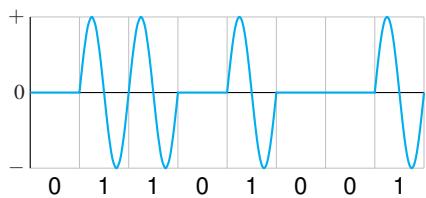
Einen ersten Lösungsansatz finden wir im Bereich der Fourier-Analysis, einem genauso fundamentalen wie faszinierenden Zweig der klassischen Mathematik. Es ist eines ihrer elementaren Ergebnisse, dass sich jede periodische Funktion  $f(x)$  in Form einer trigonometrischen Reihe

**Abb. 8.21:** Das Trägersignal ist ein periodisch schwingendes Signal, dessen Amplitude, Frequenz und Phasenversatz über die Zeit konstant bleiben. Die digitale Information wird auf das Trägersignal moduliert, indem ein oder mehrere Signalparameter für kurze Zeit verändert werden.

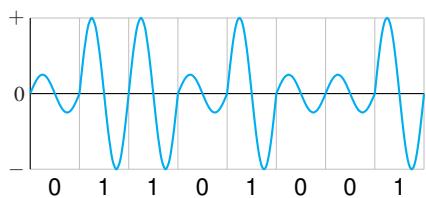


#### Amplitudenumtastung (ASK)

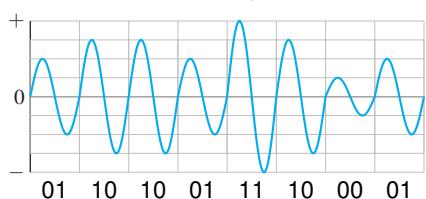
##### ■ On-Off Keying (OOK)



##### ■ Signalabsenkung um 25 %



##### ■ Quaternäre Codierung



**Abb. 8.22:** Amplitudenumtastung (ASK)

entwickeln lässt. Hieraus folgt, dass ein digitaler Signalverlauf durch die Überlagerung verschiedener Sinus- und Cosinusschwingungen beliebig approximiert werden kann, und sich damit tatsächlich analog übertragen lässt. Wie eine solche Approximation aussieht, zeigt Abbildung 8.20 am Beispiel der NRZ-codierten Bitsequenz 10. Die Funktion, die sich in diesem Fall ausschließlich aus Sinusschwingungen zusammensetzt, nähert sich dem digitalen Signalverlauf immer besser an, je mehr hochfrequente Signale in die Summe miteinbezogen werden.

In der Praxis wird diese Art der Modulation nicht verwendet, und dies hat triftige Gründe. Zum einen ist jedes physikalische Medium bandbreitenbegrenzt, sodass wir den digitalen Signalverlauf niemals exakt darstellen können. Zum anderen, und dies wiegt genauso schwer, würden wir für die Kommunikation eines einzigen Signals das volle Frequenzspektrum belegen.

### 8.3.1 Digitale Modulation

In der Praxis werden digitale Datenströme durch die Modulation eines analogen Trägersignals übertragen. Mathematisch können wir das Trägersignal durch eine sinusförmige Wellenfunktion

$$y(t) = a \sin(2\pi f t + \varphi)$$

mit der *Amplitude*  $a$  beschreiben, die mit der *Frequenz*  $f$  schwingt und den *Phasenversatz*  $\varphi$  aufweist. Diese Funktion zu *modulieren* bedeutet, die digitale Information des Nutzdatenstroms durch die Anpassung von einem oder mehreren dieser drei Parameter in das Trägersignal hineinzucodieren. Die folgenden drei Basisvarianten lassen sich unterscheiden:

## ■ Amplitudenumtastung

### Amplitude Shift Keying (ASK)

Bei der Amplitudenumtastung wird die digitale Information durch die Veränderung der Nennleistung auf das Trägersignal moduliert. Die einfachste Variante ist das *On-Off Keying*, kurz OOK, bei dem das Trägersignal für die Übertragung einer Null abgeschaltet wird (Abbildung 8.22 oben). Die zweite in Abbildung 8.22 gezeigte Variante arbeitet geringfügig anders und senkt die Nennleistung für die Übertragung einer Null lediglich auf 25 % ab. Gestattet das Übertragungsmedium eine saubere Trennung der Signalpegel, so können durch die Verwendung verschiedener Amplitudenstufen mehrere Bits auf einmal übertragen werden. Das untere Beispiel in Abbildung 8.22 demonstriert dies anhand eines Signals, das vier verschiedene Amplituden annehmen und damit in jedem Taktschritt zwei Bits auf einmal übertragen kann.

## ■ Frequenzumtastung

### Frequency Shift Keying (FSK)

Die Frequenzumtastung modifiziert die Frequenz des Trägersignals, um die Binärwerte 0 und 1 zu unterscheiden. Abbildung 8.23 demonstriert dieses Verfahren anhand zweier Beispiele. Das erste zeigt die Ausgabe eines binären Frequenzumtasters, der zwischen zwei unterschiedlichen Frequenzen auswählt. In der gezeichneten Signalkurve wird eine 0 codiert, indem die Frequenz des Trägersignals um die Hälfte reduziert wird.

Das zweite Beispiel in Abbildung 8.23 zeigt einen quaternären Frequenzumtaster, der zwischen vier verschiedenen Frequenzen auswählt und damit jeweils zwei Bits auf einmal codiert. Das bedeutet, dass die Frequenzumtastung die Eigenschaft der Amplitudenumtastung teilt, mehrere Bits gleichzeitig codieren zu können.

## ■ Phasenumtastung

### Phase Shift Keying (PSK)

Bei der binären Phasenumtastung, kurz BPSK oder 2-PSK, werden die Bits 0 und 1 mit zwei Signalen codiert, die in ihrer Amplitude und ihrer Frequenz mit dem Trägersignal übereinstimmen, gegeneinander aber einen Phasenversatz aufweisen. Um die Signale optimal voneinander zu trennen, wird meist ein Versatz von  $180^\circ$  gewählt, aber auch hier ist es möglich, durch die Erhöhung der möglichen Phasenlagen mehrere Bits auf einmal zu codieren.

Abbildung 8.24 demonstriert die Phasenumtastung anhand zweier Beispiele. Das erste ist eine Variante, bei der die binäre 0 mit einer Phasenlage von  $0^\circ$  und die binäre 1 mit einer Phasenlage von  $180^\circ$  codiert wird. Das zweite Beispiel benutzt vier Phasenlagen und codiert damit jeweils 2 Bits auf einmal.

## Frequenzumtastung (FSK)

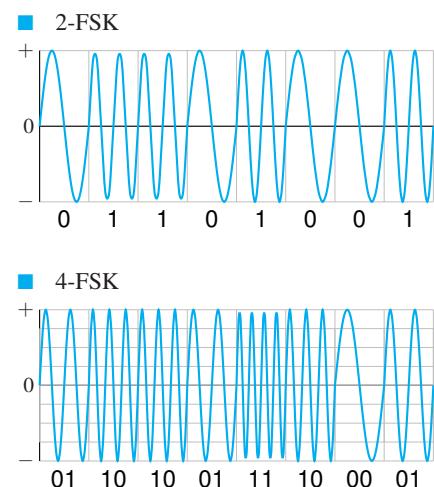


Abb. 8.23: Frequenzumtastung (FSK)

## Phasenumtastung (PSK)

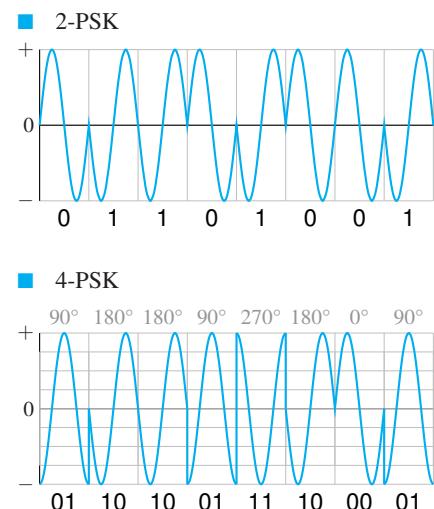
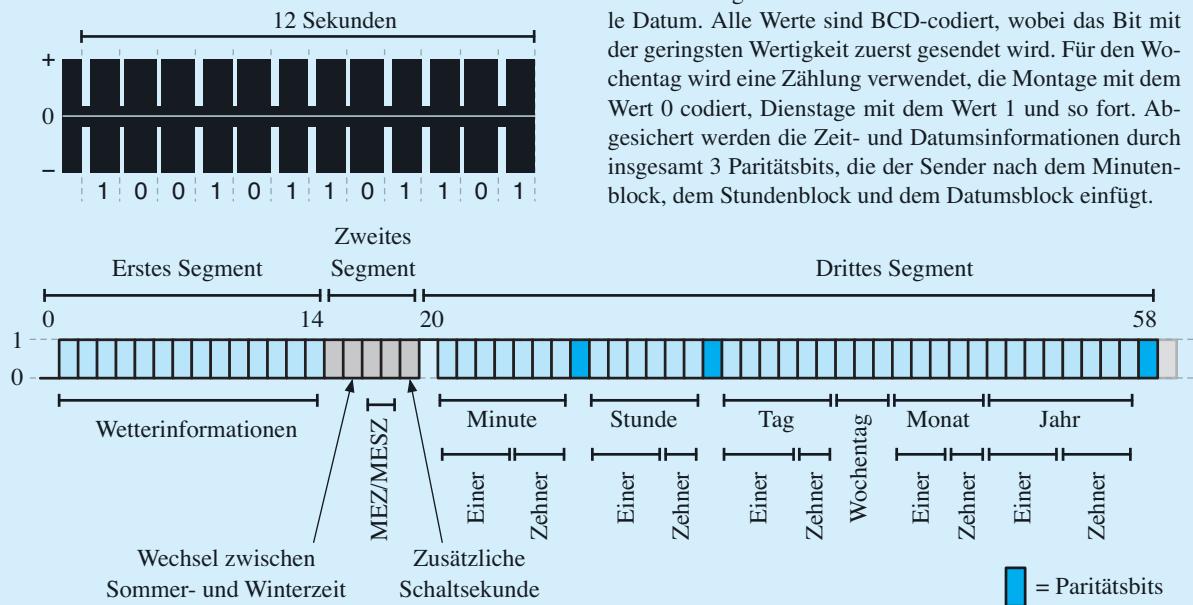


Abb. 8.24: Phasenumtastung (PSK)



Ein bekanntes Beispiel eines amplitudenumgetasteten Signals ist das langwellige DCF77-Signal, das mit einer Frequenz von 77,5 kHz aus dem Hessischen Mainflingen gesendet wird. In ganz Mitteleuropa sorgt dieses Signal für die Synchronisierung von Funkuhren. Der Sender codiert die digitale Information, indem er die Amplitude zu Beginn einer Sekunde auf ca. 15 % der Nennleistung absenkt und den reduzierten Pegel für eine gewisse Zeitspanne konstant hält. Eine Zeitspanne von 100 msec bedeutet eine logische 0 und eine Zeitspanne von 200 msec eine logische 1.



Zum Zwecke der Synchronisation wird die Absenkung jeweils in der letzten Sekunde einer Minute unterdrückt. Damit überträgt DCF77 pro Minute genau 59 Nutzdatenbits. Das unten abgebildete Diagramm zeigt, wie der Bitstrom decodiert werden kann. Insgesamt teilt er sich in drei Segmente auf, wobei das erste mit einer 0 und das zweite mit einer 1 eingeleitet wird. Der Inhalt des ersten Segments hat sich in der Vergangenheit mehrfach verändert. Zurzeit werden darin Wetterinformationen codiert.

Das zweite Segment enthält mehrere Kontrollbits, die Meatinformation zur Zeitsteuerung enthalten. Beispielsweise wird ein Funkempfänger mit den Bits 16 und 19 darüber informiert, ob ein Wechsel zwischen Sommer- und Winterzeit ansteht oder eine zusätzliche Schaltsekunde eingefügt werden muss.

Das dritte Segment enthält die aktuelle Zeit und das aktuelle Datum. Alle Werte sind BCD-codiert, wobei das Bit mit der geringsten Wertigkeit zuerst gesendet wird. Für den Wochentag wird eine Zählung verwendet, die Montage mit dem Wert 0 codiert, Dienstage mit dem Wert 1 und so fort. Abgesichert werden die Zeit- und Datumsinformationen durch insgesamt 3 Paritätsbits, die der Sender nach dem Minutenblock, dem Stundenblock und dem Datumsblock einfügt.

Angewendet wird die Phasenumtastung beispielsweise bei der Übertragung einer Faxnachricht. Es sind die Phasensprünge, die beim Senden oder Empfangen einer Nachricht als Knackgeräusche mithören können.

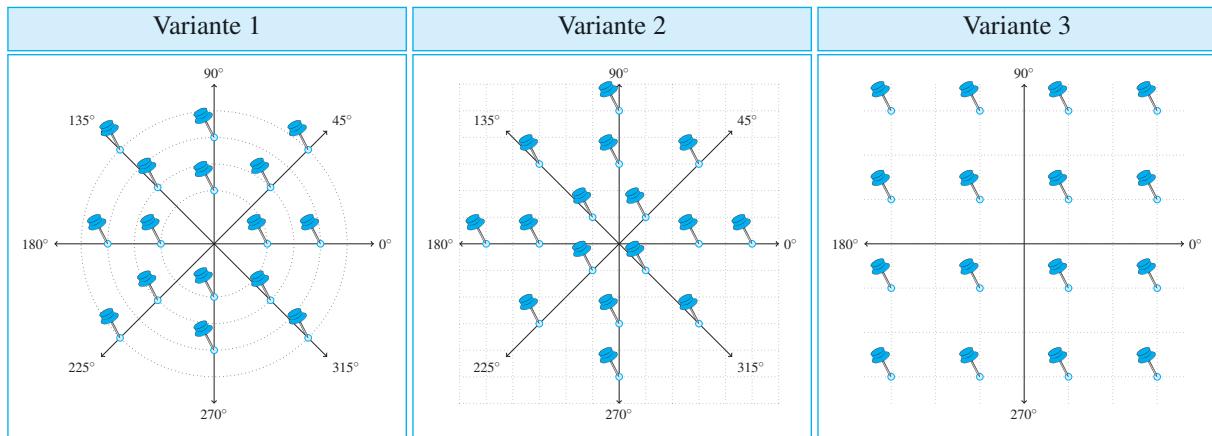


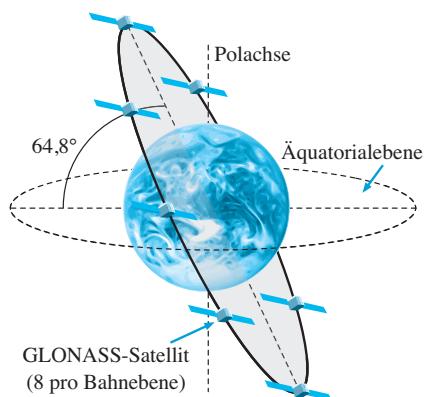
Abb. 8.25: Drei mögliche Signalraumkonstellationen der 16-stufigen Quadraturamplitudenmodulation (16-QAM)

### 8.3.2 Kombinierte Modulationsverfahren

In der Praxis werden die drei vorgestellten Grundmodulationen oft miteinander kombiniert. Wie dies geschehen kann, wollen wir uns am Beispiel der *Quadraturamplitudenmodulation*, kurz QAM, genauer ansehen. Hierbei handelt es sich um die Kombination aus der Amplitudenumtastung und der Phasenumtastung; es wird also gleichzeitig die Amplitude und die Phase des Trägersignals angepasst.

Die Zuordnung, die zwischen den modulierten Signalparametern und den codierten Bits besteht, wird gerne in Form von zweidimensionalen *Signalraumdiagrammen* angegeben, wie sie exemplarisch in Abbildung 8.25 zu sehen sind. Die Diagramme verwenden ein polares Koordinatensystem, in dem die Winkelkoordinaten die Phasenlagen und die Radialkoordinaten die Amplituden eines modulierten Trägersignals beschreiben. Jeder eingezeichnete Punkt  $(\varphi, r)$  steht für eine Konstellation aus einem Phasenwinkel  $\varphi$  und einer Amplitude  $r$ . In den drei dargestellten Diagrammen sind jeweils 16 Punkte eingezeichnet, d. h., es kann in jedem Takt eine von 16 Konstellationen auf dem Übertragungsmedium beobachtet werden. Dies ermöglicht die gleichzeitige Übertragung von 4 Bits.

Kann das eingesetzte Übertragungsmedium die Signalparameter hinreichend scharf trennen, so lässt sich die Anzahl der gleichzeitig übertragenen Bits erheblich steigern. Ein Beispiel ist der DVB-C-Standard (*Digital Video Broadcasting – Cable*), der in deutschen Kabelnetzen zum Einsatz kommt und eine Übertragung mittels 256-QAM vorsieht.



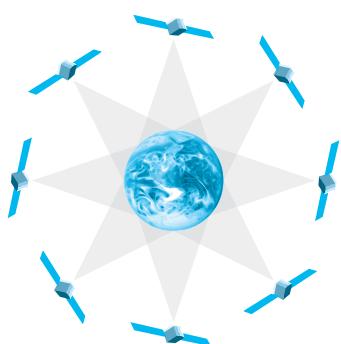
**Abb. 8.26:** Die 24 GLONASS-Satelliten umkreisen auf 3 Bahnebenen die Erde, die zur Äquatorialebene einen Winkel von  $64,8^\circ$  aufweisen. Um die Grafik übersichtlich zu halten, ist nur eine Bahnebene eingezeichnet. Die anderen entstehen, indem die eingezeichnete Ebene in  $120^\circ$ -Schritten um die Polachse gedreht wird.

Pro Takt werden dort 8 Bits gleichzeitig übertragen. Im Nachfolgestandard DVB-C2 ist die Verwendung von 4096-QAM vorgesehen, was die gleichzeitige Übertragung von 12 Bits ermöglicht.

## 8.4 Multiplexverfahren

In diesem Abschnitt werden wir uns mit einem Problem auseinandersetzen, das in der Praxis allgegenwärtig ist: die Mehrfachnutzung eines Übertragungsmediums. Mehrere Sender und Empfänger über das gleiche Medium kommunizieren zu lassen, war bereits zu Zeiten der Morse-Telegrafie ein vehement verfolgtes Ziel. Während damals aber hauptsächlich kommerzielle Interessen im Vordergrund standen, ist die Mehrfachnutzung von Übertragungsmedien heute durch die flächendeckende Verbreitung mobiler Endgeräte eine Notwendigkeit. Anders als bei einer kabelgebundenen Vernetzung haben wir bei der drahtlosen Kommunikation keinerlei Möglichkeit, das Übertragungsmedium mehrfach auszulegen: Alle Sender teilen sich den sprichwörtlichen Äther, und ohne den Einsatz ausgefeilter Multiplexverfahren würde die Kommunikation schnell im Chaos versinken.

In den folgenden Abschnitten werden wir nacheinander die vier klassischen Medienzugriffsverfahren vorstellen, die eine koordinierte Mehrfachnutzung eines Übertragungsmediums gewährleisten.



**Abb. 8.27:** Auf jeder Bahnebene wird durch die gleichmäßige Anordnung der GLONASS-Satelliten eine antipodale Anordnung erreicht: Zu jedem Satelliten existiert ein Antipode, der auf die gegenüberliegende Erdsseite ausstrahlt.

### 8.4.1 Frequenzmultiplexverfahren

In Abschnitt 8.3 haben wir herausgearbeitet, dass für die digitale Datenübertragung ein analoges Trägersignal verwendet wird, dessen Frequenz weitgehend frei gewählt werden kann. Benutzt jeder Sender eine andere Frequenz, so können die Trägersignale verlustfrei überlagert und auf der Empfängerseite mithilfe eines Bandfilters sauber voneinander getrennt werden. Genau dies ist die Idee des Frequenzmultiplexverfahrens.

Ein Anwendungsbeispiel für die FDMA-Technik (*Frequency Division Multiple Access*) ist das satellitengestützte Navigationssystem GLONASS, das vom Verteidigungsministerium der Russischen Föderation betrieben wird und nicht nur technologisch, sondern auch in Bezug auf die bereitgestellten Dienste dem US-amerikanischen GPS sehr ähnlich ist.



Am Beispiel des GLONASS-Satellitensystems ist deutlich geworden, dass die ungünstige Wahl eines Frequenzbands gewaltige Auswirkungen haben kann. Die Leittragende war in diesem Fall die Radioastronomie, die durch die Frequenzen im Bereich von 1610,6 bis 1613,8 MHz erhebliche gestört wurde.

Die Koexistenz verschiedener Technologien ist heute nur durch eine akribische Kontrolle der Frequenzbänder möglich. Dies gilt insbesondere für den Bereich des Mobilfunks, wo nicht nur unterschiedliche Übertragungsstandards, sondern auch konkurrierende Netzbetreiber koordiniert werden müssen. Die Frequenzbänder, die in Deutschland für den Mobilfunk benutzt werden, vergibt die *Regulierungsbehörde für Telekommunikation und Post*, kurz RegTP. Beispielsweise ist festgelegt, dass die Bereiche von 890 bis 915 MHz, 1710 bis 1785 MHz und 1805 bis 1880 MHz für den GSM-Mobilfunk benutzt werden. UMTS-Geräte kommunizieren auf Frequenzen zwischen 1920 und 1980 MHz bzw. zwischen 2110 und 2170 MHz [62]. Der LTE-Standard benutzt zwei Frequenzbereiche, die weit auseinander liegen. Für die Versorgung ländlicher Regionen werden Frequenzen im Bereich von 800 MHz verwendet, und in dicht besiedelten Gegenden wird auf Frequenzen im Bereich von 2600 MHz zurückgegriffen.

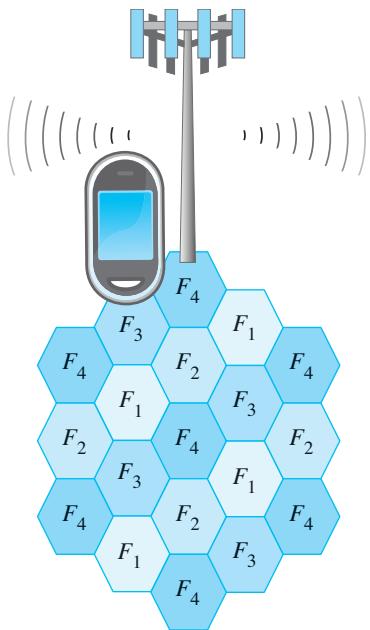
Im Vollausbau umkreisen 24 GLONASS-Satelliten die Erde, von denen für eine erfolgreiche Positionsbestimmung mindestens vier in der Reichweite des Empfängers sein müssen. Die Satelliten befinden sich auf kreisförmigen Umlaufbahnen in einer Höhe von 19 100 km und benötigen für einen vollen Umlauf ca. elf Stunden. Die Orbitalbahnen sind so ausgerichtet, dass jeweils acht Satelliten eine Bahnebene bilden (Abbildung 8.26). Auf jeder der drei Ebenen sind die Satelliten in gleichmäßigen Abständen angeordnet, d. h., es existiert zu jedem Satelliten ein Antipode, der ihm exakt gegenüber steht (Abbildung 8.27). Die Bahnebenen sind dabei so ausgerichtet, dass die GLONASS-Satelliten, von den Polen aus betrachtet, höher am Horizont stehen als die GPS-Satelliten. Dies verbessert die Ortungsgenauigkeit in sehr nördlichen und sehr südlichen Regionen.

Jeder GLONASS-Satellit übermittelt zwei Positionssignale: das SP-Signal (*Standard Precision Signal*), das für die zivile Nutzung gedacht ist, und das HP-Signal (*High Precision Signal*), das für militärische Zwecke eingesetzt wird. Das SP-Signal wird im L1-Band und das HP-Signal im L2-Band übertragen. In jedem Band stehen mehrere Kanäle zur Verfügung, denen jeweils eine unterschiedliche Frequenz zugeordnet ist. Ist  $k$  die Kanalnummer, so lassen sich die Frequenzen mithilfe der folgenden Formeln berechnen:

$$\begin{aligned}L_1 : f_{L1}(k) &= 1602 \text{ MHz} + k \cdot 562,5 \text{ kHz} \\L_2 : f_{L2}(k) &= 1246 \text{ MHz} + k \cdot 437,5 \text{ kHz}\end{aligned}$$

Bis zum Jahr 2005 wurden für die Übertragung die Kanäle  $k = 0$  bis  $k = 24$  verwendet. Mit diesen Kanalnummern erstreckt sich beispielsweise das L1-Band von 1602 MHz bis 1615,5 MHz. Im Zuge des GLONASS-Ausbaus stellte sich heraus, dass die Frequenzen im Band 1610,6 bis 1613,8 MHz mit den Radioteleskopen interferieren, die für die astronomische Forschung verwendet werden. Um dieses Problem zu umgehen, verwenden die nach 2005 gestarteten GLONASS-Satelliten die Kanalnummern –7 bis +6. Dabei sind die Kanäle 5 und 6 dem Testbetrieb vorbehalten; die restlichen 12 werden für den regulären Betrieb genutzt.

Wir rekapitulieren: Im Vollausbau umkreisen 24 GLONASS-Satelliten die Erde, es stehen seit der Neuaustrichtung der Frequenzbänder aber nur noch 12 Frequenzkanäle für den regulären Betrieb zur Verfügung. Wie kann das funktionieren?



**Abb. 8.28:** Wabenförmige Struktur eines Mobilfunknetzes. Die Platzhalter  $F_1, \dots, F_4$  stehen für verschiedene, sich nicht überlappende Frequenzbänder. Die Zuordnung ist so gewählt, dass zwei benachbarte Funkzellen niemals das gleiche Band nutzen.

## 8.4.2 Raummultiplexverfahren

Dass sich die GLONASS-Betreiber auf eine Reduzierung der Kanäle einlassen konnten, geht auf die antipodale Anordnung der Satelliten zurück. Sie sorgt dafür, dass sich jeweils zwei Satelliten exakt gegenüberstehen und somit auf jeweils disjunkte Areale abstrahlen. Da sich diese Satelliten nicht gegenseitig stören können, verwenden sie heute den gleichen Frequenzkanal, und genau das ist der Grund, warum 12 Kanäle für einen störungsfreien Betrieb von 24 Satelliten ausreichen.

Natürlich ist dieses Prinzip nicht nur im Bereich der Satellitentechnik nutzbar. Immer dann, wenn mehrere, räumlich verteilte Übertragungswege existieren, kann ein Frequenzbereich mehrfach benutzt werden, ohne dass sich die gesendeten Signale gegenseitig stören. Verfahren, die sich dieses Prinzips bedienen, werden als Raummultiplexverfahren bezeichnet (*Space Division Multiple Access*, kurz SDMA).

Ein klassischer Einsatzbereich für SDMA ist der terrestrische Mobilfunk. Dort ist die räumliche Trennung technologieinhärent, da die geringen Sendeleistungen der Endgeräte ein feinmaschiges Sendernetz erfordern. Im GSM-, UMTS- oder LTE-Netz ist das Sendegebiet in *Mobilfunkzellen* aufgeteilt, die zumeist wabenförmig angeordnet sind (Abbildung 8.28). Die Größen der Waben können dabei erheblich variieren. Ländliche Regionen werden in *Großzellen* versorgt, die einen Durchmesser zwischen 5 und 20 km aufweisen. Städtische Bereiche sind in *Kleinzellen* aufgeteilt, deren Durchmesser 0,5 bis 5 km beträgt. In Flughäfen, Bahnhöfen und Einkaufszentren kommen häufig *Mikrozellen* zum Einsatz, die Areale mit einem Durchmesser von weniger als 150 m abdecken.

Das GSM-Netz hat im Vergleich zu den moderneren Netzen den Nachteil, dass die Größe einer Zelle, unabhängig von ihrer momentanen Auslastung, immer gleich bleibt und damit ein statischer Grenzverlauf entsteht. Das bedeutet, dass die Überlast in einer Funkzelle nicht durch benachbarte, weniger belastete Zellen ausgeglichen werden kann. Im UMTS-Netz ist dies anders. Dort können die Zellen ihren Radius in gewissen Grenzen anpassen und auf diese Weise für einen Lastenausgleich sorgen. Droht eine Überlast, so wird der Radius der betreffenden Zelle verkleinert und der frei werdende Bereich von den Nachbarzellen übernommen. Dieser Effekt wird als *Zellatmung* (*cell breathing*) bezeichnet.

Probleme treten an den Grenzen zwischen zwei Funkzellen auf. Da sich diese an den Rändern überlappen müssen, um das Sendegebiet vollständig zu überdecken, dürfen zwei benachbarte Zellen nicht die glei-

chen Frequenzbänder verwenden. In Abbildung 8.28 wird dieses Problem gelöst, indem jeder Zelle eines von insgesamt vier verschiedenen Frequenzbändern  $F_1$  bis  $F_4$  zugeordnet wird. Diese sind so verteilt, dass zwei Nachbarzellen niemals das gleiche Band verwenden.

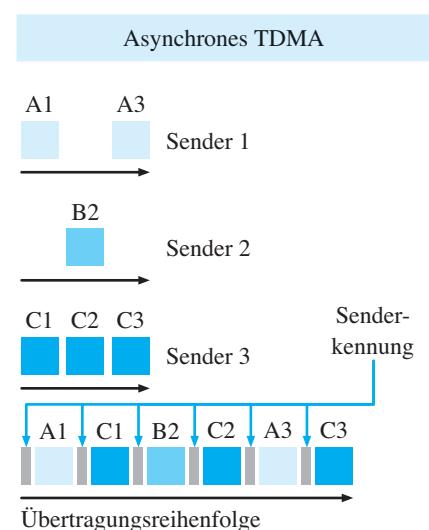
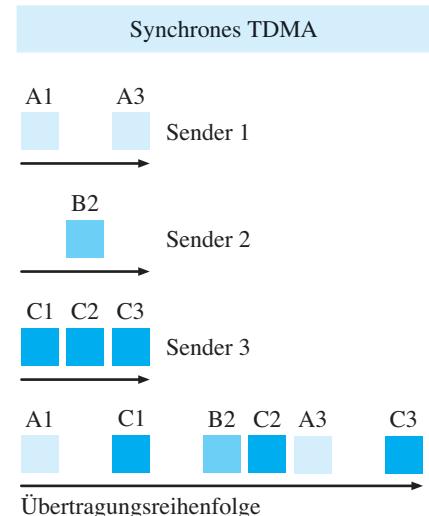
### 8.4.3 Zeitmultiplexverfahren

Erinnern Sie sich an den Telegrafen von Jean-Maurice-Émile Baudot, der im Übungsteil auf Seite 77 vorgestellt wurde? Der französische Ingenieur hatte einen Multiplexer ersonnen, über den sich bis zu sechs Sender an eine einzelne Telegrafenleitung anschließen ließen. Der Multiplexer hatte die Aufgabe, den Ausgang von jeweils einem Sender auf die Telegrafenleitung zu schalten und die anderen Sender in dieser Zeit zu blockieren. Nach dem Rotationsprinzip wurde im ersten Zeitfenster der erste Sender verbunden, im zweiten Zeitfenster der zweite und so fort. In der modernen Terminologie ist Baudots Erfindung eine Apparatur, die nach dem *synchronen Zeitmultiplexverfahren* arbeitet.

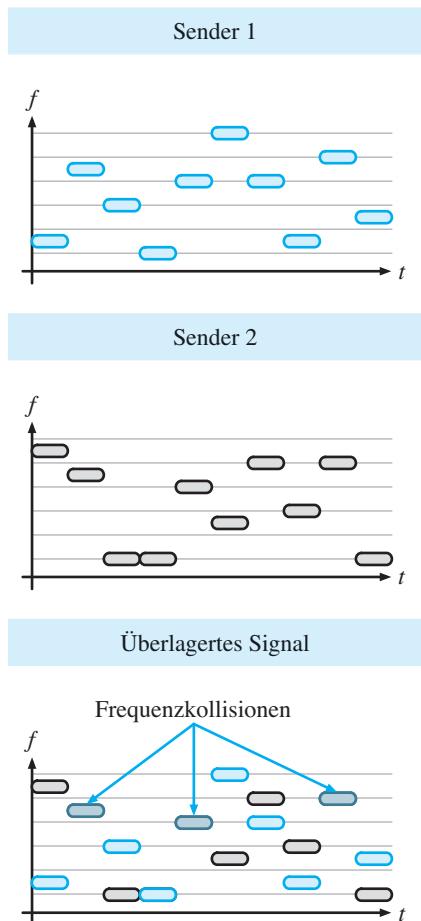
Das Wort *synchron* bezieht sich auf die Eigenschaft des Baudot-Telegrafen, einem Sender selbst dann ein Zeitfenster zuzuweisen, wenn dieser überhaupt keine Daten übertragen möchte (Abbildung 8.29 oben). Obwohl das Medium gar nicht benutzt wird, bleibt es dennoch blockiert; alle übertragungsbereiten Sender müssen auf das ihnen zugeteilte Fenster warten.

Heute werden neben dem synchronen TDMA-Verfahren auch asynchrone Varianten eingesetzt, die auf die feste Fensterzuordnung verzichten und stattdessen bedarfsorientierte Vergabealgorithmen einsetzen (Abbildung 8.29 unten). Auf der positiven Seite sorgen die asynchronen Varianten für eine besser Ausnutzung des Übertragungsmediums. Auf der negativen Seite erhöhen sie den Koordinierungsaufwand, da der Senderzeitpunkt keinerlei Auskunft mehr über den Sender gibt. Eingesetzt wird das TDMA-Verfahren beispielsweise auf dem Flexray-Bus, der ursprünglich für den Einsatz in Kraftfahrzeugen, als Ersatz für den weniger leistungsfähigen CAN-Bus, konzipiert wurde.

Vergleichen wir das Zeitmultiplexverfahren mit dem Frequenzmultiplexverfahren, so tritt ein wesentlicher Unterschied zu Tage: Im Gegensatz zum FDMA-Verfahren, bei dem mehrere Sender tatsächlich gleichzeitig senden, erfolgt die Kommunikation mittels TDMA nur scheinbar parallel. Für eine geregelte Kommunikation benötigt das TDMA-Verfahren daher eine exakte Teilnehmersynchronisation. Bei monolithischen, in sich abgeschlossenen Systemen, zu denen auch der antiquier-



**Abb. 8.29:** Das synchrone und das asynchrone Zeitmultiplexverfahren im Vergleich



**Abb. 8.30:** Frequency Hopping Spread Spectrum (FHSS)

te Baudot-Telegraf gehört, ist eine solche Synchronisation vergleichsweise einfach zu implementieren, bei dezentral organisierten Systemen wird sie dagegen rasch zu einer Herausforderung.

Ein Beispiel für ein hoch dezentralisiertes System ist der Mobilfunk mit seinen unzähligen, unabhängig voneinander betriebenen Endgeräten. Die Frage, die sich dort stellt, ist von prinzipieller Natur. Lässt sich der Erstzugriff zweier Endgeräte, die zufällig zur gleichen Zeit eingeschaltet werden, überhaupt zeitlich koordinieren? Da beide Endgeräte keinerlei Kenntnis voneinander haben, ist die Antwort Nein und entsprechend rigide wird dieses Problem gelöst: Auf die Koordination des Erstzugriffs wird vollständig verzichtet, d. h., jedes Gerät beginnt ganz einfach mit dem Versenden von Daten, sobald diese vorliegen. Erst wenn eine Antwort ausbleibt, gehen die Endgeräte von einer gegenseitigen Störung aus und versuchen, die Daten kurze Zeit später noch einmal zu senden.

Damit sich die Kollisionen nicht dauerhaft wiederholen, werden spezielle Protokolle eingesetzt, die den Wiederholungszeitpunkt um eine pseudozufällige Zeitspanne hinausschieben. Eines der ältesten Protokolle, die mit zufallsgenerierten Verzögerungen arbeiten, ist das *ALOHA-Protokoll*, das 1970 an der Universität von Hawaii für den Einsatz in dem Funknetzwerk *ALOHAnet* entwickelt wurde. ALOHAnet wurde bereits vor Jahren abgeschaltet, da es durch die Entwicklung des Internets seine Existenzberechtigung verlor. Das ALOHA-Protokoll hat die Zeit dagegen überlebt. Nahezu alle modernen Funknetzwerke greifen für den Erstzugriff auf Protokolle zurück, die wir als direkte oder indirekte Weiterentwicklungen des ursprünglichen ALOHA-Protokolls ansehen dürfen.

### Frequency Hopping Spread Spectrum

In der Praxis wird das TDMA-Verfahren häufig mit dem FDMA-Verfahren kombiniert. In diesem Fall wird die Zeitachse in Intervalle unterteilt und in jedem dieser Intervalle neu festgelegt, welches Frequenzband ein Sender nutzen darf. Die Zuteilung der Frequenzbänder kann dabei entweder koordiniert oder unkoordiniert erfolgen. Die koordinierte Zuordnung hat den Vorteil, dass die Mehrfachbenutzung des gleichen Frequenzbands unterbunden werden kann, allerdings ist hierfür eine aktive Steuerung der Sender erforderlich.

Die unkoordinierte Variante wirkt auf den ersten Blick naiv, entpuppt sich aber bei näherer Betrachtung als eine clevere Lösung: Sie funktioniert so, dass ein Sender am Ende eines Zeitschlitzes zufällig ein neues

Frequenzband auswählt und schlicht darauf hofft, dass kein anderer das gleiche Band benutzt. Passiert dies doch, so führt die Überlagerung der Signale zu einer Zerstörung der Datenpakete. In diesem Fall wird das Datenpaket in späteren Zeitschlitten so lange noch einmal gesendet, bis eine Frequenzkollision ausbleibt.

Als Ergebnis erhalten wir ein typisches Medienbelegungsmuster, wie es in Abbildung 8.30 zu sehen ist. In diesem Beispiel sind zwei Sender beteiligt, die ihre Sendefrequenz in jedem Zeitfenster pseudozufällig wählen. Jeder Sprung in der Verlaufskurve wird als *Hop* bezeichnet und das Verfahren selbst als *Frequency Hopping*. Ein Blick auf die überlagerte Kurve macht klar, dass dieses Verfahren nur dann gut funktioniert, wenn die Frequenzfolgen der beteiligten Sender nicht miteinander korreliert sind. In der Praxis wird dies durch den Einsatz von Pseudozufallsfolgen erreicht, die eine fast zufällige Verteilung der Sendefrequenzen gewährleisten.

Durch die ständig wechselnden Sendefrequenzen entsteht eine *Frequenzspreizung*. Diese Eigenschaft ist so elementar, dass sie auch für den Namen dieses Verfahrens Pate steht: Es wird in der Literatur als *Frequency Hopping Spread Spectrum*, kurz FHSS, bezeichnet.

Das Verfahren kommt in zwei Untervarianten vor: dem *Slow Frequency Hopping* (SFH) und dem *Fast Frequency Hopping* (FFH). Beide sind folgendermaßen charakterisiert:

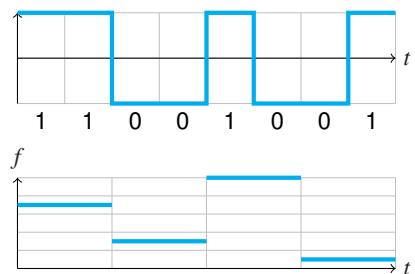
- Slow Frequency Hopping (SFH)  
Zwischen zwei Hops wird mindestens 1 Bit übertragen.
- Fast Frequency Hopping (FFH)  
Zwischen zwei Hops wird weniger als 1 Bit übertragen.

Abbildung 8.31 demonstriert die beiden Varianten an einem konkreten Beispiel. In der ersten Variante werden 2 Bits pro Hop übertragen (*slow hopping*). In der zweiten Variante wechselt die Frequenz so häufig, dass ein Bit erst nach drei Hops vollständig übertragen ist (*fast hopping*).

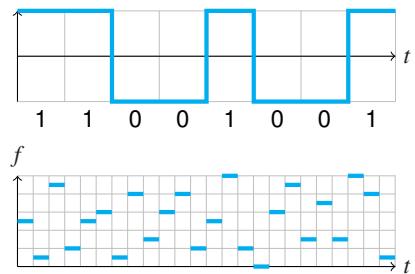
## NAVSTAR GPS

Wir kommen an dieser Stelle auf unser Eingangsbeispiel zurück: die satellitengestützte Positionsbestimmung. Weiter oben haben wir in diesem Zusammenhang über GLONASS gesprochen, aber kaum ein Wort

Slow Frequency Hopping (SFH)



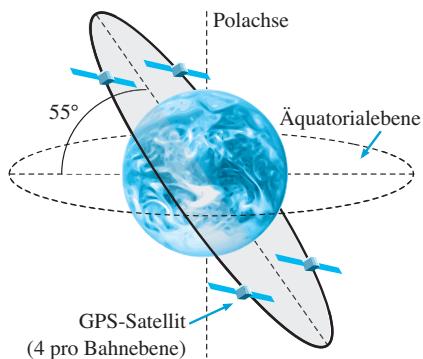
Fast Frequency Hopping (FFH)



**Abb. 8.31:** Von *Slow Frequency Hopping* (SFH) sprechen wir immer dann, wenn ein oder mehrere Bits innerhalb eines Intervalls übertragen werden. Wechselt während der Übertragung eines Bits mindestens einmal die Frequenz, so sprechen wir von *Fast Frequency Hopping* (FFH).



**Abb. 8.32:** Fotomontage eines GPS-Satelliten des Typs Navstar-IIF



**Abb. 8.33:** Die 24 GPS-Satelliten umkreisen auf 6 Bahnebenen die Erde, die zur Äquatorialebene einen Winkel von  $55^\circ$  aufweisen. Um die Grafik übersichtlich zu halten, ist nur eine Ebene eingezeichnet. Die anderen entstehen, indem die eingezeichnete Ebene in  $60^\circ$ -Schritten um die Polachse gedreht wird.

über das viel bekanntere *Global Positioning System (NAVSTAR GPS)* verloren (Abbildung 8.32).

Tatsächlich weisen beide Systeme viele Gemeinsamkeiten auf. Jeweils 24 Satelliten umrunden auf kreisförmigen Umlaufbahnen die Erde, und die Flughöhe ist ebenfalls ähnlich: Mit einem Abstand von 20.200 km sind die GPS-Satelliten ein wenig weiter von der Erde entfernt als die GLONASS-Satelliten, die ihre Signale aus 19.100 km Höhe abstrahlen.

Ebenfalls vergleichbar sind die angebotenen Dienste. Genau wie die GLONASS-Satelliten senden die GSP-Satelliten ihre Positionsdaten in zwei unterschiedlichen Formaten auf die Erde. Für die zivile Nutzung ist der C/A-Code (Coarse/Acquisition) vorgesehen, der von allen im Handel erhältlichen GPS-Empfängern ausgewertet werden kann. Daneben existiert mit dem P/Y-Code (Precision/encrYpted) ein zweites Signal, das der militärischen Nutzung vorbehalten ist.

Die Orbitalbahnen, auf denen sich die Satelliten bewegen, sind bei beiden Systemen unterschiedlich. Während bei GLONASS jeweils 8 Satelliten auf 3 Bahnebenen angeordnet sind, verlaufen die Orbits der GPS-Satelliten auf 6 verschiedenen Bahnebenen (Abbildung 8.33). Diese sind gegenüber der Äquatorialebene um  $55^\circ$  geneigt und gleichmäßig, im Winkelabstand von jeweils  $60^\circ$ , um die Polachse angeordnet. Das bedeutet, dass wir zwei Bahnebenen deckungsgleich übereinander legen können, wenn wir sie um ein Vielfaches von  $60^\circ$  um die Polachse drehen.

Wir wollen unsere Aufmerksamkeit auf ein weiteres Unterscheidungsmerkmal zwischen den beiden Satellitensystemen richten, das zunächst sehr unscheinbar daher kommt. Sehen wir nach, auf welchen Frequenzen die GPS-Satelliten ihre Signale abstrahlen, so fällt auf, dass hierfür viel weniger Frequenzen vorgesehen sind, als es bei GLONASS der Fall ist. Im Einzelnen sind dies die folgenden drei:

$$\text{L1-Frequenz: } f_{\text{L1}} = 1575,42 \text{ MHz}$$

$$\text{L2-Frequenz: } f_{\text{L2}} = 1227,60 \text{ MHz}$$

$$\text{L5-Frequenz: } f_{\text{L5}} = 1176,45 \text{ MHz}$$

Anders als die GLONASS-Satelliten, die eine gegenseitige Störung durch die Verwendung unterschiedlicher Frequenzen vermeiden, senden alle GPS-Satelliten gleichzeitig auf derselben Frequenz. Aber wie ist dies überhaupt möglich? Muss die Überlagerung der Signale nicht zu einem Signalchaos führen, das sämtliche Information zerstört? Die Antwort lautet Nein und im nächsten Abschnitt werden wir aufdecken, wie dem GPS-Empfänger dieses beeindruckende Kunststück gelingt. Seien Sie gespannt!

### 8.4.4 Codemultiplexverfahren

Wir wollen die Kernidee des Codemultiplexverfahrens (*Code Division Multiple Access*, kurz CDMA) anhand eines konkreten Beispiels herausarbeiten und nehmen an, dass vier Sender gleichzeitig eine Nachricht übermitteln möchten. Konkret handelt es sich um die folgenden Bitsequenzen, die von den vier Sendern mithilfe eines Leitungscodierers erzeugt wurden:

Sender 1 : 1010

Sender 2 : 0011

Sender 3 : 0110

Sender 4 : 1101

Die mathematische Modellierung des Codemultiplexverfahrens gelingt sehr einfach, wenn anstelle der binären Werte 0 und 1 die Zahlen  $-1$  und  $+1$  verwendet werden. Aus diesem Grund werden wir eine Nachricht im Folgenden als einen Vektor auffassen, in dem die Zahlen  $-1$  und  $1$  vorkommen. Das bedeutet, dass sich hinter den oben angegebenen Bitsequenzen in Wirklichkeit die folgenden vier Vektoren verbergen:

Sender 1 :  $(1, -1, 1, -1)$

Sender 2 :  $(-1, -1, 1, 1)$

Sender 3 :  $(-1, 1, 1, -1)$

Sender 4 :  $(1, 1, -1, 1)$

Das Codemultiplexverfahren basiert auf der Idee, ein einzelnes Bit durch eine *Chipsequenz* darzustellen, die sich ebenfalls aus den Werten  $-1$  und  $1$  zusammensetzt. Um einer begrifflichen Verwirrung vorzubeugen, benutzen wir für die Werte  $1$  und  $-1$  die folgende Terminologie: Stammt der Wert  $-1$  oder  $1$  aus einer der zu übertragenden Nachrichten, so sprechen wir von einem Bit. Ist der Wert  $-1$  oder  $1$  einer Chipsequenz entnommen, so sprechen wir von einem *Chip* (Abbildung 8.34).

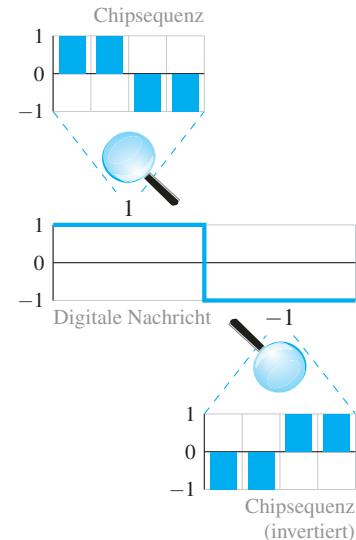
In unserem Beispiel nehmen wir an, dass ein Bit mit jeweils 4 Chips dargestellt wird und die Sender für die Codierung ihrer Nachrichten die folgenden Chipsequenzen verwenden:

Chipsequenz 1 :  $(1, 1, 1, 1)$

Chipsequenz 2 :  $(1, 1, -1, -1)$

Chipsequenz 3 :  $(1, -1, 1, -1)$

Chipsequenz 4 :  $(1, -1, -1, 1)$



**Abb. 8.34:** Das Codemultiplexverfahren basiert auf der Idee, ein einzelnes Bit des Datenstroms durch eine binäre Chipsequenz darzustellen. Ob eine  $1$  oder eine  $-1$  codiert wird, hängt davon ab, ob die Chipsequenz unverändert oder invertiert erscheint.

Um Chipsequenzen übersichtlich darzustellen, benutzen wir eine Kurzschreibweise, die den Wert 1 durch das Symbol „+“ und den Wert  $-1$  durch das Symbol „-“ ersetzt. In dieser Schreibweise lesen sich die Chipsequenzen so:

Chipsequenz 1 : + + + +

Chipsequenz 2 : + + - -

Chipsequenz 3 : + - + -

Chipsequenz 4 : + - - +

Behalten Sie stets im Gedächtnis, dass diese Kurzschreibweise lediglich dazu gedacht ist, um die Lesbarkeit zu erhöhen. Mathematisch gesehen sind Chipsequenzen, genau wie die zu übermittelnden Nachrichten, Vektoren über der Menge  $\{-1, 1\}$ .

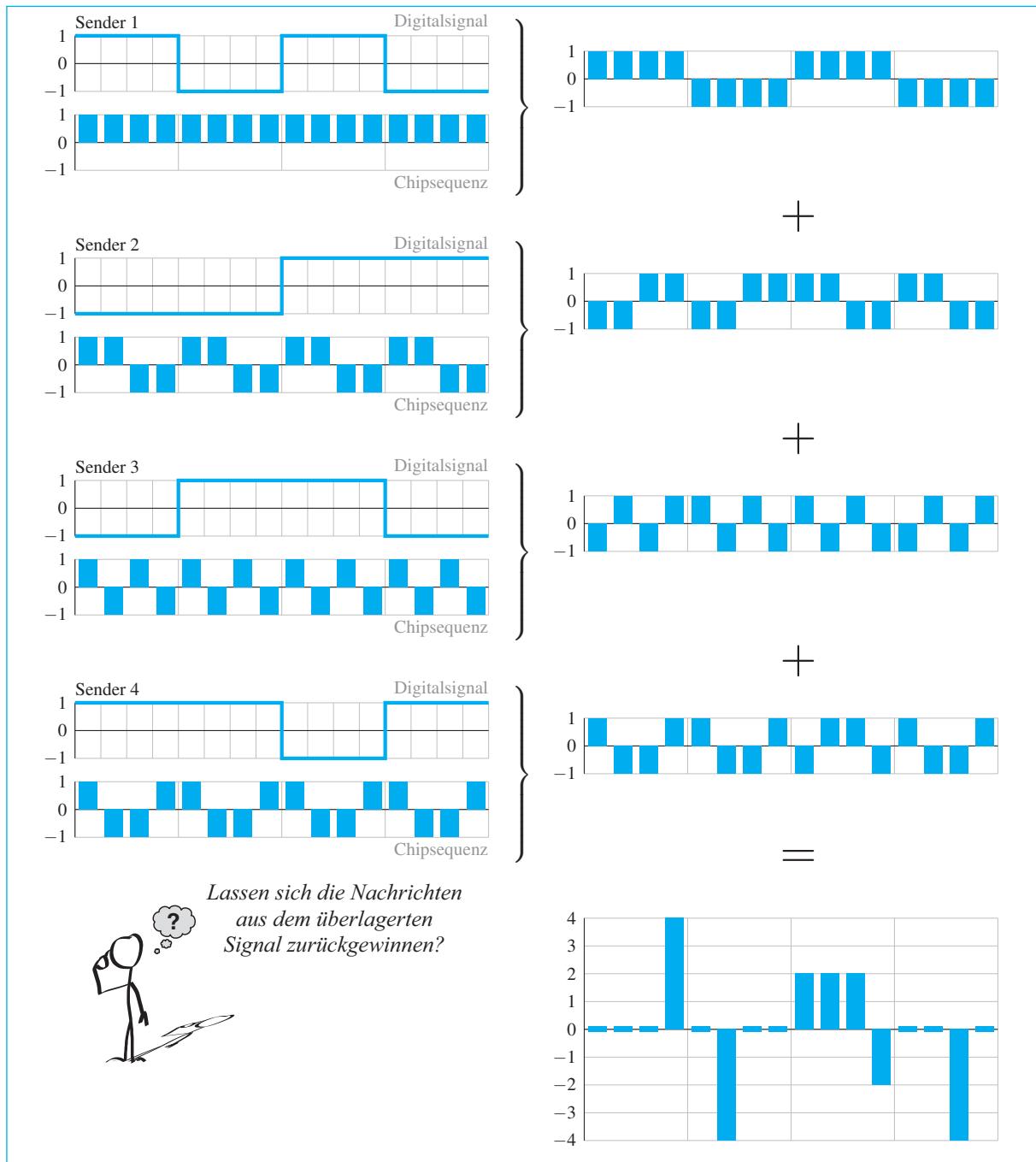
Abbildung 8.35 zeigt an einem konkreten Beispiel, wie das Codemultiplexverfahren funktioniert. Jeder der vier Sender codiert die zu übertragenden Datenbits, indem er die ihm zugewiesene Chipsequenz mehrfach wiederholt. Immer dann, wenn eine 1 übertragen werden soll, wird die Chipsequenz unverändert wiedergegeben, und immer dann, wenn eine 0 übertragen werden soll, wird sie invertiert. In unserem Beispiel produziert jeder Sender 16 Chips, um seine 4 Nachrichtenbits zu codieren.

Nachdem die Codierung abgeschlossen ist, werden die vier Signale einzeln ausgesendet. Den Empfänger erreichen sie in Form eines einzigen, überlagerten Summensignals  $s$ , das in unserem Beispiel folgendermaßen aussieht:

$$s = (0 \ 0 \ 0 \ 4 \ 0 \ -4 \ 0 \ 0 \ 2 \ 2 \ 2 \ -2 \ 0 \ 0 \ -4 \ 0)$$

Das Codemultiplexverfahren basiert auf der Eigenschaft, dass sich die vier Einzelsignale auf einfache Weise aus dem Summensignal rekonstruieren lassen. Um beispielsweise das erste Bit zu rekonstruieren, muss ein Empfänger lediglich die ersten vier Chips des Summensignals mit der Chipsequenz des gewünschten Senders skalarmultiplizieren und das Ergebnis anschließend durch 4 dividieren. Abbildung 8.36 zeigt für alle vier Sender, wie sich die jeweils gesendeten Bitsequenzen auf diese Weise aus dem Summensignal rekonstruieren lassen.

Wir wollen mit einem mathematischen Argument offenlegen, was hier vor sich geht. In unserer Überlegung bezeichnen  $c_1, \dots, c_n$  die Chipsequenzen, die von  $n$  verschiedenen Sendern für die Codierung ihrer Datenbits verwendet werden. Ferner nehmen wir an, dass jeder Sender genau 1 Bit codiert, und bezeichnen das Bit des  $i$ -ten Senders mit  $b_i$



**Abb. 8.35:** Das Codemultiplexverfahren in Aktion

## Sender 1: Chipsequenz (1 1 1 1)

$$\underbrace{\frac{1}{4} \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \\ 4 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}}_{\text{1}} = \underbrace{\frac{1}{4} \cdot \begin{pmatrix} 0 \\ -4 \\ 0 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}}_{\text{0}} = \underbrace{-\frac{1}{4} \cdot \begin{pmatrix} 2 \\ 2 \\ 2 \\ -2 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}}_{\text{-1}} = \underbrace{\frac{1}{4} \cdot \begin{pmatrix} 0 \\ 0 \\ -4 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}}_{\text{-1}}$$



## Sender 2: Chipsequenz (1 1 -1 -1)

$$\underbrace{\frac{1}{4} \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \\ 4 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ -1 \\ -1 \end{pmatrix}}_{\text{1}} = \underbrace{-\frac{1}{4} \cdot \begin{pmatrix} 0 \\ -4 \\ 0 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ -1 \\ -1 \end{pmatrix}}_{\text{0}} = \underbrace{-\frac{1}{4} \cdot \begin{pmatrix} 2 \\ 2 \\ 2 \\ -2 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ -1 \\ -1 \end{pmatrix}}_{\text{1}} = \underbrace{\frac{1}{4} \cdot \begin{pmatrix} 0 \\ 0 \\ -4 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ -1 \\ -1 \end{pmatrix}}_{\text{1}}$$



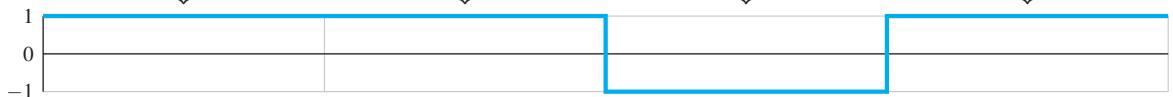
## Sender 3: Chipsequenz (1 -1 1 -1)

$$\underbrace{\frac{1}{4} \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \\ 4 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix}}_{\text{1}} = \underbrace{-\frac{1}{4} \cdot \begin{pmatrix} 0 \\ -4 \\ 0 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix}}_{\text{0}} = \underbrace{\frac{1}{4} \cdot \begin{pmatrix} 2 \\ 2 \\ 2 \\ -2 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix}}_{\text{1}} = \underbrace{\frac{1}{4} \cdot \begin{pmatrix} 0 \\ 0 \\ -4 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix}}_{\text{-1}}$$



## Sender 4: Chipsequenz (1 -1 -1 1)

$$\underbrace{\frac{1}{4} \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \\ 4 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \end{pmatrix}}_{\text{1}} = \underbrace{1 \frac{1}{4} \cdot \begin{pmatrix} 0 \\ -4 \\ 0 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \end{pmatrix}}_{\text{0}} = \underbrace{\frac{1}{4} \cdot \begin{pmatrix} 2 \\ 2 \\ 2 \\ -2 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \end{pmatrix}}_{\text{-1}} = \underbrace{-\frac{1}{4} \cdot \begin{pmatrix} 0 \\ 0 \\ -4 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \end{pmatrix}}_{\text{1}}$$



**Abb. 8.36:** Auf der Empfängerseite lässt sich die Information eines bestimmten Senders rekonstruieren, indem das überlagerte Signal mit dessen Chipsequenz multipliziert wird.

$(b_i \in \{-1, 1\})$ . Dann können wir das überlagerte Sendesignal folgendermaßen beschreiben:

$$s = \sum_{j=1}^n b_j c_j$$

Als Nächstes wollen wir abklären, wie sich dieses Signal auf der Empfängerseite verhält, wenn es dort mit der Chipsequenz des  $i$ -ten Senders multipliziert wird. Die folgende Rechnung gibt die Antwort:

$$\begin{aligned} c_i \cdot s &= c_i \sum_{j=1}^n b_j \cdot c_j = \sum_{j=1}^n b_j (c_i \cdot c_j) \\ &= \sum_{\substack{j=1 \\ j \neq i}}^n b_j (c_i \cdot c_j) + b_i (c_i \cdot c_i) \end{aligned} \quad (8.1)$$

An dieser Stelle kommt eine entscheidende Eigenschaft unserer Chipsequenzen ins Spiel. Greifen wir zwei beliebige heraus, so können wir feststellen, dass diese jeweils senkrecht aufeinander stehen:

$$c_i \perp c_j \quad (\text{für } i \neq j)$$

Die Orthogonalität ist gleichbedeutend mit der Eigenschaft, dass das Skalarprodukt zweier Chipsequenzen immer 0 ergibt:

$$c_i \cdot c_j = 0 \quad (\text{für } i \neq j)$$

Damit lässt sich Gleichung (8.1) deutlich vereinfachen. Wir erhalten:

$$c_i \cdot s = b_i (c_i \cdot c_i) \quad (8.2)$$

Da die Chipsequenzen nur die Werte  $-1$  und  $1$  enthalten, entspricht das Produkt  $c_i \cdot c_i$  der Länge der Chipsequenz. Bezeichnen wir diese mit  $|c_i|$ , so können wir die Formel (8.2) folgendermaßen umformen:

$$b_i = \frac{c_i \cdot s}{|c_i|} \quad (8.3)$$

Jetzt steht das Ergebnis schwarz auf weiß vor uns: Multiplizieren wir das empfangene Summensignal mit der Chipsequenz des  $i$ -ten Senders, so entspricht das Ergebnis – bis auf den konstanten Faktor  $|c_i|$  – dem gesendeten Datenbit. In unserem Beispiel konnten wir genau dies beobachten. Da wir Chipsequenzen der Länge 4 verwendet haben, ergab das Skalarprodukt stets einen der Werte 4 oder  $-4$  als Ergebnis. Damit ist die Kernidee, auf dem das Codemultiplexverfahren beruht, offengelegt:



Mit Gleichung (8.3) können wir eine Frage beantworten, die sich bei der praktischen Verwendung des CDMA-Verfahrens unweigerlich stellt. Wie beeinflusst die Länge der Chipsequenz die Störsicherheit? Die Antwort liegt vor uns. Da die Multiplikation des Summensignals mit der Chipsequenz einen Wert ergibt, dessen Betrag mit der Länge der Chipsequenzen identisch ist, sorgt eine Verlängerung der Chipsequenzen dafür, dass das empfangene Signal immer deutlicher wird. Eine hohe Fehlerwahrscheinlichkeit lässt sich daher durch die Verwendung langer Chipsequenzen ausgleichen, und dies ist auch einer der Gründe dafür, dass in der Praxis viel längere Chipsequenzen eingesetzt werden, als wir es in unserem Beispiel getan haben. Das Satellitenavigationssystem GPS benutzt beispielsweise Chipsequenzen der Länge 1023, die für jedes Bit 20-mal hintereinander wiederholt werden. Die langen Chipsequenzen führen dazu, dass sich die Datenrate auf magere 50 Bit pro Sekunde reduziert, gleichsam sind sie der Schlüssel dafür, dass der Empfang des GPS-Signals überhaupt in der von uns gewohnten Zuverlässigkeit funktioniert.

Es ist die Orthogonalität der Chipsequenzen, die für das Funktionieren des Verfahrens verantwortlich zeichnet.

Wir wissen nun, dass der Erfolg einer CDMA-Übertragung in wesentlichen Teilen darauf beruht, die verschiedenen Sender mit passenden Chipsequenzen auszustatten. Nur wenn es gelingt, einen paarweise orthogonalen oder „fast“ orthogonalen Satz von Chipsequenzen zu finden, lassen sich die gesendeten Bits sauber aus dem Summensignal herausrechnen. Glücklicherweise ist die Konstruktion solcher Sequenzen, die in der Literatur als *Spreizcodes* bezeichnet werden, gut untersucht.

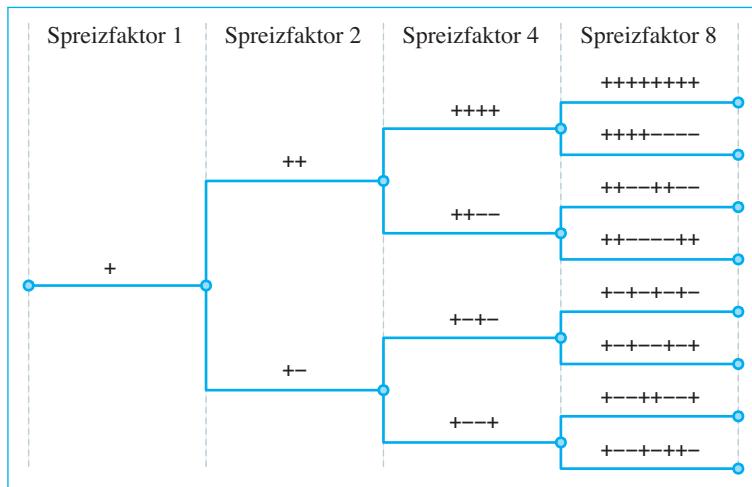
## 8.5 Spreizcodes

Wir beginnen mit einem Blick auf die sogenannten *OVSF-Codes*, deren Name aus dem englischen Begriff *Orthogonal Variable Spreading Factor* abgeleitet ist. Bereits der Name gibt Auskunft darüber, dass wir es mit einem *orthogonalen Code* zu tun haben, d. h. mit einem Code, der aus paarweise aufeinander senkrecht stehenden Chipsequenzen besteht. Diese Eigenschaft wiesen auch die Sequenzen auf, die wir weiter oben verwendet haben, und dieser Zusammenhang kommt nicht von ungefähr. Ohne es zu benennen, haben wir in unserem Beispiel bereits einen OVSF-Code benutzt: den OVSF-Code mit dem *Spreizfaktor 4*. Was sich dahinter genau verbirgt, werden wir jetzt klären.

### 8.5.1 OVSF-Codes

In der Literatur wird für die Konstruktion der OVSF-Codes gerne auf eine Baumdarstellung zurückgegriffen, wie sie in Abbildung 8.37 zu sehen ist. Der Baum ist so gezeichnet, dass sich seine Wurzel links befindet und sich seine Pfade nach rechts ausbreiten. Die Wurzel repräsentiert den OVSF-Code mit dem Spreizfaktor 1. Er besteht aus einem einzigen Codewort, das in der Baumdarstellung in der Kurzform ,+‘ notiert ist, und mathematisch dem einelementigen Vektor (1) entspricht.

Die OVSF-Codes mit höheren Spreizfaktoren entstehen, indem der Baum von links nach rechts weitergezeichnet und dabei jedes Mal in einen oberen und einen unteren Zweig aufgespalten wird. In jedem dieser Schritte wird die Codewortlänge verdoppelt. Das obere Codewort wird gebildet, indem das Vatercodewort zweimal wiederholt wird. Das untere Codewort wird gebildet, indem das Vatercodewort invertiert an sich selbst angehängt wird. Wie die Grafik zeigt, ist der *Spreizfaktor* mit



**Abb. 8.37:** Baumdarstellung von OVSF-Codes. Die Codewörter mit dem Spreizfaktor  $2^{n+1}$  entstehen aus den Codewörtern mit dem Spreizfaktor  $2^n$  durch das Prinzip der Selbstwiederholung. Hierbei wird jedes Codewort einmal unverändert und einmal negiert an sich selbst angefügt.

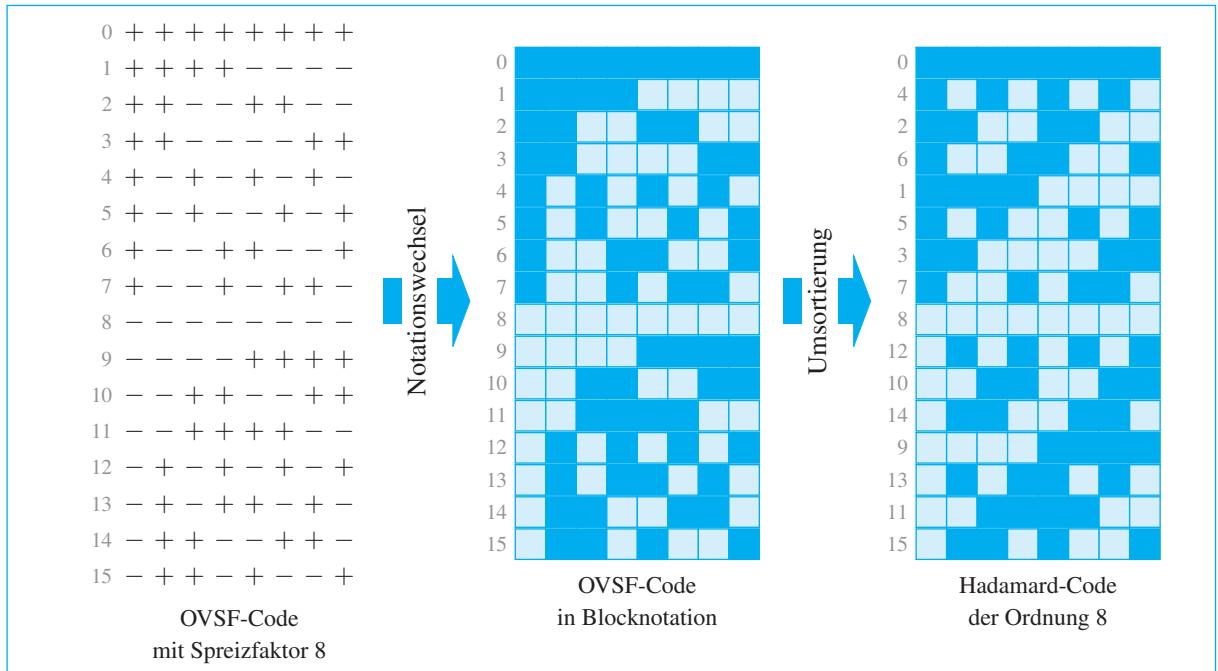
der Anzahl der Chips identisch, aus denen sich die einzelnen Codewörter zusammensetzen.

Beachten Sie, dass in Abbildung 8.37 nur die Hälfte der Codewörter dargestellt sind. Da die Chipsequenz für die Codierung einer 0 mit  $-1$  multipliziert wird, sind die negierten Chipsequenzen ebenfalls Codewörter. Das bedeutet, dass der OVSF-Code mit dem Spreizfaktor  $n$  aus insgesamt  $2n$  Codewörtern besteht, und wir jeweils 2 dieser Codewörter einer bestimmten Chipsequenz zuordnen können.

Hat Sie der Eindruck beschlichen, dass Ihnen in diesem Abschnitt alter Wein in neuen Schläuchen dargeboten wird? Falls ja, dann haben Sie gut aufgepasst! Dass die OVSF-Codes für uns überhaupt nichts Neues sind, wird klar, wenn wir die Codewörter, wie in Abbildung 8.38 geschehen, in die Blockschreibweise übersetzen und geringfügig umsortieren. Als Ergebnis erhalten wir eine Darstellung, die Sie auf Seite 448 in Abbildung 6.80 schon einmal gesehen haben. Sie macht deutlich, dass der OVSF-Code mit dem Spreizfaktor  $n$  aus den gleichen Codewörtern besteht wie der Hadamard-Code der Ordnung  $n$ . Wir können diesen Zusammenhang noch prägnanter formulieren: Die OVSF-Codes sind die Hadamard-Codes.

## Synchrones CDMA

An dieser Stelle wollen wir unser Augenmerk auf ein wichtiges Verhaltensmerkmal von Sendern lenken, die ihre Signale CDMA-codiert



**Abb. 8.38:** Ein Notationswechsel und die anschließende Umsortierung der Codewörter reichen aus, um die wahre Natur der OVSF-Codes zu entlarven: Hinter den OVSF-Codes verbergen sich die Hadamard-Codes aus Abschnitt 6.4.6.

abstrahlen. Wir sind stillschweigend davon ausgegangen, dass alle Sender synchronisiert arbeiten, den ersten Chip einer Sequenz also immer zu den gleichen Zeitpunkten aussenden. Ist dies der Fall, so sprechen wird von *synchronem CDMA*, und die OVSF-Codes sind gut geeignet, um die Daten der beteiligten Sender zu codieren.

### Asynchrones CDMA

Leider können wir in der Praxis nur in seltenen Fällen davon ausgehen, dass die Chipsequenzen in einer zeitlich synchronisierten Abfolge den Empfänger erreichen. Ein Beispiel hierfür ist das Global Positioning System GPS, mit dem wir uns weiter oben beschäftigt haben. Hier wäre der gleichzeitige Empfang zweier Datenpakete reiner Zufall, da zwei gleichzeitig abgestrahlte Chipsequenzen aufgrund der unterschiedlichen Entfernungen zwischen den Satelliten und dem Empfänger zeitlich versetzt ankommen. In diesem Fall sprechen wir von *asynchronem CDMA*.

Für die Bewertung, ob ein Spreizcode für das asynchrone Codemultiplexverfahren geeignet ist, reicht es nicht aus, das Skalarprodukt zwischen den Chipsequenzen zu betrachten. Wir müssen untersuchen, wie sich die Chipsequenzen verhalten, wenn wir sie mit zyklischen Verschiebungen der anderen Sequenzen multiplizieren. Zu diesem Zweck treffen wir die folgende begriffliche Vereinbarung:



### Definition 8.1 (Korrelationsprodukt)

Es seien  $c_i$  und  $c_j$  zwei Chipsequenzen und  $\delta \in \mathbb{N}$ . Wir nennen

$$R_{ij}(\delta) := c_i \cdot (c_j \ll \delta)$$

das *diskrete Korrelationsprodukt* von  $c_i$  und  $c_j$ .

In dieser Definition bezeichnet  $(c_j \ll \delta)$  die Chipsequenz, die aus  $c_j$  durch die Rotation um  $\delta$  Stellen nach links hervorgeht. Setzen wir  $\delta = 0$ , so ist das Korrelationsprodukt mit dem Skalarprodukt identisch.

Für die Chipsequenzen

$$c_1 = +--+$$

$$c_2 = ++-+$$

erhalten wir die folgenden Korrelationsprodukte:

$$R_{12}(0) = \begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \\ -1 \end{pmatrix} = -2$$

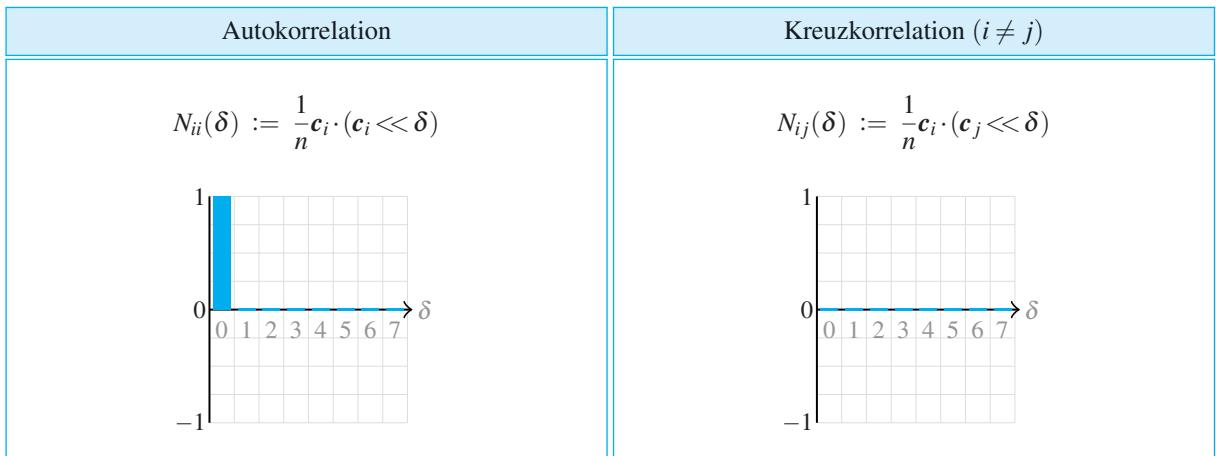
$$R_{12}(1) = \begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ -1 \\ 1 \end{pmatrix} = 2$$

$$R_{12}(2) = \begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ -1 \\ 1 \\ 1 \end{pmatrix} = 2$$

$$R_{12}(3) = \begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} -1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = -2$$



In der Darstellung des asynchronen CDMA-Szenarios haben wir mehrere Vereinfachungen vorgenommen, die an dieser Stelle nicht unerwähnt bleiben sollen. Zum einen haben wir eine binäre Sichtweise angelegt und angenommen, dass alle gesendeten Signale mit dem Gewicht 1 oder  $-1$  in das Summensignal eingehen. In der Realität ist dies keineswegs der Fall, wie ein Szenario mit mehreren verteilten Funksendern zeigt. Selbst wenn alle Sender ihre elektromagnetischen Wellen mit der gleichen Leistung abstrahlen, wird das Signal eines weit entfernten Senders das Summensignal viel weniger beeinflussen als das Signal eines Senders, der sich in unmittelbarer Nähe befindet. Ein anderes Problem wird durch unterschiedliche Laufzeiten verursacht. In unseren Überlegungen sind wir zwar nicht, wie bei den synchronen CDMA-Szenarien, davon ausgegangen, dass zwei Chipsequenzen zeitlich synchronisiert eintreffen, für einzelne Chips haben wir diese Annahme aber dennoch beibehalten. In der Realität ist auch dies nicht der Fall. In den Signalen, die wir real empfangen können, sind die Chips in der Regel nicht deckungsgleich, sondern überlagern sich auch dort leicht versetzt. Die genannten Vereinfachungen wurden bewusst gemacht, um die einführende Darstellung in die CDMA-Technik nicht zu überfrachten. In der Praxis dürfen diese Effekte natürlich nicht vernachlässigt werden. Sie sind der Grund, warum dort in aller Regel mit deutlich mehr Redundanz gearbeitet wird, als es die Theorie erfordert.



**Abb. 8.39:** Für das asynchrone CDMA-Verfahren werden Chipsequenzen benötigt, deren Korrelationsprodukte, abgesehen von der Autokorrelation mit  $\delta = 0$ , nahe bei 0 liegen.

Um die Korrelationsprodukte von Spreizcodes vergleichen zu können, führen wir den Begriff des *normalisierten Korrelationsprodukts* ein, der von der Länge der Chipsequenzen abstrahiert:

### **Definition 8.2**

Ist  $n$  die Länge der Chipsequenzen  $\mathbf{c}_i$  und  $\mathbf{c}_j$ , so heißt

$$N_{ij}(\delta) := \frac{1}{n} R_{ij}(\delta) = \frac{1}{n} \mathbf{c}_i \cdot (\mathbf{c}_j \ll \delta)$$

das *normalisierte Korrelationsprodukt* von  $\mathbf{c}_i$  und  $\mathbf{c}_j$ .

Im Folgenden werden wir streng unterscheiden, ob eine Chipsequenz mit sich selbst ( $i = j$ ) oder mit einer anderen Chipsequenz ( $i \neq j$ ) korreliert wird. Im ersten Fall sprechen wir von einer *Autokorrelation* oder *Selbstkorrelation* und im zweiten Fall von einer *Kreuzkorrelation*.

Abbildung 8.39 zeigt, welche Kriterien ein Spreizcode erfüllt, der für das asynchrone Codemultiplexverfahren optimal geeignet ist.

- Das normalisierte Autokorrelationsprodukt einer Chipsequenz ist für  $\delta = 0$  gleich 1 und für  $\delta \neq 0$  gleich 0. Diese Eigenschaft wird für die Synchronisation des Empfängers benötigt. Fehlt sie, so kann der Empfänger nicht erkennen, wo eine Chipsequenz beginnt.

- Das normalisierte Kreuzkorrelationsprodukt zweier Chipsequenzen ist, unabhängig von der Wahl von  $\delta$ , gleich 0. Auf diese Weise ist sichergestellt, dass sich die überlagerten Signale nicht stören und auf der Empfängerseite sauber aus dem Summensignal extrahiert werden können.

Abbildung 8.40 macht deutlich, dass die OVSF-Codes die geforderten Eigenschaften nicht erfüllen. Die Orthogonalität der Chipsequenzen sorgt lediglich dafür, dass das Kreuzkorrelationsprodukt für den Versatz  $\delta = 0$  den Wert 0 ergibt; etliche andere Versätze führen jedoch ebenfalls zu merklichen Ausschlägen. Auch die Autokorrelation verfehlt unsere Anforderungen. Beispielsweise liefert das Korrelationsprodukt der Chipsequenz „++++“ mit sich selbst für alle Werte von  $\delta$  einen maximalen Wert. Der Empfänger hat in diesem Fall keine Möglichkeit, den Beginn einer Chipsequenz zu erkennen.

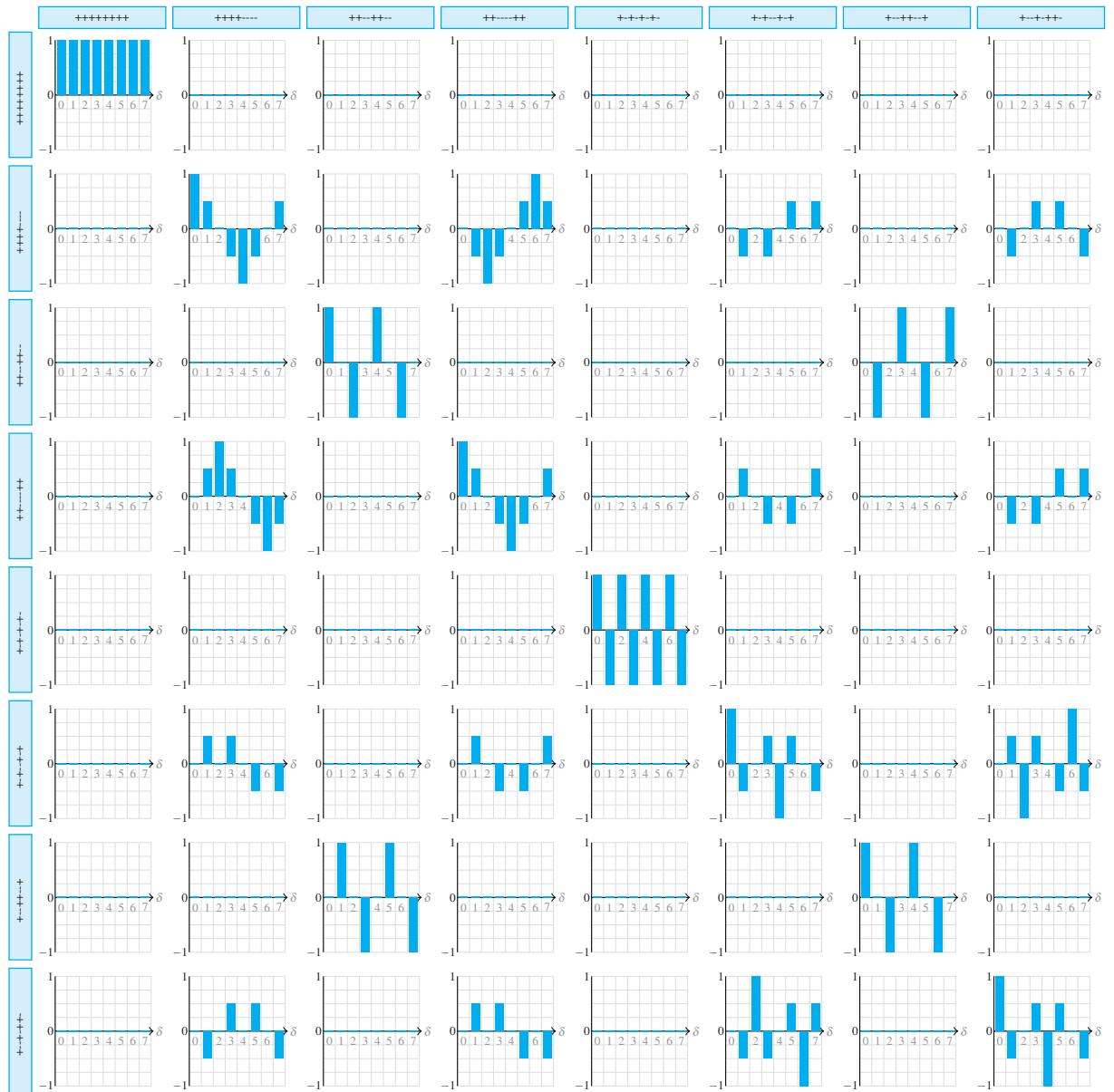
Dies wirft eine interessante Frage auf: Werden die geforderten Eigenschaften vielleicht dann erfüllt, wenn die Chips in den OVSF-Sequenzen in eine andere Reihenfolge gebracht werden? Da die Umsortierung so geschehen müsste, dass die Eigenschaft der Orthogonalität, zumindest weitgehend, gewahrt bleibt, ist die Antwort negativ. Tatsächlich sind alle Codes, die  $2n$  paarweise orthogonale Codewörter der Länge  $n$  umfassen, zu den OVSF-Codes äquivalent.

Für uns gibt es an dieser Stelle nur den Ausweg, die Länge der Chipsequenzen deutlich zu vergrößern. Ist nämlich die Anzahl der Chips sehr groß, so gibt es eine einfache Lösung für unser Problem: Die geforderten Eigenschaften lassen sich dann herstellen, indem jedem Sender eine zufällig generierte Folge zugeordnet wird. Dass dieser naiv wirkende Ansatz wirklich funktioniert, ist leicht einzusehen: Vergleichen wir zwei zufällig generierte Folgen, so unterscheiden sich diese durchschnittlich an der Hälfte der Positionen, und das Gleiche gilt, wenn wir eine solche Folge gegen sich selbst verschieben. Wenn die Lösung also so nahe liegt, bleibt nur noch eine Frage zu klären: Lassen sich Chipsequenzen, die eine nahezu zufällige Werteverteilung aufweisen, mit einem Algorithmus erzeugen, der auf den banalen Einsatz eines Würfels verzichtet? Die Antwort lautet Ja, und sie führt uns auf direktem Wege in die Tiefen der Mathematik zurück. Wir finden die Lösung in der heute gut untersuchten Theorie der Pseudozufallsfolgen.



In allen Beispielen, die wir in diesem Abschnitt betrachtet haben, entspricht die Länge der Chipsequenzen der Länge eines Bits. Damit ist gemeint, dass für die Codierung eines binären Datenstroms jedes Bit mit der gleichen Sequenz multipliziert wird und wir eine Chipsequenz deshalb als den Repräsentanten eines einzelnen Bits ansehen dürfen. In der CDMA-Terminologie wird in diesem Fall von einem *Short code* gesprochen.

Übersteigt die Länge der Chipsequenz die Länge eines einzelnen Bits, so liegt ein *Long code* vor. Verwendet werden diese beispielsweise für die Codierung des militärische P(Y)-Signals des Global Positioning Systems. Die GPS-Satelliten sind mit Code-Generatoren bestückt, die nur alle 38 Wochen in ihren Initialzustand zurückkehren. Die komplette Sequenz wird dort allerdings nicht genutzt. Nach jeweils einer Woche werden die Satelliten künstlich in ihren Initialzustand zurückversetzt.



**Abb. 8.40:** Die normalisierten Korrelationprodukte für die OVSF-Chipsequenzen mit dem Spreizfaktor 8

## 8.5.2 Pseudozufallsfolgen

Pseudozufallsfolgen lassen sich auf einfache Weise durch rückgekoppelte Schieberegister erzeugen. In ihrer einfachsten Form bestehen solche Register aus mehreren seriell verbundenen Bitspeichern, die ihren Inhalt taktsynchron an ihren jeweiligen Nachfolger weiterreichen (Abbildung 8.41). In Hardware lässt sich ein Schieberegister durch eine Reihe hintereinander geschalteter D-Flipflops implementieren.

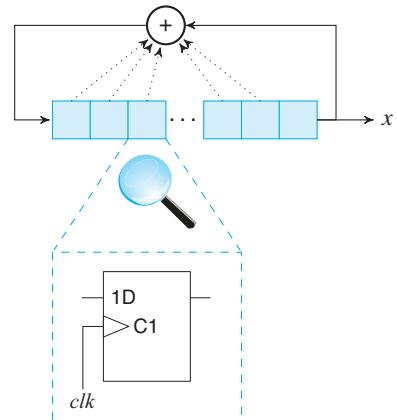
In Abbildung 8.42 sind mehrere solche Register mit jeweils 4 Schiebestufen zu sehen. Um eine Zufallsfolge zu generieren, werden die Bitspeicher in ihren Initialzustand versetzt und der Registerinhalt dann schrittweise rotiert. Das am weitesten rechts stehende Bit ist gleichzeitig die Ausgabe des Schieberegisters, d. h., es wird in jedem Schritt eine einzelne Ziffer. Gleichzeitig wird in jedem Schritt links ein neues Bit eingefügt. Dieses entsteht, indem das Ausgabebit mit einem oder mehreren anderen Registerbits XOR-verknüpft wird.

Alle Registerbits zusammen repräsentieren den *Zustand* des Schieberegisters. Da unsere Beispielregister allesamt vierstufig sind, befinden sie sich zu jedem Zeitpunkt in einem von 16 Zuständen. Wie die produzierte Zufallsfolge aussieht, hängt davon ab, welche Zustände ein Register nacheinander durchläuft. Um die Reihenfolge zu verdeutlichen, ist der aktuell eingenommene Zustand in Abbildung 8.42 in dezimaler Darstellung links neben den Tabellenzeilen notiert.

Beachten Sie, dass die Startkonfiguration einen großen Einfluss auf die erzeugten Sequenzen hat. Initialisieren wir beispielsweise alle Register mit dem Wert 0, so kommt die Erzeugung einer Zufallsfolge erst gar nicht in Fahrt. In diesem Fall ergibt die Modulo-2-Summe der Zustandsbits stets den Wert 0, sodass die Schaltung für immer in diesem im Englischen treffend als *Lock-up State* bezeichneten Zustand verharrt. Dies ist der Grund, warum wir unsere Beispielschaltungen initial in den Zustand 1, und nicht in den Zustand 0 versetzen.

Behalten Sie ferner im Gedächtnis, dass die Berechnung der Ausgabefolge, so zufällig diese auch wirken mag, deterministisch verläuft. Befindet sich ein Schieberegister zu zwei verschiedenen Zeitpunkten in demselben Zustand, so entstehen, zeitversetzt, die gleichen Zahlenfolgen. Das bedeutet, dass Pseudozufallsfolgen, die mit Schieberegistern erzeugt werden, stets periodisch sind, und die maximale Periodenlänge durch die Anzahl der Zustände nach oben begrenzt ist.

Wir können diese Abschätzung noch weiter präzisieren. Da der Zustand 0 immer zu einer Pseudozufallsfolge mit der Periode 0 führt, müssen wir



**Abb. 8.41:** Ein rückgekoppeltes Schieberegisters für die Erzeugung von Pseudozufallsfolgen ( $clk$  ist das Taktsignal)

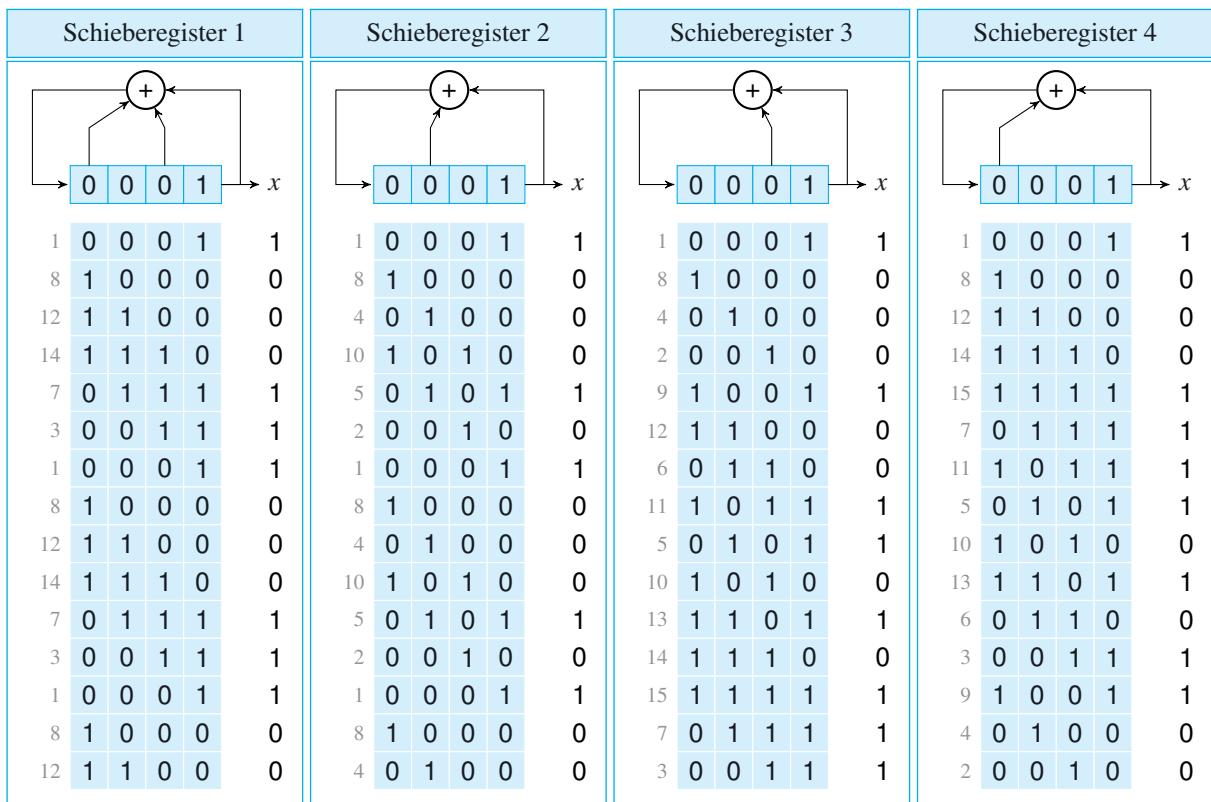


Abb. 8.42: Vierstufige Schieberegister für die Erzeugung von Pseudozufallsfolgen

ihn in unserer Betrachtung ausklammern. Das bedeutet, dass die maximale Periode einer Pseudozufallsfolge durch den Ausdruck  $2^n - 1$  nach oben begrenzt ist, wobei  $n$  die Anzahl der Schiebestufen beschreibt.

Dass es tatsächlich möglich ist, eine Periode von  $2^n - 1$  zu erzielen, stellen die beiden rechten Schieberegister in Abbildung 8.42 unter Beweis. Sie besitzen 4 Registerstufen ( $n = 4$ ) und generieren beide eine Pseudozufallsfolge der Länge  $2^4 - 1 = 15$ . Die beiden linken Schieberegister verhalten sich anders. Beide kehren bereits nach 6 Schiebeschritten in den Initialzustand zurück. Gute Pseudozufallsfolgen weisen eine möglichst lange Periodenlänge auf, und so sind die Folgen der beiden rechten Register den Folgen der beiden linken überlegen.

Zufallsfolgen mit einer maximalen Periodenlänge sind in der Codierungstheorie so wichtig, dass sie einen eigenen Namen besitzen: Sie werden im englischen Sprachraum *Maximum Length Sequences*, kurz

MLS, genannt. Wir bezeichnen sie, wie es in der deutschen Literatur üblich ist, als *Folgen maximaler Länge*.

MLS-Schieberegister, d. h. Schieberegister, die eine Folge maximaler Länge produzieren, weisen eine Reihe interessanter Eigenschaften auf. Da die formale Herleitung ein gehöriges Wissen aus der Algebra erfordert, wollen wir uns an dieser Stelle auf deren Nennung beschränken:

- Um die erste Eigenschaft überhaupt formulieren zu können, müssen wir auf eine alternative Beschreibungsform zurückgreifen. Diese basiert auf der Tatsache, dass ein Schieberegister mit einem Polynom  $p(x)$  aus der Menge  $\mathbb{Z}_2[x]$  identifiziert werden kann. Das Polynom

$$p(x) = x^n + a_0x^{n-1} + \dots + a_{n-2}x + 1$$

wird dabei so gewählt, dass der Grad  $n$  der Anzahl der Schiebestufen entspricht. Die Koeffizienten  $a_0, \dots, a_{n-2}$  repräsentieren die einzelnen Registerbits und werden genau dann auf 1 gesetzt, wenn das zugehörige Registerbit für die Berechnung der Modulo-2-Summe verwendet wird. Für unsere vier Beispielregister erhalten wir die folgenden Polynome (Abbildung 8.43):

$$p_1(x) = x^4 + x^3 + x + 1 \quad (8.4)$$

$$p_2(x) = x^4 + x^2 + 1 \quad (8.5)$$

$$p_3(x) = x^4 + x^3 + 1 \quad (8.6)$$

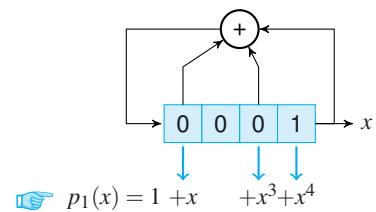
$$p_4(x) = x^4 + x + 1 \quad (8.7)$$

Als Nächstes benötigen wir den Begriff des *primitiven Polynoms*. Von einem primitiven Polynom haben wir bisher noch nicht gesprochen, dafür aber von primitiven Elementen, die eine enge Verwandtschaft aufweisen. Wir erinnern uns: Ein primitives Element  $\alpha$  eines Körpers  $\mathbb{K}$  zeichnet sich durch die Eigenschaft aus, dass seine Potenzen  $\alpha^0, \alpha^1, \alpha^2, \dots$  alle von 0 verschiedenen Körperelemente durchlaufen. Betrachten wir beispielsweise den Körper  $\mathbb{F}_{2^4}$ , den wir in Abschnitt 6.4.4.1 verwendet haben, so sind die folgenden Elemente primitiv:

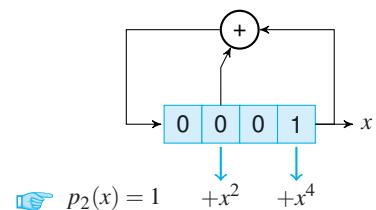
$$2, 3, 4, 5, 9, 11, 13, 14$$

Als Nächstes erinnern wir uns an den Begriff des *Minimalpolynoms*. Wir haben diesen Begriff ebenfalls in Abschnitt 6.4.4.1 eingeführt und dort gesehen, dass jedes Körperelement die Nullstelle eines Minimalpolynoms ist. Das bedeutet, dass auch jedes primitive Element die Nullstelle eines Minimalpolynoms ist, und damit sind wir begrifflich auch schon am Ziel: Ist  $\alpha$  ein primitives Element, so heißt das Minimalpolynom von  $\alpha$  ein *primitives Polynom*.

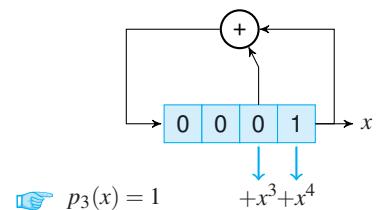
Schieberegister 1



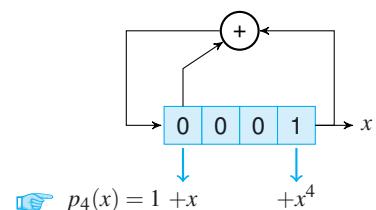
Schieberegister 2



Schieberegister 3



Schieberegister 4



**Abb. 8.43:** Jedes Schieberegister mit  $n$  Stufen lässt sich durch ein Polynom  $p(x) \in \mathbb{Z}_2[x]$  mit dem Grad  $n$  beschreiben.

Um die primitiven Polynome des Körpers  $\mathbb{F}_{2^4}$  zu bestimmen, müssen wir lediglich auf Seite 410 zurückblättern und einen Blick auf Tabelle 6.8 werfen. Aus dieser geht hervor, dass die primitiven Elemente

$$2,3,4,5 \quad (8.8)$$

eine Nullstelle des Minimalpolynoms

$$m_4(x) = x^4 + x + 1 \quad (8.9)$$

und die primitiven Elemente

$$9,11,13,14 \quad (8.10)$$

eine Nullstelle des Minimalpolynoms

$$m_5(x) = x^4 + x^3 + 1 \quad (8.11)$$

sind. Das bedeutet, dass (8.9) und (8.11) die primitiven Polynome von  $\mathbb{F}_{2^4}$  sind.

Ein Blick auf die Polynome (8.4) bis (8.7) zeigt, dass von unseren Beispielregistern genau jene eine Folge maximaler Länge hervorbringen, deren Verknüpfungsstruktur einem primitiven Polynom entspricht. Tatsächlich gilt dieser Zusammenhang für alle Schieberegister des gezeigten Typs: Genau dann, wenn das zugeordnete Polynom primitiv ist, erzeugt ein Schieberegister eine Folge maximaler Länge. Eine faszinierende Eigenschaft!

- Die zweite Eigenschaft, für die wir uns an dieser Stelle interessieren, wird im Englischen treffend als *Shift-and-add Property* bezeichnet. Für eine Sequenz  $S$  maximaler Länge besagt sie das Folgende: Wird  $S$  rotiert und das Ergebnis anschließend mit  $S$  XOR-verknüpft, so entsteht erneut eine rotierte Variante von  $S$ .

Etwas mathematischer können wir dies auch so ausdrücken. Ist  $S$  eine Sequenz maximaler Länge, so existiert für jede Zahl  $i \in \mathbb{Z}$  mit  $S \neq (S \gg i)$  ein  $j \in \mathbb{Z}$  mit

$$S \oplus (S \gg i) = S \gg j$$

Dass wir die Eigenschaft  $S \neq (S \gg i)$  fordern müssen, liegt auf der Hand: Ist  $S = (S \gg i)$ , so würden wir  $S$  mit sich selbst verknüpfen und als Ergebnis die Nullfolge erhalten. Abbildung 8.44 demonstriert das Gesagte anhand mehrerer Beispiele.

Eine reale Anwendung dieser Eigenschaft werden Sie in Abschnitt 8.5.3 kennenlernen. GPS-Satelliten machen sich das geschilderte Phänomen zu Nutze, um mehrere zeitlich verschobene Varianten einer bestimmten Chipsequenz zu erzeugen.

### Beispiel 1

1	0	0	0	0	1	0	0	1	1	0	1	0	1	1	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
1	1	0	0	0	1	0	0	1	1	0	1	0	1	1	
14	0	1	2	3	4	5	6	7	8	9	10	11	12	13	
0	1	0	0	1	1	0	1	0	1	1	1	1	0	0	
3	4	5	6	7	8	9	10	11	12	13	14	0	1	2	

$$\text{☞ } S \oplus (S \gg 1) = (S \ll 3)$$

### Beispiel 2

1	0	0	0	0	1	0	0	1	1	0	1	0	1	1	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
0	0	0	1	0	0	1	1	0	1	0	1	1	1	1	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	0	
1	0	0	1	1	0	1	0	1	1	1	1	0	0	0	
4	5	6	7	8	9	10	11	12	13	14	0	1	2	3	

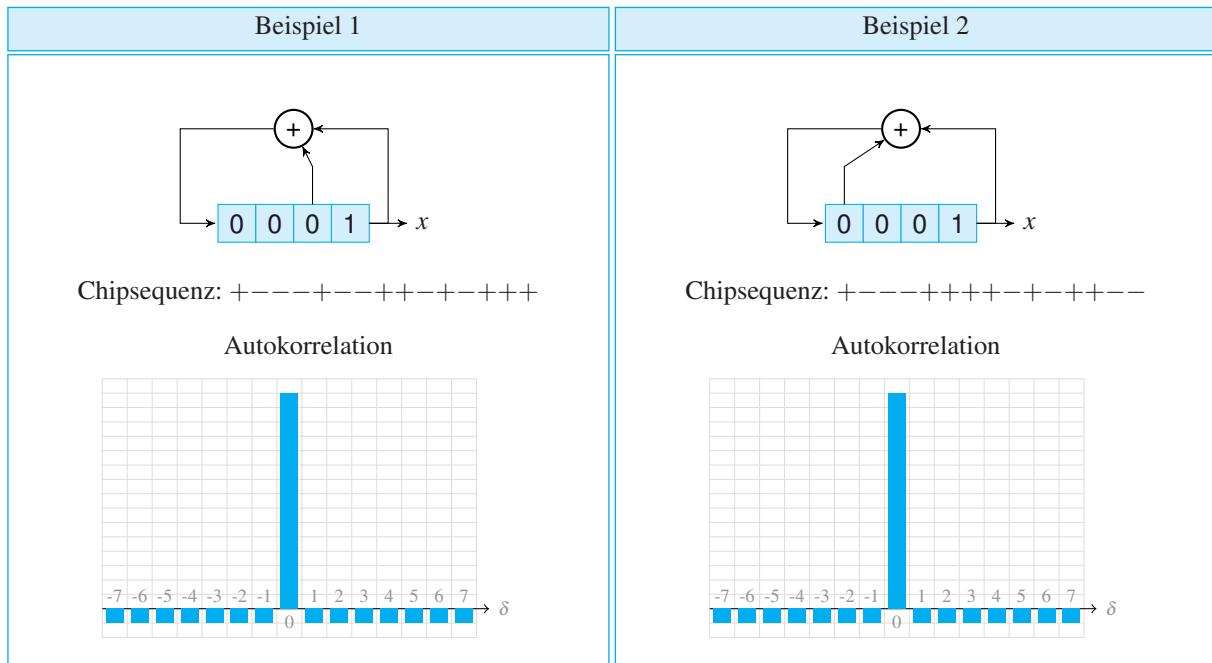
$$\text{☞ } S \oplus (S \ll 1) = (S \ll 4)$$

### Beispiel 3

1	0	0	0	0	1	0	0	1	1	0	1	0	1	1	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
0	0	1	1	0	1	0	1	1	1	1	0	0	0	1	
5	6	7	8	9	10	11	12	13	14	0	1	2	3	4	
1	0	1	1	1	1	0	0	0	1	0	0	1	1	0	
10	11	12	13	14	0	1	2	3	4	5	6	7	8	9	

$$\text{☞ } S \oplus (S \ll 5) = (S \ll 10)$$

**Abb. 8.44:** Bilden wir die Modulo-2-Summe einer Sequenz maximaler Länger mit einer rotierten Variante von sich selbst, so entsteht erneut eine rotierte Variante der ursprünglichen Sequenz.



**Abb. 8.45:** Chipsequenzen, die mithilfe von MLS-Schieberegistern gewonnen werden, weisen hervorragende Autokorrelationseigenschaften auf. Neben dem Peak bei  $\delta = 0$  ergibt das Korrelationsprodukt für  $\delta \neq 0$  immer den Wert  $-1$ .

Abbildung 8.45 zeigt am Beispiel der Folgen, die von den MLS-Schieberegistern aus Abbildung 8.42 generiert werden, dass Folgen maximaler Länge hervorragende Autokorrelationseigenschaften aufweisen. Neben dem erwarteten Ausschlag bei  $\delta = 0$  erhalten wir für alle anderen Werte von  $\delta$  den Wert  $-1$ . Auch dies ist kein Zufall! Es lässt sich zeigen, dass ausnahmslos jede Folge maximaler Länge den gezeigten Werteverlauf hervorbringt. Ist  $c_i$  eine Pseudozufallsfolge der Länge  $n$ , so gilt für das Autokorrelationsprodukt die folgende Formel:

$$R_{ii}(\delta) = \begin{cases} n & \text{falls } \delta \bmod n = 0 \\ -1 & \text{falls } \delta \bmod n \neq 0 \end{cases}$$

Erneut zeigt sich, dass sich hinter MLS-Schieberegistern weit mehr Geheimnisse verbergen, als es ihre simple Erscheinungsform auf den ersten Blick vermuten lässt.

Gerade Registerlänge  $n$ 

$$R_{ij}(\delta) = \begin{cases} -2^{\left(\frac{n+2}{2}\right)} - 1 & (12,5\%) \\ -1 & (75\%) \\ 2^{\left(\frac{n+2}{2}\right)} - 1 & (12,5\%) \end{cases}$$

Ungerade Registerlänge  $n$ 

$$R_{ij}(\delta) = \begin{cases} -2^{\left(\frac{n+1}{2}\right)} - 1 & (25\%) \\ -1 & (50\%) \\ 2^{\left(\frac{n+1}{2}\right)} - 1 & (25\%) \end{cases}$$

**Abb. 8.46:** Gold-Folgen besitzen die Eigenschaft, dass ihr Korrelationsprodukt nur drei verschiedene Werte annehmen kann (vgl. [96]).

## 8.5.3 Gold-Folgen

Gerade haben wir gesehen, dass Folgen maximaler Länge hervorragende Autokorellationseigenschaften aufweisen. Für eine praktische Verwertung ist dies aber nur die halbe Miete. Statten wir nämlich mehrere Sender mit solchen Sequenzen aus, so müssen wir neben der Autokorrelation auch die Kreuzkorrelation im Auge behalten. Weiter oben haben wir erwähnt, dass das Kreuzkorrelationsprodukt zweier Zufallsfolgen gegen 0 konvergiert, wenn wir die Länge der Chipsequenzen kontinuierlich erhöhen. In der Praxis können wir die Chipsequenzen aber nicht beliebig lange wählen, und so bleibt die Gefahr bestehen, dass zufällig gewählte Sequenzen dennoch eine ungünstige Kreuzkorrelation aufweisen. Aus diesem Grund haben sich in der Vergangenheit mehrere Wissenschaftler mit der Frage beschäftigt, wie sich Sequenzen mit einem vorhersagbaren Kreuzkorellationsverhalten konstruieren lassen.

Im Jahr 1967 entdeckte der Mathematiker Robert Gold, dass sich solche Sequenzen erstaunlich einfach erzeugen lassen. Er fand heraus, dass unter den vielen möglichen Schieberegistern gewisse Paare existieren, die in besonderer Weise zueinander passen. Für diese Paare gilt, dass das Kreuzkorrelationsprodukt der produzierten Pseudozufallsfolgen nur drei verschiedene Werte annehmen kann. Zudem war Gold die Konstruktion zweier Formeln gelungen, mit denen sich diese Werte voraussagen lassen. Die eine Formel gilt für den Fall, dass die Anzahl der Schieberegisterstufen, wie in unserem Beispiel, gerade ist, und die andere für den Fall, dass sie ungerade ist. Abbildung 8.46 zeigt, um welche Formeln es sich im Detail handelt, und gibt zugleich Auskunft darüber, mit welchen Wahrscheinlichkeiten wir diese Werte beobachten können.

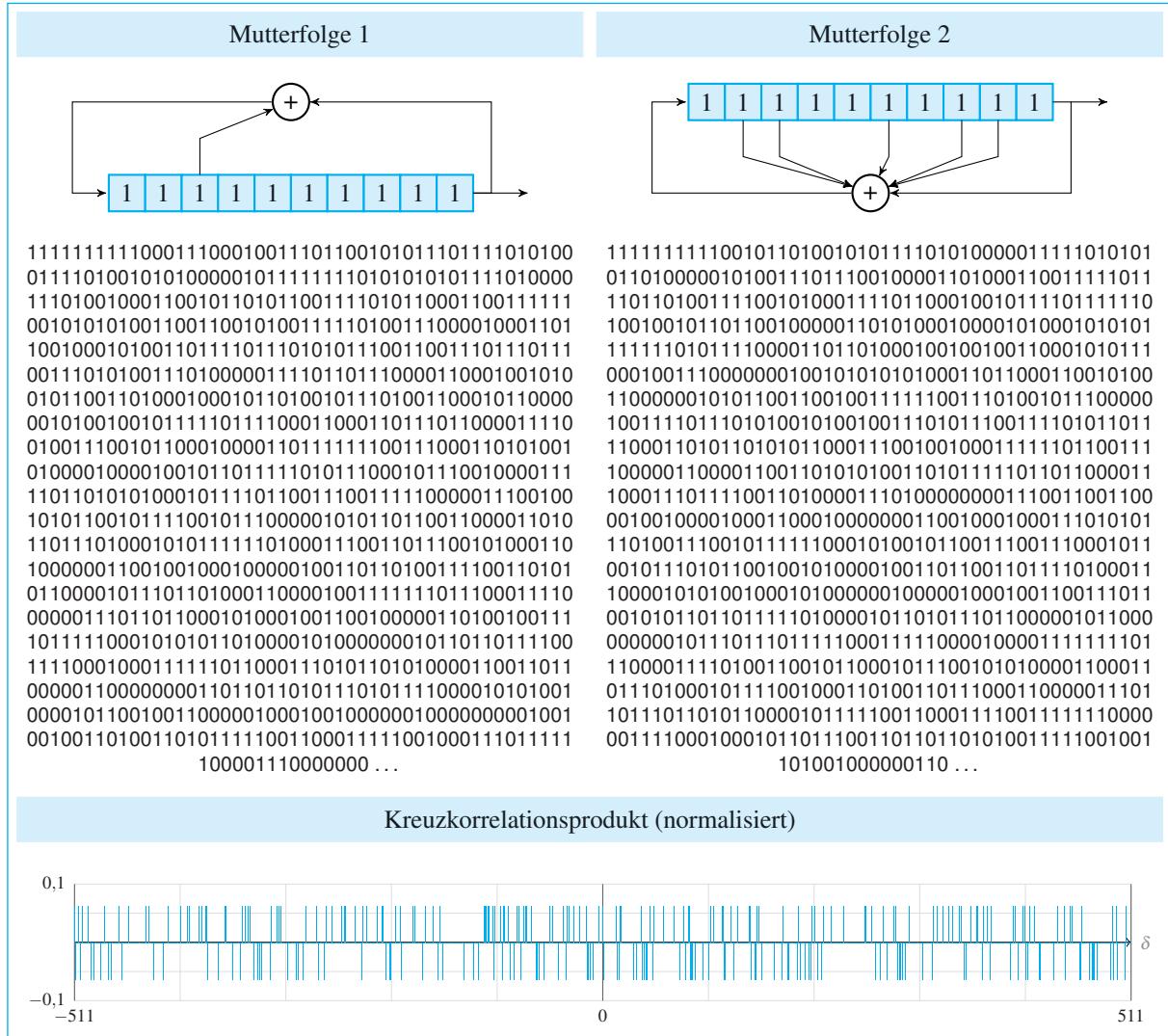
Ein Schieberegisterpaar, das in Golds Sinne zueinander passt, ist in Abbildung 8.47 gezeigt. Beide bestehen aus jeweils 10 Registerelementen und unterscheiden sich nur durch die Auswahl der Bits, die in die Berechnung der Modulo-2-Summe eingehen.

Starten wir in der initialen Konfiguration

$$(1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1),$$

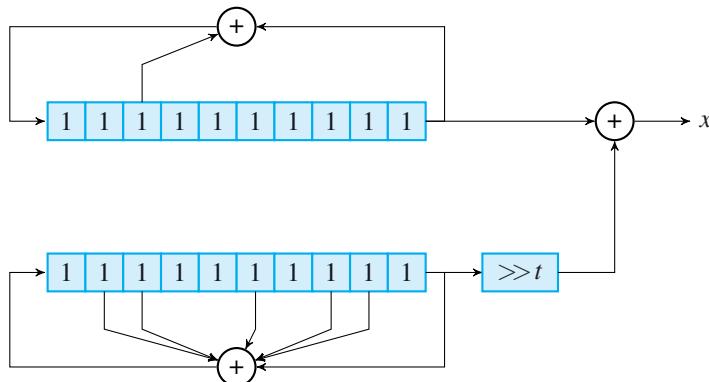
so produzieren die Schieberegister zwei Pseudozufallsfolgen, die wir in Anlehnung an [47] als *Mutterfolgen* bezeichnen.

Die normalisierte Kreuzkorrelation der beiden Mutterfolgen ist ebenfalls in Abbildung 8.47 wiedergegeben. Ein Blick auf den Werteverlauf zeigt, dass das Korrelationsprodukt tatsächlich nur drei verschiedene Werte annimmt: Es sind die Werte  $-65$ ,  $-1$  und  $63$ , die normalisiert die Werte  $-0,06354$ ,  $-0,00098$  und  $0,06159$  ergeben.



**Abb. 8.47:** Zwei Mutterfolgen, die für die Konstruktion von Gold-Codes geeignet sind.

Gold hatte aber noch etwas anderes entdeckt: Bilden wir die Modulo-2-Summe der beiden Mutterfolgen, so entsteht eine dritte Folge mit genau der gleichen Kreuzkorrelationseigenschaft. Und damit nicht genug: Die Eigenschaft bleibt selbst dann erhalten, wenn wir die zweite Mutterfolge vor der Summenbildung um  $t$  Positionen zyklisch verschieben.

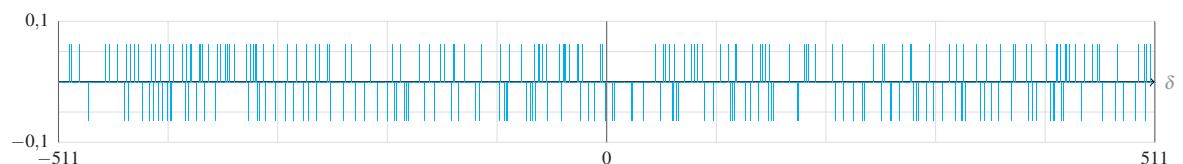


## Gold-Code mit $t = 256$

## Gold-Code mit $t = 512$

11110001101000010000000110100011001100111101100  
1111001010111101111110001100010101101010011011  
11010000110111001111100011001000100101101110010  
10110001010010000101011001011100010000010101101  
00011000110100001111011000110011011011111000010  
111100010101110010000011011000000111000000110110  
011110000101110011001100110001110000111010011110  
011111010001111100000110010011000101010101010011  
100010000101011001011011010100100101111000110001  
10110001111010101100000101111101000000001011100  
1000100101000110001001000111101010001111100101111  
11100110010010011011111111111100000111000110011  
000000011001001101110110110000100110001100111111  
11000110001000110000111111001011101110111101100  
011001111101010010011000011100100000101001100110  
01010100111111000011101011001010001111010100111  
001010110111110110110000100110100111010100010000  
00111000111001100110011001101100111110111001  
010101000111010001111001111000010011011000011100  
011010001011010000110111111101111010001100000101  
1111001111010111100001111010011000001111100101100

### Kreuzkorrelationsprodukt (normalisiert)



**Abb. 8.48:** Die Modulo-2-Summe der gegeneinander verschobenen Muttercodes lässt die Gold-Codes entstehen.

Mit diesem Wissen bereitet uns der Aufbau eines Gold-Code-Generators keine Verständnisschwierigkeiten mehr. Wie in Abbildung 8.48 zu sehen ist, besteht dieser aus zwei separaten Schieberegistern, die für die Produktion der Mutterfolgen zuständig sind. Aus beiden Mutterfolgen wird die Modulo-2-Summe gebildet, wobei die zweite Mutterfolge vor der Verknüpfung um  $t$  Positionen rotiert wird. Die überlagerte Pseudozufallsfolge ist der *Gold-Code*. Die Tatsache, dass uns der Zeitversatz  $t$  als freier Parameter zur Verfügung steht, führt dazu, dass wir aus zwei Muttercodes eine ganze Schar verschiedener Gold-Codes ableiten können.

### Global Positioning System (GPS)

Die Schieberegister, die wir in den Abbildungen 8.47 und 8.48 als Beispiele benutzt haben, waren nicht zufällig gewählt. Es sind genau jene, mit denen ein GPS-Satellit die CDMA-Chipsequenzen des zivil genutzten C/A-Signals erzeugt. Dass wir die Registerinhalte in unseren Beispielen mit dem Wert 1 beschrieben hatten, geschah ebenfalls im Vorgriff auf diesen Abschnitt. Es ist die Initialkonfiguration, mit denen ein GPS-Satellit die Generierung der Chipsequenz startet.

Damit unterschiedliche Chipsequenzen entstehen, wird die zweite Mutterfolge in jedem Satelliten um einen anderen Wert von  $t$  zeitversetzt. Aus Tabelle 8.49 können wir beispielsweise ablesen, dass der erste und der zweite GPS-Satellit die zweite Mutterfolge um 5 bzw. 6 Chips versetzen, bevor sie mit der ersten Mutterfolge zusammengeführt wird. Aber wie berechnen die Satelliten diesen Versatz? Muss die zweite Mutterfolge bis zu der benötigten Stelle zeitaufwendig vorberechnet und in einem Zwischenspeicher abgelegt werden?

Die Antwort ist Nein! In Wirklichkeit ist die Verschiebung der zweiten Mutterfolge denkbar einfach und basiert auf einer Eigenschaft von Schieberegisterfolgen, die wir im letzten Abschnitt bereits beschrieben haben. Dort haben wir herausgestellt, dass die Modulo-2-Summe einer Folge maximaler Länge mit einer verschobenen Variante von sich selbst wiederum eine verschobene Variante dieser Folge hervorbringt.

Abbildung 8.50 zeigt, wie die ersten beiden GPS-Satelliten dieses Prinzip ausnutzen, um den gewünschten Zeitversatz zu erzeugen. Die linke Schaltung berechnet zunächst die Modulo-2-Summe aus dem zweiten und dem sechsten Bit des zweiten Schieberegisters und verknüpft das Ergebnis anschließend mit dem ersten Muttercode. Die Berechnung der Modulo-2-Summe führt dazu, dass die Folge des zweiten Schieberegis-

ID	$t$	Register-summe	Chipsequenz	ID	$t$	Register-summe	Chipsequenz
1	5	$2 \oplus 6$	110010000011...	13	255	$6 \oplus 7$	111111010010...
2	6	$3 \oplus 7$	111001000011...	14	256	$7 \oplus 8$	111111101011...
3	7	$4 \oplus 8$	111100100011...	15	257	$8 \oplus 9$	111111110111...
4	8	$5 \oplus 9$	111110010011...	16	258	$9 \oplus 10$	111111111001...
5	17	$1 \oplus 9$	100101101100...	17	469	$1 \oplus 4$	100110111000...
6	18	$2 \oplus 10$	110010110100...	18	470	$2 \oplus 5$	110011011110...
7	139	$1 \oplus 8$	100101100111...	19	471	$3 \oplus 6$	111001101101...
8	140	$2 \oplus 9$	110010110001...	20	472	$4 \oplus 7$	111100110100...
9	141	$3 \oplus 10$	111001011010...	21	473	$5 \oplus 8$	111110011000...
10	251	$2 \oplus 3$	110100010010...	22	474	$6 \oplus 9$	111111001110...
11	252	$3 \oplus 4$	111010001011...	23	509	$1 \oplus 3$	100011001111...
12	254	$5 \oplus 6$	111110100001...	24	512	$4 \oplus 6$	111100011010...

Abb. 8.49: Chipsequenzen der GPS-Satellitenflotte

ters mit sich selbst überlagert wird und somit eine verschobene Mutterfolge entsteht. Die rechte Schaltung funktioniert nach dem gleichen Prinzip, bezieht in die Berechnung aber die Inhalte des dritten und des siebten Registers mit ein.

Die Chipsequenzen der anderen Satelliten lassen sich nach dem gleichen Prinzip erzeugen. Als Beispiel zeigt Abbildung 8.51 die Gold-Code-Generatoren der Satelliten 14 und 24. Der erste führt die Inhalte der Register 7 und 8 zusammen und der zweite die Inhalte der Register 4 und 6. Ein Blick auf die generierten Sequenzen zeigt, dass wir diese bereits kennen: Sie sind mit den beiden Gold-Folgen identisch, die wir auf Seite 616 in Abbildung 8.48 erzeugt haben.

Jeder GPS-Satellit strahlt zwei hochfrequente Trägerwellen auf die Erde ab. Die erste hat eine Frequenz von 1575,42 MHz und wird als *Link 1*, kurz L1, bezeichnet. *Link 2* hat eine Frequenz von 1227,6 MHz und wird mit L2 abgekürzt.

Ferner hat jeder GPS-Satellit mehrere Atomuhren an Bord, aus denen er die Sendefrequenzen präzise ableiten kann. Die Uhren arbeiten mit einer Grundfrequenz von 10,23 MHz und hängen folgendermaßen mit den L1- und L2-Frequenzen zusammen:

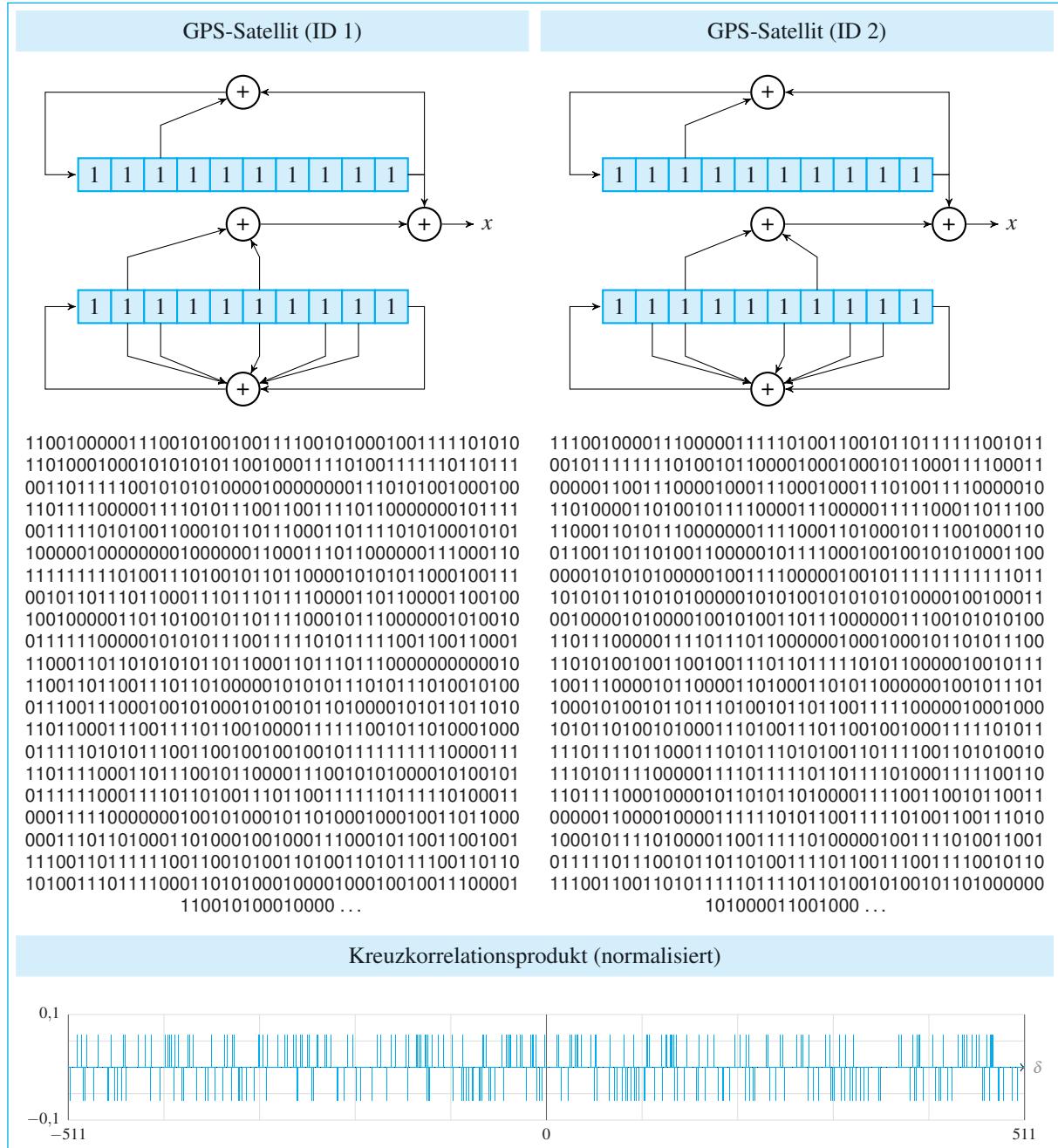
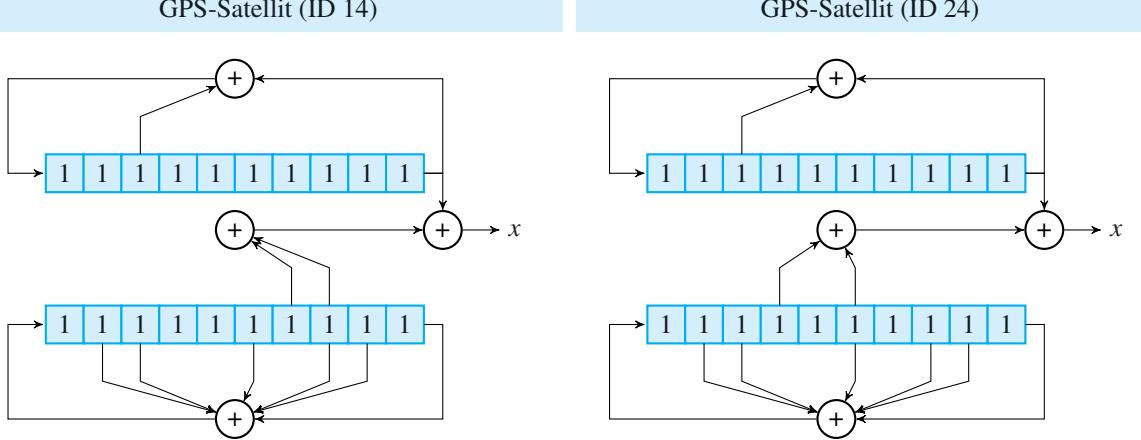


Abb. 8.50: Gold-Code-Generatoren der ersten beiden GPS-Satelliten



11111110101100000110111110011101001101000101010  
10011011101110110000011101010011111111100001  
010100110011011001001000011100001010010100110001  
111101110011010110111011011000101000001101110101  
100011110110111100110010011101010011010010010011  
111010001001101111110000111110001000011010111111  
01011110011101101000101000001110100010000110110  
1010001011101010001100110000110100010000110101111  
0110000100100110000010100001001110111011100001011  
0101011001000111001000100010010000110000110010101  
100101101111010111100101111100110111010110110101  
00011101001010000111100010000111101011111100011  
000011111110101011001001101001111110101100110011  
010011101100100111100110001011100010110010100111  
010111100001000100001111001010110010001000101011  
111100011010111111011111110000010100111010000111  
10100001011100101111101001000001100111101101000101  
11100000011001000100100000010100001010000011000011  
011010110001010100110000000100001010000000110000101  
0011110010000010110001101110110001000000001001000  
001101011010000111010010111000000011000010110100  
001101110101100 ...

11110001101000010000000110100011001100111101100  
1111001010111101111110001100010101101010011011  
110100001101110011111000110001000100101101110010  
10110001010010000101011001011100010000010101101  
00011000110100000111011000110011011011111000010  
11100010101100010000011011000000111000000110110  
011110000101110011001100110001110000111010011110  
01111001000111100000110010011000101010101010011  
100010000101011001011011010100100101111000110001  
10110001111010101100000101111101000000001011100  
100010010100011000100100011101010001111100101111  
11100110010010011011111111111100000111000110011  
00000001100100110111011011000010011000110011111  
11000110001000110000111111001011101110111101100  
011001111101010010011000011100100000101001100110  
011010100111111000011101011001010001111010100111  
001010110111110110110000100110100111010100010000  
001110001110011001100011001101100011111011110011  
010101000111010001111001111000010011011000011100  
011010001011010000011011111101111010001100000101  
1111001111010111100001111010011000001111100101100  
110001001010000 ...

### Kreuzkorrelationsprodukt (normalisiert)

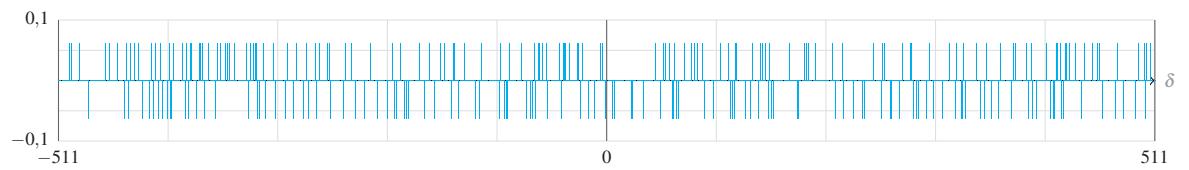


Abb. 8.51: Gold-Code-Generatoren der GPS-Satelliten 14 und 24

$$L1 = 1575,42 \text{ MHz} = 154 \cdot 10,23 \text{ MHz}$$

$$L2 = 1227,6 \text{ MHz} = 120 \cdot 10,23 \text{ MHz}$$

Auf den L1-Träger wird das C/A-Signal mithilfe der Phasenumtastung (*Phase Shift Keying*) moduliert, die wir aus Abschnitt 8.3.1 kennen. Der Modulator verwendet einen Winkelversatz von  $180^\circ$  und erzeugt pro Sekunde 1.023.000 potenzielle Phasenwechsel. Mit anderen Worten: Das C/A-Signal wird mit einer Frequenz von 1,023 MHz auf den L1-Träger moduliert. Wie wir weiter oben gesehen haben, besteht jede CDMA-Chipsequenz aus 1023 Chips, sodass pro Sekunde exakt 1000 Chipsequenzen übertragen werden. Um die Empfangsqualität zu erhöhen, wiederholen die GPS-Satelliten die Chipsequenz für jedes Bit 20-mal hintereinander. Das bedeutet, dass ein einzelnes Bit in Wirklichkeit durch 20.460 Chips dargestellt wird und für die Übertragung 20 msec benötigt. Pro Sekunde erreichen einen GPS-Empfänger damit lediglich 50 Bits an Nutzdaten.

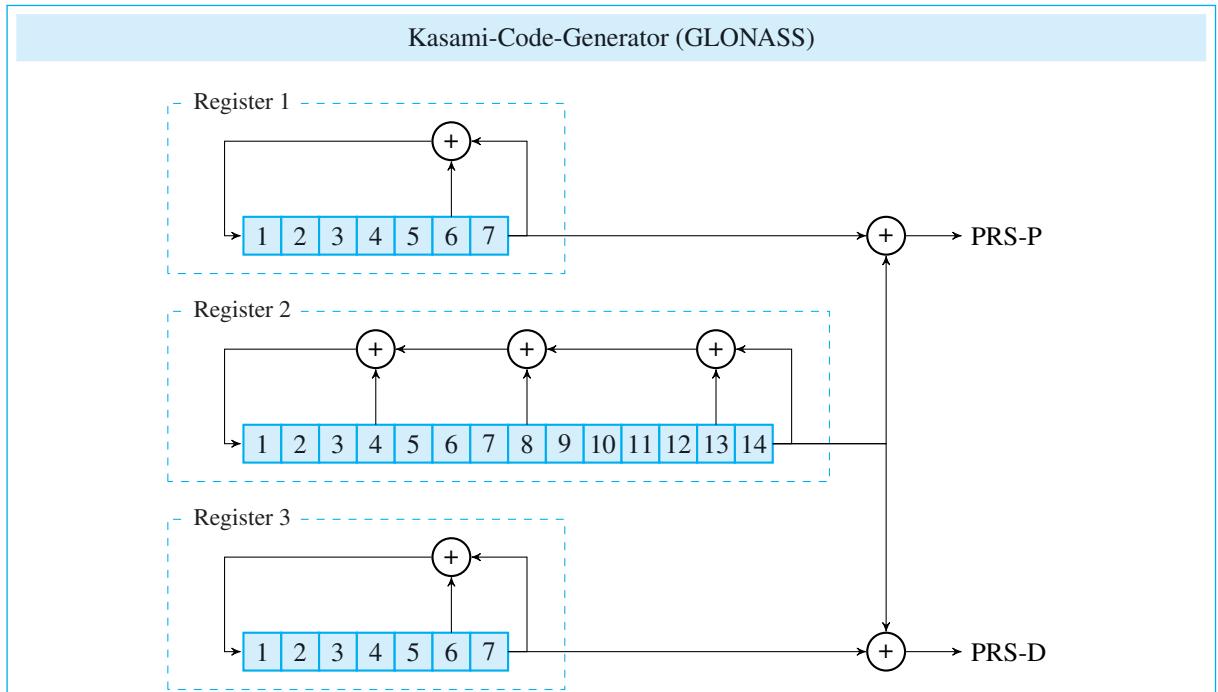
Das militärisch genutzte P(Y)-Signal wird ebenfalls auf den L1-Träger moduliert, mit einer um den Faktor 10 erhöhten Frequenz von 10,23 MHz. Genau wie das C/A-Signal wird auch das P(Y)-Signal mit einem Phasenumtaster erzeugt, der einen Winkelversatz von  $180^\circ$  verwendet. Um das C/A-Signal und das P(Y)-Signal voneinander trennen zu können, sind ihre Phasen gegenseitig um  $90^\circ$  verschoben. Nach der Zusammenführung der beiden Signale entsteht auf der L1-Frequenz somit ein quadraturmoduliertes Signal mit den vier möglichen Phasenwinkeln ( $0^\circ, 90^\circ, 180^\circ, 270^\circ$ ). Auf der L2-Frequenz wird im Normalbetrieb nur das P(Y)-Signal gesendet.

## 8.5.4 Kasami-Folgen

Zum Schluss dieses Kapitels wollen wir einen kurzen Blick auf die *Kasami-Folgen* werfen, die im Jahr 1966 von dem Japaner Tadao Kasami vorgeschlagen wurden. Von den Gold-Folgen unterscheiden sich die Kasami-Folgen unter anderem dadurch, dass die beiden Mutterfolgen durch Schieberegister unterschiedlicher Länge erzeugt werden.

Der in Abbildung 8.52 gezeigte Kasami-Code-Generator ist ein prominenter: Er wird von den GLONASS-Satelliten für die Generierung von CDMA-Chipsequenzen eingesetzt.

An dieser Stelle müssen wir kurz innehalten. Haben wir nicht weiter oben erklärt, dass GLONASS, anders als GPS, mithilfe des Frequenz-



**Abb. 8.52:** Das PRS-P-Signal und das PRS-D-Signal der GLONASS-Satelliten werden im Codemultiplexverfahren übertragen.

multiplexverfahrens für ein reibungsfreies Nebeneinander der verschiedenen Satelliten sorgt? Tatsächlich war dies nur die halbe Wahrheit. Seit dem Jahr 2011 senden alle neu gestarteten GLONASS-Satelliten zwei zusätzliche, CDMA-codierte Signals aus.

Konkret handelt es sich dabei um einen *Datenkanal* (PRS-D-Signal), der die Navigationsdaten enthält, und einen *Pilotkanal* (PRS-P-Signal). Der Pilotkanal enthält keine Navigationsdaten, ermöglicht den Empfängern aber eine schnellere Synchronisation.

Für die Erzeugung der Chipsequenzen, die für die Übertragung des Pilot- und des Datensignals benötigt werden, hält jeder GLONASS-Satellit drei Schieberegister vor, von denen eines doppelt verwendet wird. Anders als die GPS-Satelliten, die jeweils eine andere XOR-Logik für die Erzeugung ihrer Chipsequenzen verwenden, ist die Logik bei allen GLONASS-Satelliten die gleiche. Dass jeder Satellit eine unterschiedliche Folgen erzeugt, wird bei GLONASS über die Initialkonfiguration des ersten und des dritten Schieberegisters erreicht, die bei

ID	Startvektor Register 1	Startvektor Register 3	Chipsequenz (PRS-P-Code)	Chipsequenz (PRS-D-Code)
1	0000001	0100001	10011101101101010000000101101...	10011001101011010101000010001...
2	0000010	0100010	01011101001101100000101101010...	01011001001011100101101010110...
3	0000011	0100011	1101110000110000000111100101...	11011000001010000100111011001...
4	0000100	0100100	00111100011100011001101000110...	00111000011010011100101111010...
5	0000101	0100101	1011110101110111000111001001...	101110010110111110111110101...
6	0000110	0100110	01111101111101001000010001110...	011110011111011001101010110010...
7	0000111	0100111	1111110011110010100100000001...	111110001110101100000111101...
8	0001000	0101000	00001100110100100101001010000...	00001000110010100000001101100...
9	0001001	0101001	10001101110101000100011011111...	10001001110011000001011100011...
10	0001010	0101010	010011010101110100110011000...	0100100101001110001110100100...
11	0001011	0101011	11001100010100010101100010111...	11001000010010010000100101011...
12	0001100	0101100	00101100000100001101110110100...	00101000000010001000110001000...
13	0001101	0101101	10101101000101101100100111011...	101010010001110100110000111...
14	0001110	0101110	01101101100101110000111100...	01101001100011011001001000000...
15	0001111	0101111	1110110010010011101011110011...	11101000100010111000011001111...
16	0010000	0110000	00010100100000111011011011011...	0001000010011011110011100111...
17	0010001	0110001	10010101100001011010001010100...	10010001100111011111001101000...
18	0010010	0110010	01010101000001101010100010011...	01010001000111101111100101111...
19	0010011	0110011	1101010000000001011110011100...	11010000000110001110110100000...
20	0010100	0110100	00110100010000010011100111111...	00110000010110010110100000011...
21	0010101	0110101	101101010001110010110110000...	1011000101011110111110001100...
22	0010110	0110110	0111010111000100001001111011...	0111000110111000111011001011...
23	0010111	0110111	11110100110000100011001111000...	11110000110110100110001000100...
24	0011000	0111000	00000100111000101111000101001...	00000000111110101010000010101...

Tab. 8.3: Chipsequenzen der GLONASS-Satellitenflotte

jedem Satelliten eine andere ist. Mit welchen Inhalten das erste und das dritte Schieberegister gefüllt werden, ist in Tabelle 8.3 vermerkt.

Die Werte lassen sich sofort aus der Satellitennummer  $n$  ableiten. Das initiale Bitmuster des ersten Registers entspricht der Binärdarstellung von  $n$  und das initiale Bitmuster des dritten Registers der Binärdarstellung von  $n + 32$ . Die Initialkonfiguration des zweiten Schieberegisters ist für alle GLONASS-Satelliten gleich. Sie lautet:

00110100111000

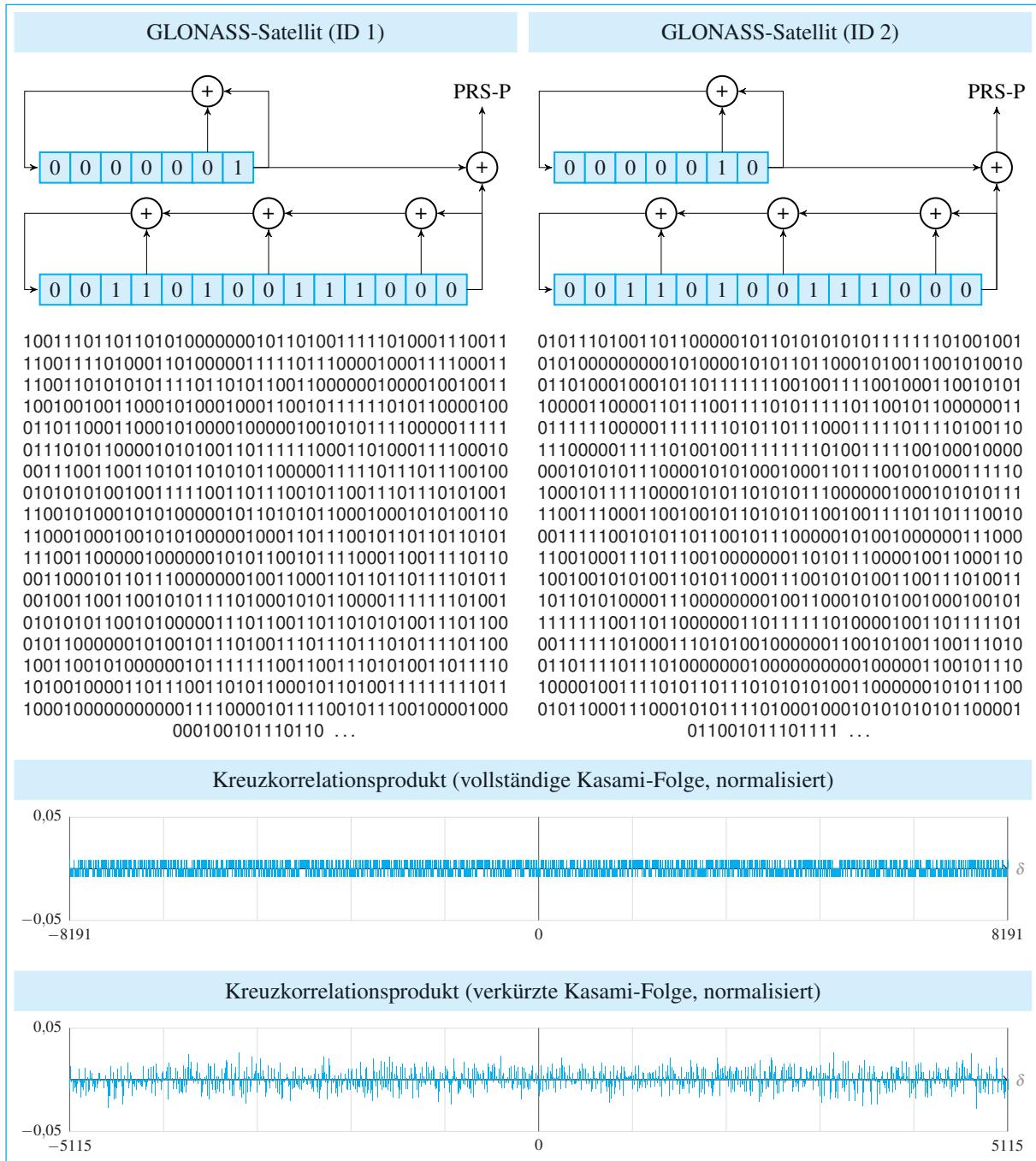


Abb. 8.53: Kasami-Code-Generatoren der ersten beiden GLONASS-Satelliten (PRS-P-Signal)

Die Chipsequenzen, die an den Ausgängen PRS-P und PRS-D ausgegeben werden, sind Kasami-Folgen mit der Periode

$$2^{14} - 1 = 16383$$

Von dieser werden jedoch nur die ersten 10.230 Bits verwendet; danach werden die Register in ihrer jeweiligen Initialkonfiguration neu gestartet. Das bedeutet, dass GLONASS im Vergleich zu GPS um den Faktor 10 verlängerte Chipsequenzen verwendet.

Abbildung 8.53 zeigt etwas ausführlicher, wie die Chipsequenzen der ersten beiden GLONASS-Satelliten berechnet werden und wie diese Sequenzen kreuzkorrelieren. Das Korrelationsprodukt ist zweimal dargestellt; einmal für die Kasami-Folgen in ihrer vollen Länge und einmal in der gekürzten Variante, wie sie von den GLONASS-Satelliten verwendet werden.

Das erste Diagramm zeigt, dass Kasami-Folgen eine Eigenschaft aufweisen, die wir von den Gold-Folgen her kennen: Das Korrelationsprodukt nimmt genau drei verschiedene Werte an. Das zweite Diagramm zeigt, dass diese Eigenschaft verloren geht, wenn wir die Folgen kürzen.

## 8.6 Übungsaufgaben

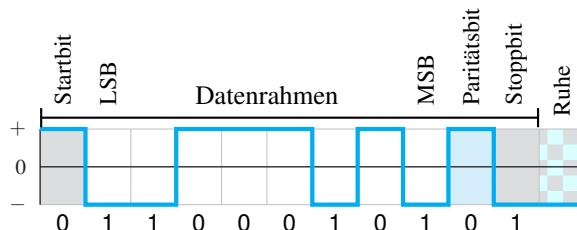
### Aufgabe 8.1

   
 Webcode  
8014

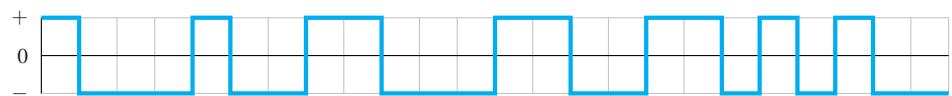
Werden Daten über eine RS232-Schnittstelle übertragen, so sind diese zumeist in einem logischen Datenrahmen eingebettet. Ein Beispiel ist der 9E1-Datenrahmen, der sich aus insgesamt 11 Bits zusammensetzt. Eingeleitet wird er durch ein Startbit (= 0), danach folgen

- 9 Bits, von denen die ersten 8 Datenbits sind und  9E1)
- das letzte ein Paritätsbit ist (gerade Parität, *Even parity*), sowie  9E1)
- 1 Stoppsbit (= 1).  9E1)

Die folgende Grafik klärt die Details (LSB – *Least Significant Bit*, MSB – *Most Significant Bit*):



- a) Rekonstruieren Sie aus den folgenden Signalverläufen die gesendeten Datenbits:



- b) Wiederholen Sie die Decodierung für die nachstehende Sequenz, dieses Mal stehen Ihnen allerdings keine Hilfslinien zur Verfügung. Was stellen Sie fest?

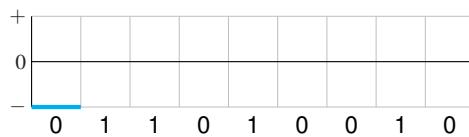


Vervollständigen Sie die folgenden Signalverläufe:

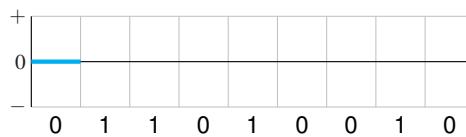
**Aufgabe 8.2**

**Webcode  
8058**

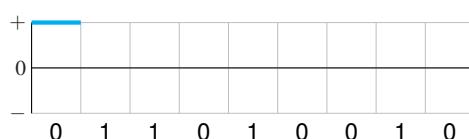
NRZ-Codierung



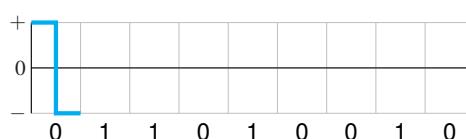
RZ-Codierung (unipolar)



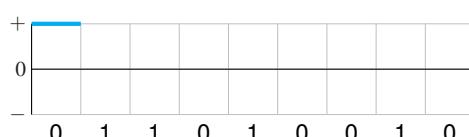
NRZI-M-Codierung



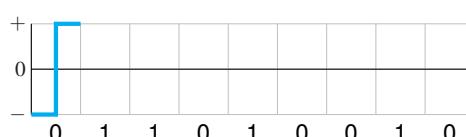
Manchester-Codierung



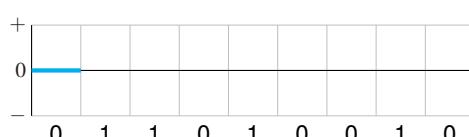
NRZI-S-Codierung



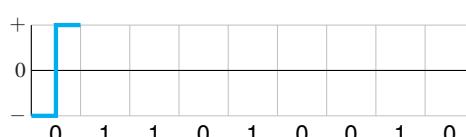
Manchester-II-Codierung



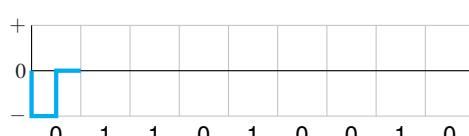
AMI-Codierung



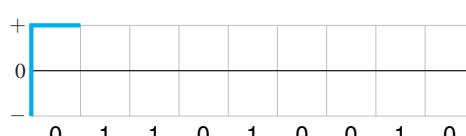
Differenzielle Manchester-Codierung



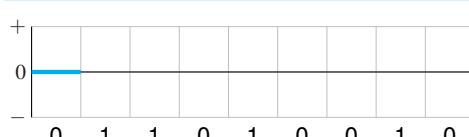
RZ-Codierung



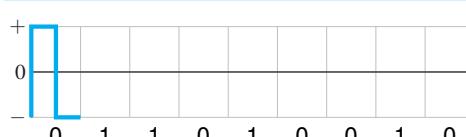
Biphase-M-Codierung



RZ-Codierung (bipolar)



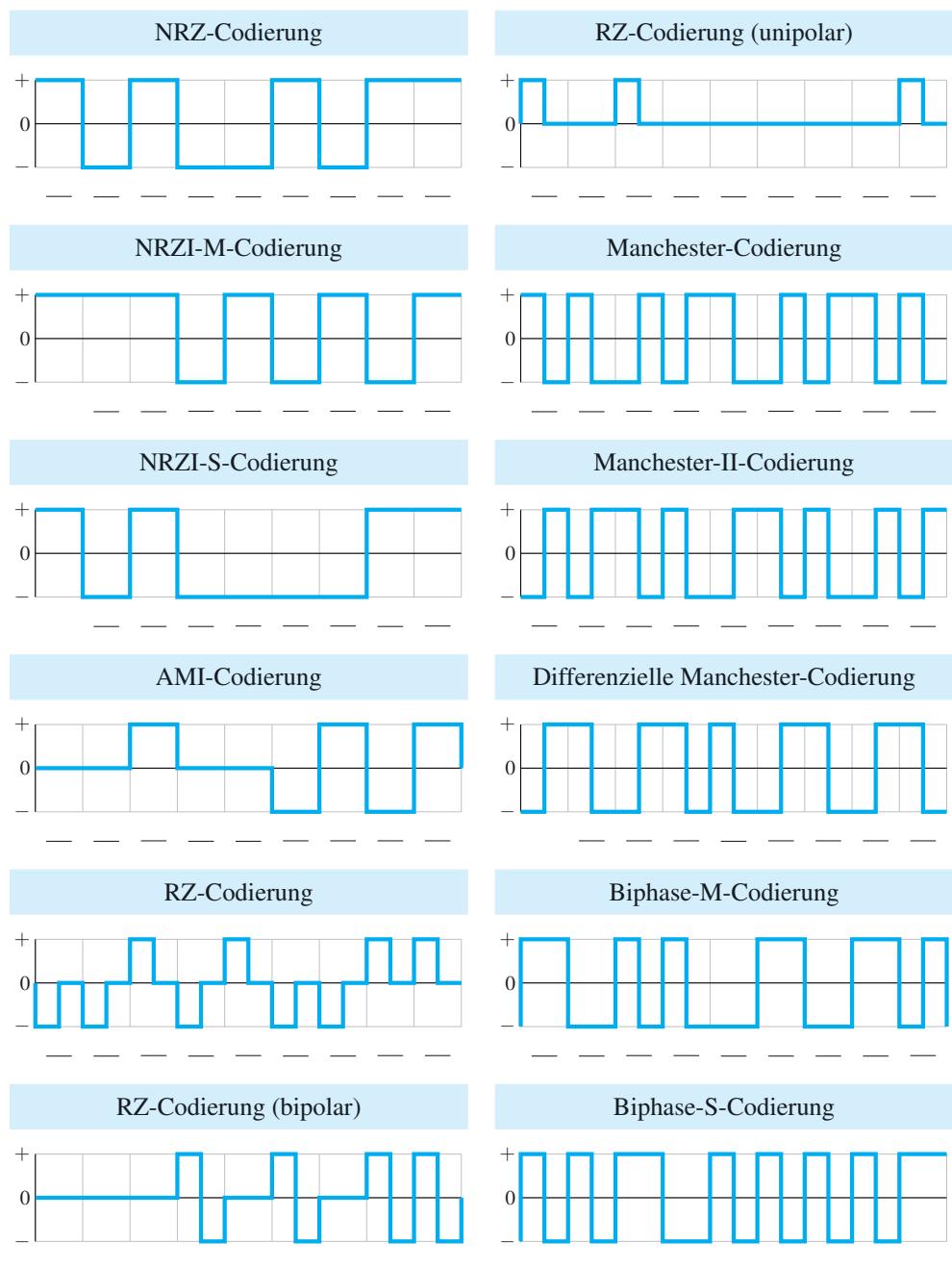
Biphase-S-Codierung



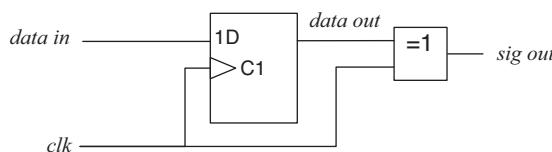
**Aufgabe 8.3**

 **Webcode**  
**8158**

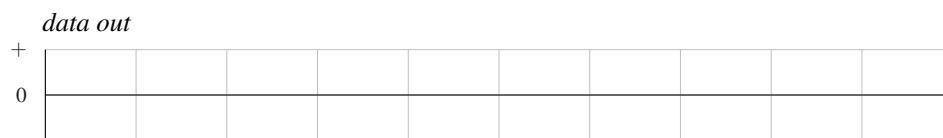
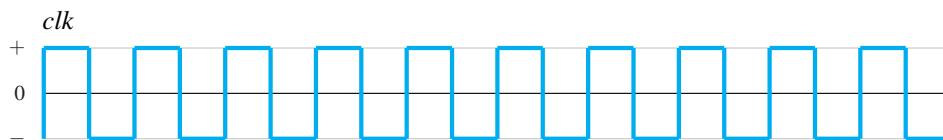
Decodieren Sie die folgenden Signalverläufe:



Gegeben sei die folgende Hardwareschaltung:

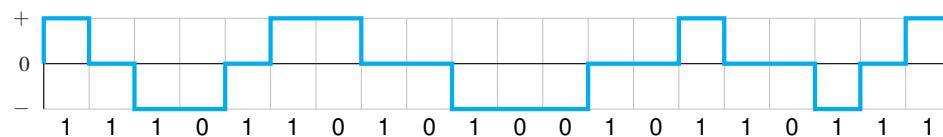


- a) Ergänzen Sie den Signalverlauf für *data out* und *sig out*, unter der Annahme, dass die Eingangssignale *clk* und *data in* folgendermaßen gegeben sind:



- b) Welcher Ihnen bekannte Leitungscode wird hier erzeugt?

Der folgende Signalverlauf wurde mit einem *MLT-3-Encoder* erzeugt:



- a) Welche Idee verfolgt die MLT-3-Codierung?

Aufgabe 8.4



Webcode  
8166

Aufgabe 8.5



Webcode  
8200

- b) Beschreiben Sie das Verhalten eines MLT-3-Encoders durch einen endlichen Automaten.

### Aufgabe 8.6

 **Webcode  
8212**

Wir nehmen an, ein Sender und ein Empfänger seien so mit einem Kupferkabel verbunden, dass pro Sekunde bis zu 800.000 Spannungswechsel sicher erzeugt und detektiert werden können. Welche Datenraten ergeben sich, wenn

- a) die NRZ-Codierung
- c) die Manchester-Codierung
- b) die NRZI-Codierung
- d) eine 4B/5B-Codierung

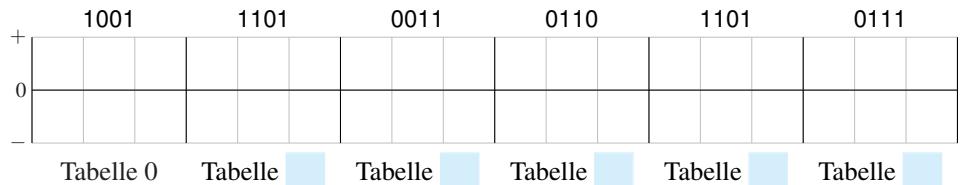
verwendet wird?

### Aufgabe 8.7

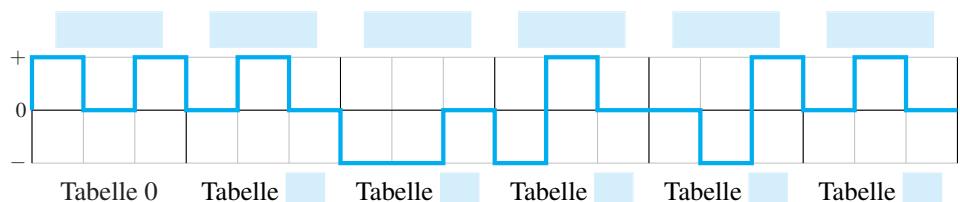
 **Webcode  
8321**

In Abschnitt 8.2.2.1 haben Sie die MMS43-Codierung kennengelernt, die jeweils 4 Bits auf 3 Ternärpegel abbildet.

- a) Codieren Sie die folgende Bitsequenz mithilfe der MMS43-Codierung:



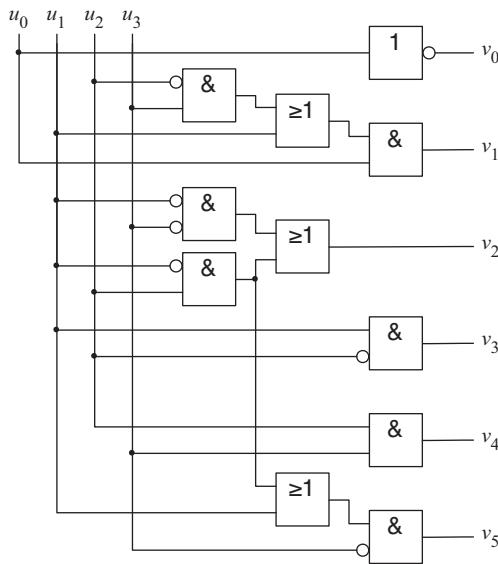
- b) Decodieren Sie die folgende MMS43-codierte Sequenz:



Die nachstehend abgebildete Hardwareschaltung implementiert einen 4B/6B-Leitungscodierer, der die Eingabebits  $u_0, \dots, u_3$  auf das Codewort  $v_0, \dots, v_5$  abbildet:

**Aufgabe 8.8**


**Webcode**  
**8356**



- a) Stellen Sie für jede der 6 Ausgangsleitungen eine boolesche Funktion auf:

$$v_0 = \overline{u_0}$$

$$v_2 = \text{[blue box]}$$

$$v_4 = \text{[blue box]}$$

$$v_1 = u_0(\overline{u_2}u_3 \vee u_1)$$

$$v_3 = \text{[blue box]}$$

$$v_5 = \text{[blue box]}$$

- b) Benutzen Sie die Ergebnisse aus Teil a), um die folgende Tabelle auszufüllen:

$u_0u_1u_2u_3$	$v_0v_1v_2v_3v_4v_5$	$u_0u_1u_2u_3$	$v_0v_1v_2v_3v_4v_5$
0000	101000	1000	
0001	100000	1001	
0010		1010	
0011		1011	
0100		1100	
0101		1101	
0110		1110	
0111		1111	

- c) Implementiert die Schaltung die (1,7)-RLL-Codierung aus Abbildung 8.16?

**Aufgabe 8.9**

**Webcode**  
**8465**

Die folgende Bitsequenz stammt aus einem DCF77-Signal. Wie Sie bereits wissen, codieren in einem Datenpaket die Bits 21 bis 58 die aktuelle Uhrzeit und das Datum:

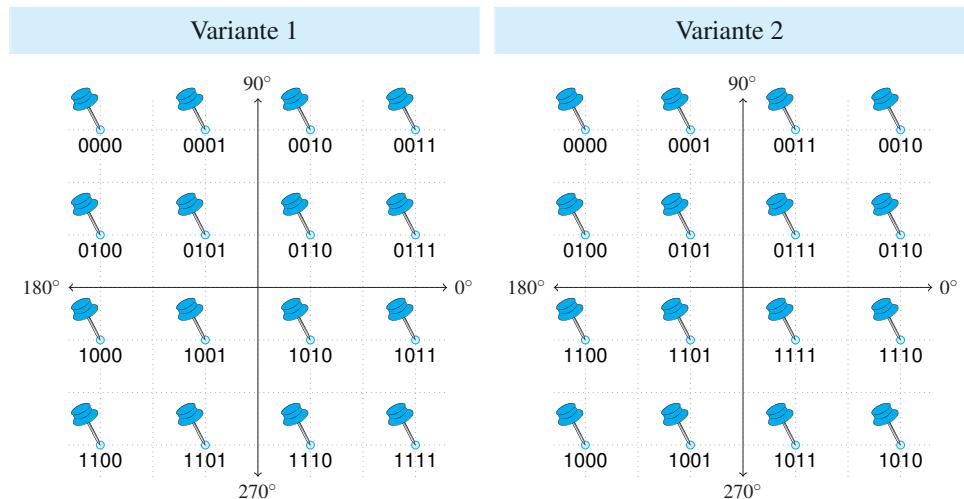
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58					
1	0	0	0	0	0	0	1	1	1	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- Wann wurde dieses Datenpaket gesendet?
- Wie lange muss eine Funkuhr mindestens eingeschaltet sein, bis sie die aktuelle Uhrzeit anzeigen kann?
- Wie lange muss eine Funkuhr höchstens laufen, bis die aktuelle Uhrzeit angezeigt werden kann? Gehen Sie bei der Berechnung davon aus, dass die Uhr das DCF77-Signal fehlerfrei empfängt.

**Aufgabe 8.10**

**Webcode**  
**8472**

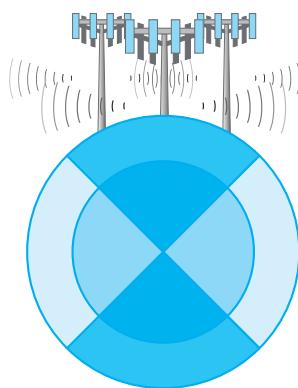
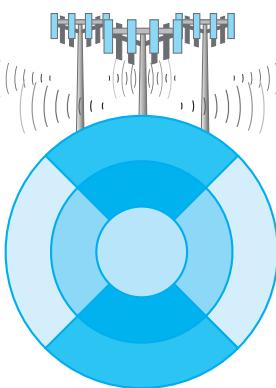
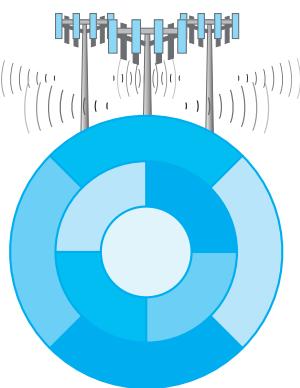
Betrachten Sie die nachstehend abgebildeten QAM-16-Signalraumkonstellationen zusammen mit der jeweiligen Zuordnung zu den codierten Bitsequenzen.



- Welche Codierung ist für die Praxis besser geeignet? Begründen Sie Ihre Antwort.
- An welche Codierung erinnert Sie die rechte Codewortverteilung?

Die folgenden drei Abbildungen zeigen eine fiktive Aufteilung eines Sendegebiets in verschiedene Funkzellen.

**Aufgabe 8.11**

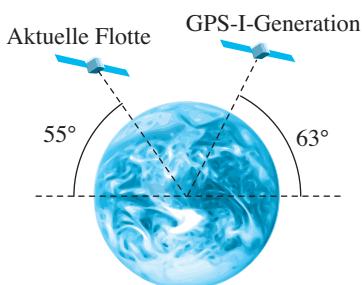
**Webcode**
**8491**
**1. Aufteilung**

**2. Aufteilung**

**3. Aufteilung**


Den Funkzellen wurden verschiedene Frequenzbänder zugeteilt, die anhand ihrer Farbe unterschieden werden können. Wie sich leicht überprüfen lässt, belegen angrenzende Funkzellen jeweils andere Bänder, sodass es an den Zellengrenzen zu keinen Störungen kommen kann. Leider ist Zuordnung nicht optimal.

- Versuchen Sie eine andere Zuordnung zu finden, die eine geringere Zahl an Frequenzbändern erfordert.
- Geben Sie für jedes Diagramm an, wie viele Frequenzbänder mindestens benötigt werden.
- Lässt sich eine Funkzellentopologie konstruieren, die noch mehr Frequenzbänder benötigt, als es in den Beispieldiagrammen der Fall war?

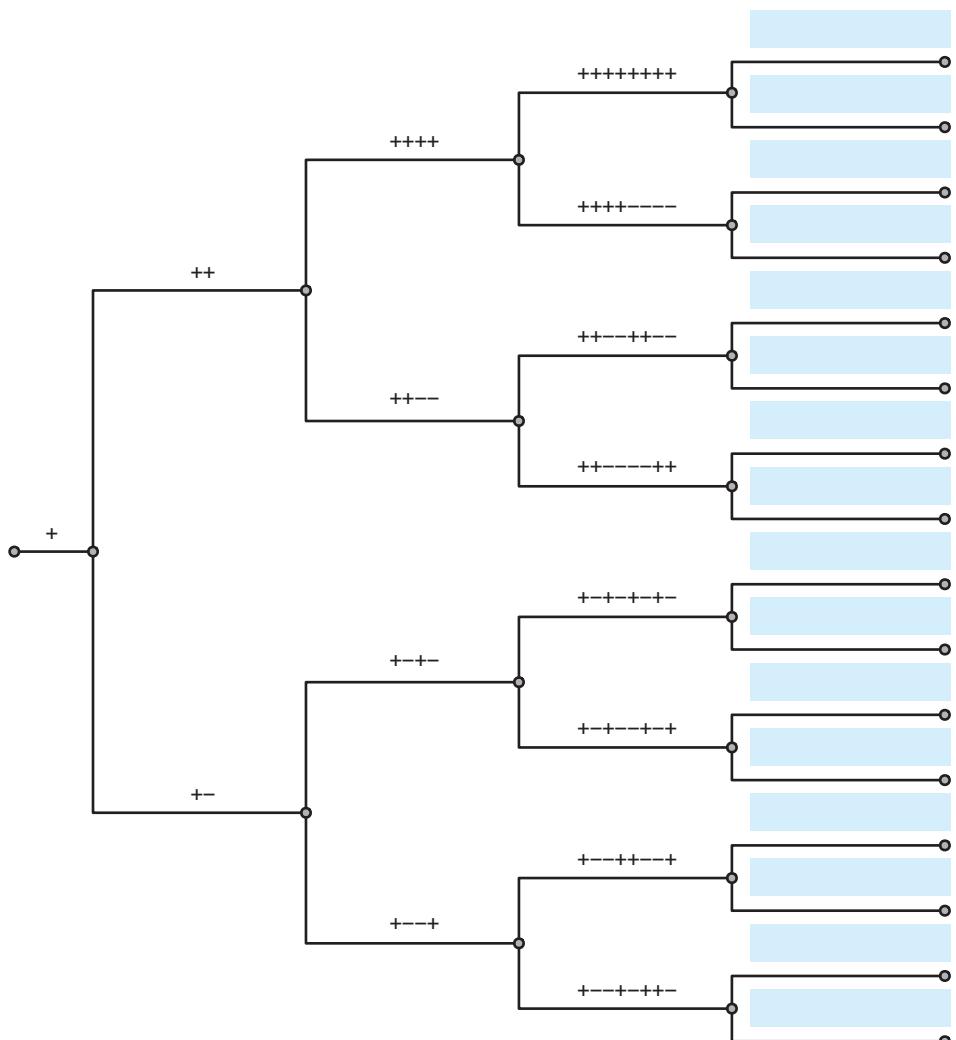
Auf Seite 596 wurde beschrieben, dass die Orbitalebahnungen der GPS-Satelliten  $55^\circ$  gegenüber der Äquatorialebene geneigt sind. Dies war nicht immer so! Die Satelliten der ersten GPS-Generation wiesen eine Inklination von  $63^\circ$  auf. Von den beiden Polkappen aus betrachtet, stehen die Satelliten heute tiefer über dem Horizont, was die Positionsbestimmung in sehr nördlichen und sehr südlichen Regionen erschwert. Warum wurde die Ausrichtung der Bahnebene Ihrer Meinung nach verändert?

**Aufgabe 8.12**

**Webcode**
**8541**


**Aufgabe 8.13****Webcode  
8578**

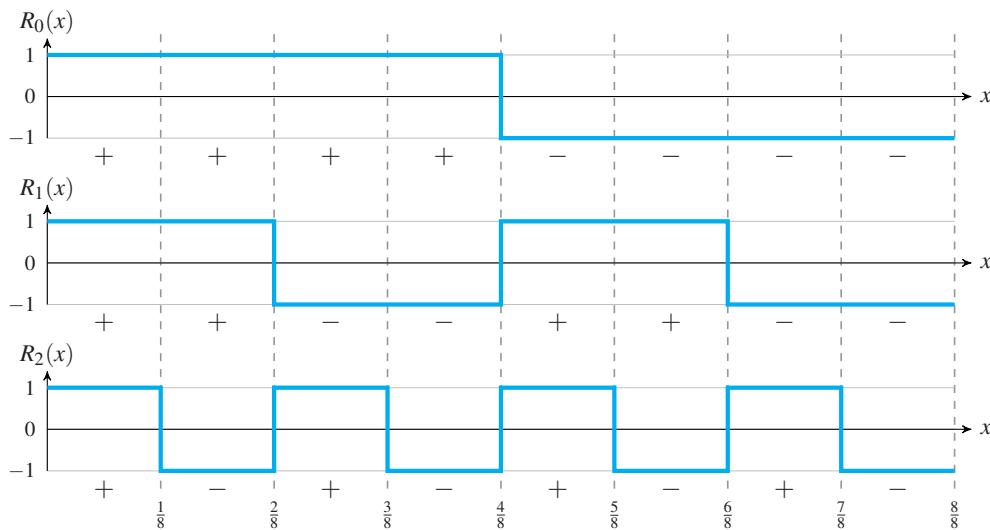
Auf Seite 603 ist in Abbildung 8.37 der Anfang des OVSF-Codebaums zu sehen. Erweitern Sie den Baum um eine zusätzliche Blätterschicht:



Hinter den *Rademacher-Funktionen*, benannt nach dem deutschen Mathematiker Hans Rademacher, verbergen sich Funktionen der Form

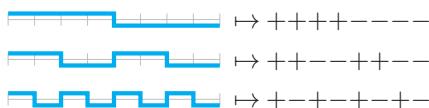
$$R_n(x) : [0; 1] \rightarrow \{-1, 1\}$$

Die Funktionen sind sehr einfach aufgebaut. Sie beschreiben gleichförmige Rechtecksignale:



Ist  $n$  eine natürliche Zahl, so können wir aus den Rademacher-Funktionen  $R_0(x)$  bis  $R_{n-1}(x)$  einen Code ableiten, der aus  $n$  Codewörtern der Länge  $2^n$  besteht. Hierfür teilen wir das Intervall  $[0; 1]$  in  $2^n$  Partitionen auf und interpretieren die Funktionswerte, die in jeder Partition konstant sein müssen, als die Symbole eines Codeworts.

Für  $n = 3$  können wir die Rademacher-Codewörter direkt aus den oben gezeichneten Graphen ableiten. Wir erhalten das folgende Ergebnis:



Für die Darstellung der Codewörter benutzen wir die Kürzel  $,+^{'}$  und  $,-^{'}$ , die stellvertretend für die Zahlenwerte 1 bzw.  $-1$  stehen.

- Erzeugen Sie den Rademacher-Code für  $n = 4$ .
- Sind die Rademacher-Codewörter paarweise orthogonal zueinander?

### Aufgabe 8.14

Webcode  
8618

**Aufgabe 8.15**
**Webcode  
8625**

In der vorherigen Teilaufgabe haben Sie die Rademacher-Funktionen kennengelernt und gesehen, wie wir sie für die Erzeugung von Codewörtern verwenden können. Wenn Sie die Aufgabe richtig bearbeitet haben, wissen Sie bereits, dass alle Rademacher-Codewörter paarweise orthogonal zueinander sind. In dieser Aufgabe wollen wir uns mit dem Umstand auseinandersetzen, dass Rademacher-Codes nicht *vollständig* sind. Damit ist gemeint, dass weitere Codewörter der gleichen Länge existieren, die paarweise orthogonal zu den übrigen sind.

Eine Möglichkeit, die Rademacher-Codes zu vervollständigen, besteht darin, zwei oder mehrere Rademacher-Codewörter komponentenweise miteinander zu multiplizieren. Das Ergebnis ist dann ebenfalls orthogonal zu den bisherigen Codewörtern und kann als neues Codewort hinzugenommen werden.

a) Setzen Sie das Gesagte in die Tat um. Ergänzen Sie hierfür die nachstehende Tabelle:

Rademacher-Produkt	Walsh-Codewort
0	++++++-----++++++
1	R <sub>0</sub>
2	R <sub>1</sub>
3	R <sub>1</sub> · R <sub>0</sub>
4	R <sub>2</sub>
5	R <sub>2</sub> · R <sub>0</sub>
6	R <sub>2</sub> · R <sub>1</sub>
7	R <sub>2</sub> · R <sub>1</sub> · R <sub>0</sub>
8	R <sub>3</sub>
9	R <sub>3</sub> · R <sub>0</sub>
10	R <sub>3</sub> · R <sub>1</sub>
11	R <sub>3</sub> · R <sub>1</sub> · R <sub>0</sub>
12	R <sub>3</sub> · R <sub>2</sub>
13	R <sub>3</sub> · R <sub>2</sub> · R <sub>0</sub>
14	R <sub>3</sub> · R <sub>2</sub> · R <sub>1</sub>
15	R <sub>3</sub> · R <sub>2</sub> · R <sub>1</sub> · R <sub>0</sub>

- b) Der Code, den Sie gerade erzeugt haben, heißt *Walsh-Code*, benannt nach dem amerikanischen Mathematiker Joseph Leonard Walsh. Mit welchem Ihnen bereits bekannten Code, ist der Walsh-Code identisch?
- c) Die Reihenfolge, in der die Rademacher-Codewörter in der Tabelle erzeugt wurden, war nicht zufällig gewählt. Sie geht auf den englischen Mathematiker Raymond Paley zurück und folgt einem elementaren Schema. Um welches Schema handelt es sich?

Die Matrizen  $C_1, C_2, C_4, \dots$  sind durch die folgende Bildungsvorschrift gegeben:

$$C_1 := (1), \quad C_2 := \begin{pmatrix} C_1(1) & C_1(1) \\ C_1(1) & -C_1(1) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad \dots, \quad C_{2n} := \begin{pmatrix} C_n(1) & C_n(1) \\ C_n(1) & -C_n(1) \\ C_n(2) & C_n(2) \\ C_n(2) & -C_n(2) \\ \vdots \\ C_n(n) & C_n(n) \\ C_n(n) & -C_n(n) \end{pmatrix}$$

Der Ausdruck  $C_j(k)$  steht für die  $k$ -te Zeile der Matrix  $C_j$ .

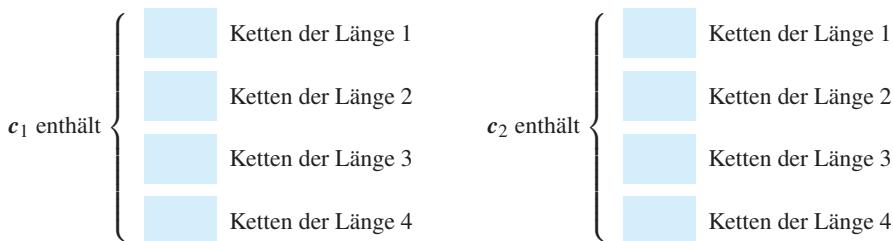
- Berechnen Sie die Matrizen  $C_4$  und  $C_8$ .
- Welche Bedeutung besitzen die Matrizen, wenn ihre Zeilen als Codewörter aufgefasst werden?

In Abschnitt 8.5.2 haben wir gesehen, dass die beiden Pseudozufallsfolgen

$$\begin{aligned} c_1 &= 100010011010111 \\ c_2 &= 100011110101100 \end{aligned}$$

sogenannte *Folgen maximaler Länge* sind. In dieser Aufgabe wollen wir uns an eine wichtige Eigenschaft solcher Folgen herantasten, die mit der Auftrittswahrscheinlichkeit von Einser- und Nullerketten zu tun hat.

- Analysieren Sie, wie viele Einser- und Nullerketten in  $c_1$  und  $c_2$  vorkommen. Eine Nullerkette der Länge  $n$  besteht aus  $n$  Nullen, die von zwei Einsen umgeben ist, und das Gleiche gilt für Einserketten. Zählen Sie an den Grenzen so, als würde die Sequenz mit sich selbst fortgesetzt werden. Das bedeutet, dass die Sequenz 1001 jeweils eine Einser- und eine Nullerkette der Länge 2 enthält.



- Können Sie an den Beispielen eine allgemeine Regel erkennen?

### Aufgabe 8.16



**Webcode**  
**8732**

### Aufgabe 8.17



**Webcode**  
**8785**

**Aufgabe 8.18****Webcode  
8793**

In Abschnitt 8.5.3 haben wir gezeigt, wie zwei Mutterfolgen zu jenen Folgen verschmolzen werden, die nach ihrem Entdecker als Gold-Folgen bezeichnet werden.

- Die Mutterfolgen sind Pseudozufallsfolgen maximaler Länge. Zeigen Sie, dass zwei solche Folgen nicht orthogonal zueinander sein können.
- Folgt aus Teil a), dass zwei Gold-Folgen ebenfalls nicht orthogonal zueinander sind?
- Für die Generierung der Chipsequenzen verzögert ein GPS-Satellit die zweite Mutterfolge, indem er die Inhalte mehrerer Register vor der Verschmelzung mit der ersten Mutterfolge XOR-verknüpft. Kann auf diese Weise jeder gewünschte Phasenversatz erzeugt werden?

**Aufgabe 8.19****Webcode  
8810**

Ein GPS-Empfänger wird für 1 ms in Betrieb genommen und empfängt in diesem Zeitraum die folgenden Chips:

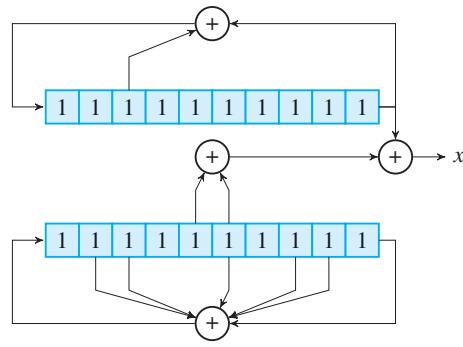
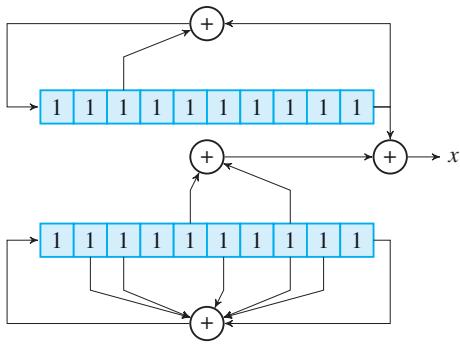
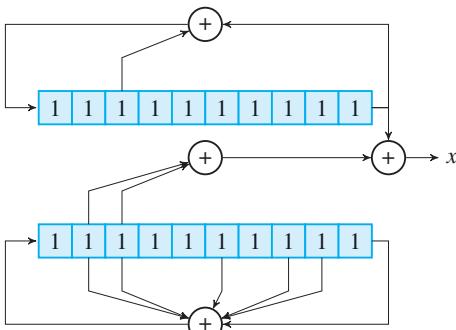
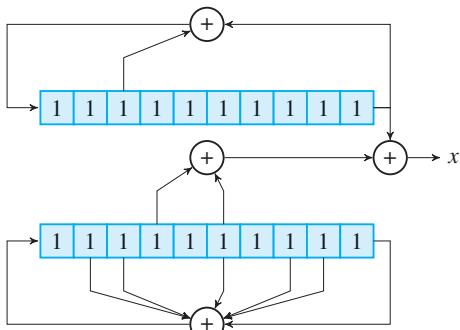
0	-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
64	+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
128	+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
192	+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
256	+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
320	+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
384	+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
448	+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
512	+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
576	+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
640	+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
704	+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
768	+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
832	+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
896	+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
960	+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

- Finden Sie heraus, von welchem Satelliten die Sequenz gesendet wurde.

Hinweis: Da wir nicht wissen können, ob der erste empfangene Chip den Beginn oder vielleicht die Mitte einer Chipsequenz markiert, ist es fast hoffnungslos, die Berechnung per Hand auszuführen. Schreiben Sie stattdessen ein Computerprogramm, mit dem das Korrelationsprodukt zwischen der empfangenen Sequenz und den Chipsequenzen der GPS-Satelliten automatisch ausgerechnet werden kann.

- An welcher Position beginnt eine neue Chipsequenz?

Geben Sie für jeden der abgebildeten Gold-Code-Generatoren an, von welchem GPS-Satelliten er für die Erzeugung der CDMA-Chipsequenzen genutzt wird:

Satelliten-ID = Satelliten-ID = Satelliten-ID = Satelliten-ID = 

In Abschnitt 8.5.3 wurde beschrieben, dass ein GPS-Satellit das C/A-Signal mit einer Frequenz von 1,023 MHz auf den L1-Träger moduliert und somit in einer Sekunde 1000 Chipsequenzen überträgt.

**Aufgabe 8.20**
**Webcode**  
8851

- Wir wollen uns eine räumliche Vorstellung von dieser elektromagnetischen Welle verschaffen. Geben hierzu an, wie viele Meter der Welle einer Chipsequenz und wie viele Meter einem einzigen Chip entsprechen.
- Wenn wir schon bei der räumlichen Ausdehnung der elektromagnetischen Welle sind: Wie viele Chipsequenzen passen, räumlich gesehen, auf die Strecke zwischen einem GPS-Satelliten und einem GPS-Empfänger? Wie viele Bits „befinden“ sich auf dieser Strecke?

**Aufgabe 8.21**
**Webcode**  
8878

- c) Auf Seite 618 wurde erwähnt, dass ein GPS-Satellit mehrere Atomuhren an Bord mitführt, die mit einer Grundfrequenz von 10,23 MHz arbeiten. Betrachtet man die verbauten Taktgeber genauer, so fällt auf, dass diese gar nicht mit den spezifizierten 10,23 MHz arbeiten, sondern mit einer geringfügig verringerten Frequenz von

10,229999995433 MHz



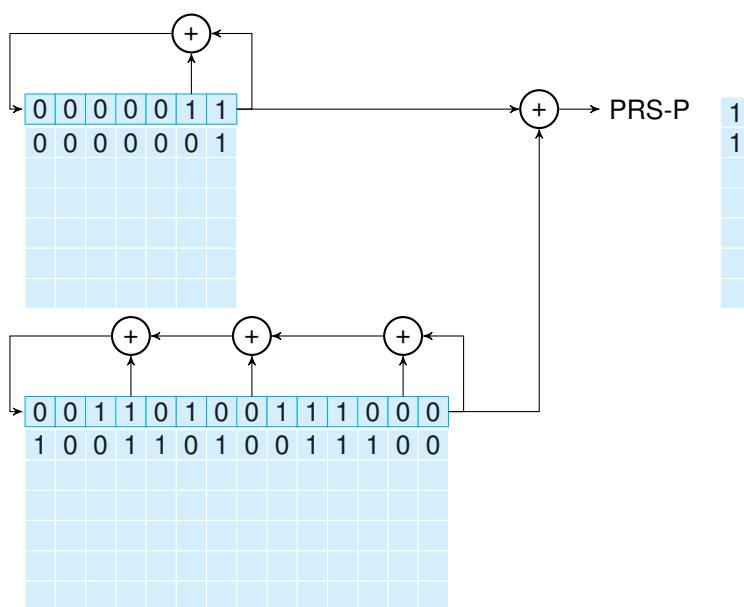
Was ist der Grund für diese Abweichung?

- d) Werden die GPS-Signale durch den Doppler-Effekt beeinflusst?

### Aufgabe 8.22

Webcode  
8923

Nachstehend ist der Kasami-Code-Generator abgedruckt, mit dem die GLONASS-Satelliten die CDMA-Chipsequenzen für das PRS-P-Signal erzeugen.



- a) Zu welchem Satelliten gehört die eingetragene Startkonfiguration?  
 b) Simulieren Sie den Code-Generator für weitere 5 Zeitschritte, indem Sie die leer gelassenen Felder ergänzen.

## Epilog

Von den Leuchtfuern in Argos, wo unsere Reise begann, zu den satellitengestützten Navigationssystemen, mit denen sie an dieser Stelle zu Ende geht, war es ein langer Weg. Es ist unbestritten: Ohne die ausgefeilte Ingenieurskunst, die sich die Menschheit über hunderte von Jahren angeeignet hat, gäbe es keine der raffinierten Apparaturen, die uns heute in nahezu sämtlichen Lebenslagen unterstützend zur Seite stehen. Dass sich die moderne Kommunikationstechnik einem komplexen mathematischen Apparat bedient und kaum ohne die grundlegenden Erkenntnisse aus der Informations- und Codierungstheorie existieren könnte, wird dabei häufig übersehen. In den hinter uns liegenden Kapiteln habe ich den Versuch unternommen, den Blick für diese mathematische Seite der Kommunikationstechnik zu schärfen, ohne den Bezug zur Praxis zu verlieren. Ob es mir gelungen ist, müssen Sie entscheiden, liebe Leser!

# Literaturverzeichnis

---

- [1] Aischylos: *Agamemnon*. www.gutenberg.org: Projekt Gutenberg, 458 v. Chr.
- [2] Allen, J.; Becker, J.: *The Unicode Standard, Version 5.0*. Amsterdam: Addison-Wesley Longman, 2006
- [3] Aschoff, V.: *Nachrichtentechnische Entwicklungen in der ersten Hälfte des 19. Jahrhunderts*. Berlin, Heidelberg, New York: Springer-Verlag, 1987
- [4] Aschoff, V.: *Beiträge zur Geschichte der Nachrichtentechnik von ihren Anfängen bis zum Ende des 18. Jahrhunderts*. 2. Auflage. Berlin, Heidelberg, New York: Springer-Verlag, 1989
- [5] Attneave, F.: *Informationstheorie in der Psychologie. Grundbegriffe, Techniken, Ergebnisse*. Huber-Verlag, 1965
- [6] Barbier, C.: *Typographie privée, de poche et d'ambulance*. Paris: Bachelier, Libraire, 1832  
(Émancipation intellectuelle d'expédition française)
- [7] Bauer, F. L.: *Historische Notizen zur Informatik*. Berlin, Heidelberg, New York: Springer-Verlag, 2009
- [8] Beauchamp, K.: *History of Telegraphy*. London: Institution of Engineering and Technology, 2001
- [9] Bell Telephone Laboratories (Hrsg.): *Press Release*. Juli. 463 West Street, New York 14, Chelsea: Bell Telephone Laboratories, 1948
- [10] Berrou, C.; Glavieux, A.; Thitimajshima, P.: Near Shannon limit error-correcting coding and decoding. In: *Proceedings of IEEE International Communications Conference*, 1993, S. 1064–1070
- [11] Bertram, Bill: *Commodore 64C system with 1541-II floppy drive and 1084S RGB monitor*.  
<http://creativecommons.org/licenses/by-sa/2.5/>. Version: 2005. – Creative Commons License 2.5, Typ: BY-SA
- [12] Bosch, K.: *Brückenkurs Mathematik: Eine Einführung mit Beispielen und Übungsaufgaben*. München: Oldenbourg Wissenschaftsverlag, 2007
- [13] Burrows, M.: Foreword by Mike Burrows. In: *Theoretical Computer Science* (2007), Nr. 387, S. 197–199
- [14] Burrows, M.; Wheeler, D. J.: A block-sorting lossless data compression algorithm / Digital Equipment Corporation. 1994 (124). – Forschungsbericht. – Technical Report
- [15] Cantor, G.: über eine Eigenschaft des Inbegriffes aller reellen algebraischen Zahlen. In: *Crelles Journal für Mathematik* 77 (1874), S. 258–262
- [16] Cayley, Arthur: On the theory of groups as depending on the symbolic equation  $\theta^n = 1$ . In: *Philosophical Magazine* 7 (1854), Nr. 4, S. 40–47
- [17] Clausius, R.: *Abhandlungen über die mechanische Wärmetheorie*. Braunschweig: Friedrich Vieweg und Sohn, 1864
- [18] Coe, L.: *The Telegraph. A History of Morse's Invention and its Predecessors in the United States*. Jefferson, North Carolina: McFarland, 2003
- [19] Coe, L.: *The telephone and its several inventors: A history*. Jefferson, North Carolina: McFarland, 2006
- [20] Computer Laboratory, University of C.: *University of Cambridge Computer Laboratory May 1949 members*. <http://creativecommons.org/licenses/by/2.0/>. Version: 1949. – Creative Commons License 2.0, Typ: BY
- [21] Corporation, Rand: *A million random digits with 100,000 normal deviates*. New York: Free Press, 1955
- [22] Corporation, Sony: *Method of recording on recording medium and recording device, and method of reproducing from recording medium and reproducing device*. U.S. Patent No. 6.980.498, 2004
- [23] Denniss: *Foto eines Volksempfängers*.  
<http://creativecommons.org/licenses/by-sa/3.0/>. Version: 2006. – Creative Commons License 3.0, Typ: Attribution-ShareAlike Unported

- [24] Department, Radio: The vacuum detector and how it works. In: *The Electrical Experimenter* 4 (1916), August, S. 250, 288
- [25] Drescher, W.; Fettweis, G.: VLSI Architectures for Multiplication in  $GF(2^m)$  for Application Tailored Digital Signal Processors. In: *Workshop on VLSI Signal Processing IX, San Francisco, CA*, 1996, S. 55–64
- [26] Duda, J.: Asymmetric Numeral Systems. In: *CoRR* abs/0902.0271 (2009)
- [27] Elias, P.: Coding for noisy channels. In: *IRE Convention Record* (1955), Nr. 4, S. 37–47
- [28] Ertel, W.: *Angewandte Kryptographie*. München: Hanser-Verlag, 2007
- [29] Ertel, W.: *Angewandte Kryptographie*. 4. Auflage. München: Hanser-Verlag, 2012
- [30] Evenson, A. E.: *The Telephone Patent Conspiracy of 1876: The Elisha Gray – Alexander Bell Controversy*. Jefferson, North Carolina: McFarland, 2000
- [31] Facebook: *zstd*. GitHub-Repository. <https://github.com/facebook/zstd>
- [32] Fano, R.: The transmission of information. Cambridge, MA: Research Laboratory of Electronics at MIT, 1949 (65). – Forschungsbericht. – Technical Report No. 65
- [33] GEO kompakt: *Der Mensch und seine Gene*. Bd. 7. 2006
- [34] Gilbert, W.: *De Magnete, Magneticisque Corporibus, et de Magno Magnete Tellure*. London: Peter Short, 1600
- [35] Golay, M. J. E.: Notes on Digital Coding. In: *Proceedings of the Institute of Radio Engineers* 37 (1949), Nr. 6, S. 657
- [36] Google: *pik*. GitHub-Repository. <https://github.com/google/pik>
- [37] Gray, F.: *U.S. Patent No. 2.632.058*. 1953
- [38] Hadamard, J.: Résolution d'une question relative aux déterminants. In: *Bulletin des Sciences Mathématiques* 17 (1893), S. 240–246
- [39] Hamming, R. W.: Error-detecting and Error-correcting Codes. In: *Bell System Technical Journal* 2 (1950), Nr. 26, S. 147–160
- [40] Hankerson, D. R.; Hoffman, D. G.; Leonard, D. A.; Lindner, C. C.; Phelps, K. T.; Rodger, C. A.; Wall, J. R.: *Coding Theory and Cryptography. The Essentials*. Second Edition. New York: Marcel Dekker, Inc., 2000
- [41] Hertz, H.: *Über die Beziehung zwischen Licht und Elektrizität*. Heidelberg, September 1889
- [42] Holzmann, G. J.; Pehrson, B.: *The early history of data networks*. Hoboken, NJ: Wiley, 2003
- [43] Horgan, J.: Profile: Claude E. Shannon: unicyclist, juggler and father of information theory. In: *Scientific American* (1990), Januar, S. 16–17
- [44] Huffman, D.: A Method for the Construction of Minimum-Redundancy Codes. In: *Proceedings of the Institute of Radio Engineers* 40 (1952), September, Nr. 9, S. 1098–1101
- [45] Huurdeman, A. A.: *The Worldwide History of Telecommunications*. New York: John Wiley and Sons, 2003
- [46] Jacquez, E.: *Claude Chappe: Notice biographique*. Paris: Alphonse Picard et Fils, 1893
- [47] Kammerer, K.: *Nachrichtenübertragung*. Wiesbaden: Vieweg+Teubner Verlag, 2008
- [48] Kane, J. N.: *Famous First Facts: A Record of First Happenings, Discoveries and Inventions in American History*. New York: H. W. Wilson, 1981
- [49] Karpfinger, C.; Meyberg, K.: *Algebra: Gruppen – Ringe – Körper*. Heidelberg: Spektrum Akademischer Verlag, 2010
- [50] Kemnitz, A.: *Mathematik zum Studienbeginn: Grundlagenwissen für alle technischen, mathematisch-naturwissenschaftlichen und wirtschaftswissenschaftlichen Studiengänge*. Wiesbaden: Vieweg+Teubner Verlag, 2010
- [51] Korpela, J.: *Unicode Explained*. Sebastopol: O'Reilly Media, 2006
- [52] Kraft, Leon G.: *A device for quantizing, grouping, and coding amplitude modulated pulses*. Boston, Massachusetts Institute of Technology, Diplomarbeit, 1949
- [53] Krey, U.; Owen, A.: *Basic Theoretical Physics: A Concise Overview*. Berlin, Heidelberg, New York: Springer-Verlag, 2007

- [54] Lagrange, J. L.: Nouvelle méthode pour résoudre les problèmes indéterminés en nombres entiers. In: *Mémoires de l'Academie royale des sciences et Belles-Lettres de Berlin* (1770), Nr. 24
- [55] Lebow, I.: *Information Highways and Byways*. Piscataway, NJ, USA: IEEE Press, 1995
- [56] Lochmann, D.: *Vom Wesen der Information*. Norderstedt: Books on Demand GmbH, 2006
- [57] Lzfse: *lzfse*. GitHub-Repository. <https://github.com/lzfse/lzfse>
- [58] MacKay, D. J. C.: *Information Theory, Inference, and Learning Algorithms*. Cambridge: Cambridge University Press, 2010
- [59] MATH 690A, Lecture N.: *BCH Codes*. Iowa State University, Fall 2004
- [60] Mellor, C. M.: *Louis Braille: A Touch of Genius*. Boston: National Braille Press, 2006
- [61] Mission X, Staffel 1: *Letzte Chance Transatlantik*. DVD, UIG Entertainment GmbH, 2006
- [62] Mobilfunk, Informationszentrum: *Funkzellen und Netze*. <http://www.izmf.de>. Version: 2013
- [63] Moore, G.: Cramming More Components onto Integrated Circuits. In: *Electronics Magazine* 38 (1965), April, Nr. 8
- [64] Muller, D. E.: Application of Boolean Algebra to Switching Circuit Design and to Error Detection. In: *IRE Transactions on Electronic Computers* (1954), Nr. 3, S. 6–12
- [65] Museum für Kommunikation: *In 28 Minuten von London nach Kalkutta*. Zürich: Chronos Verlag, 2000
- [66] N100400, ISO/IEC JTC 1.: *JPEG White Paper: JPEG XL image coding system v2.0*. White paper, January 2023
- [67] Nelson, M.: Data Compression with the Burrows-Wheeler Transform. In: *Dr. Dobb's Journal* 9 (1996), S. 46–50
- [68] newsticker, Heise: *Heise Newsticker-Nachricht vom 23.3.2008*. <http://www.heise.de>
- [69] Paley, R. E. A. C.: On orthogonal matrices. In: *Journal of Mathematics and Physics* 12 (1933), S. 311–320
- [70] Plambeck, Thane: *Portraitphoto von Elwyn Berlekamp*. <http://creativecommons.org/> licenses/by/2.0/. Version: 2005. – Creative Commons License 2.0, Typ: Attribution Generic
- [71] Polybius: *Polybii historiarum quae supersunt*. Leipzig: Tauchnitz, 1816
- [72] Prange, E.: Cyclic error-correcting codes in two symbols / Electronics Research Directorate, Air Force Cambridge Research Center. 1957 (No. AFCRC-TN-57-103). – Forschungsbericht. – ASTIA Document No. AD133749
- [73] Pschyrembel: *Klinisches Wörterbuch*. Berlin: Walter de Gruyter Verlag, 2004
- [74] Reed, I. S.: A Class of Multiple-Error-Correcting Codes and the Decoding Scheme. In: *IEEE Transactions on Information Theory* 4 (1954), S. 38–49
- [75] Reed, I. S.; Solomon, G.: Polynomial Codes Over Certain Finite Fields. In: *Journal of the Society for Industrial and Applied Mathematics* 8 (1960), Nr. 2, S. 300–304
- [76] Rissanen, J. J.: Generalized kraft inequality and arithmetic coding. In: *IBM Journal of Research and Development* 20 (1976), May, S. 198–203
- [77] Roth, R. R.: A History of Lagrange's Theorem on Groups. In: *Mathematics Magazine* 74 (2001), Nr. 2, S. 99–108
- [78] Shannon, C. E.: *A Symbolic Analysis of Relay and Switching Circuits*. Masterthesis, Massachusetts Institute of Technology, 1937
- [79] Shannon, C. E.: A Symbolic Analysis of Relay and Switching Circuits. In: *Transactions of the American Institute of Electrical Engineers* 57 (1938), Nr. 12, S. 713–723
- [80] Shannon, C. E.: *An algebra for theoretical genetics*. Dissertation, Massachusetts Institute of Technology, 1940
- [81] Shannon, C. E.: A Mathematical Theory of Communication. In: *Bell System Technical Journal* 27 (1948), S. 379–423, 623–656
- [82] Shannon, C. E.; Weaver, W.: *The Mathematical Theory of Communication*. Illinois: University of Illinois Press, 1949
- [83] Singh, S.: *Fermat's letzter Satz*. München: Deutscher Taschenbuch Verlag, 2000

- [84] Singleton, R. C.: Maximum distance q-nary codes. In: *IEEE Transactions on Information Theory* 10 (1964), April, Nr. 2
- [85] Staelin: *Portraitphoto von Abraham Lempel*. <http://creativecommons.org/licenses/by/3.0/>. Version: 2007. – Creative Commons License 3.0, Typ: Attribution Unported
- [86] Standage, T.: *The Victorian Internet*. New York: Walker Publishing Company, 1998
- [87] Stingl, P.: *Mathematik für Fachhochschulen: Technik und Informatik*. München: Hanser-Verlag, 2009
- [88] Stix, G.: Profile: Information Theorist David A. Huffman. In: *Scientific American* 265 (1991), September, Nr. 3, S. 54–58
- [89] Stryer, L.: *Biochemie*. 6. Auflage. Heidelberg: Spektrum Akademischer Verlag, 2007
- [90] Sylvester, J. J.: Thoughts on inverse orthogonal matrices, simultaneous sign successions, and tessellated pavements in two or more colours, with applications to Newton's rule, ornamental tile-work, and the theory of numbers. In: *Philosophical Magazine* 34 (1867), S. 461–475
- [91] Taylor, L. W.: The Untold Story of the Telephone. In: *American Physics Teacher* (1937), December
- [92] Thaer, Clemens (Hrsg.): *Euklid. Die Elemente. Buch I – XIII.* Frankfurt: Verlag Harri Deutsch, 2003 (Ostwalds Klassiker)
- [93] Thompson, R. L.: *Wiring a continent*. New York: Arno Press, 1972
- [94] Thompson, S. P.: *Philipp Reis: inventor of the telephone. A biographical sketch, with documentary testimony, translations of the original papers of the inventor and contemporary publications*. London, New York: E. & F. N. Spon, 1883
- [95] Tribus, M.; McIrvine, E. C.: Energy and Information. In: *Scientific American* 225 (1971), September, Nr. 3, S. 179–188
- [96] Tsui, J. B. Y.: *Fundamentals of Global Positioning System Receivers. A Software Approach*. 2nd edition. Hoboken, New Jersey: John Wiley and Sons, 2005
- [97] Vermani, L. R.: *Elements of algebraic coding theory*. New York: Chapman & Hall Mathematics, 1996
- [98] Viterbi, A. J.: Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. In: *IEEE Transactions on Information Theory* 13 (1967), Nr. 2, S. 260–269
- [99] Watson, T. A.: *Exploring Life: The Autobiography of Thomas A. Watson*. New York: D. Appleton & Company, 1926
- [100] Webster's New World Dictionary. Cleveland, Ohio, 1989
- [101] Welch, T. A.: A Technique for High-Performance Data Compression. In: *IEEE Computer* 17 (1984), Nr. 6, S. 8–19
- [102] Wikipedia: *Antonio Meucci*. Webartikel. [http://en.wikipedia.org/wiki/Antonio\\_Meucci](http://en.wikipedia.org/wiki/Antonio_Meucci)
- [103] Wikipedia: *Russian-American Telegraph*. Webartikel. [http://en.wikipedia.org/wiki/Russian-American\\_telegraph](http://en.wikipedia.org/wiki/Russian-American_telegraph)
- [104] Wikipedia: *Submarine cable cross-section 3D*. Public domain image. [http://en.wikipedia.org/wiki/Submarine\\_communications\\_cable](http://en.wikipedia.org/wiki/Submarine_communications_cable)
- [105] Wikisource: *Popular Science Monthly, volume 71, page 228*. Webartikel. [http://en.wikisource.org/wiki/Page:Popular\\_Science\\_Monthly\\_Volume\\_71.djvu/234](http://en.wikisource.org/wiki/Page:Popular_Science_Monthly_Volume_71.djvu/234)
- [106] Wiles, A.: Modular Elliptic Curves and Fermat's last theorem. In: *Annals of Mathematics* 141 (1995), S. 443–551
- [107] Willems, W.: *Codierungstheorie*. Berlin: De Gruyter Verlag, 1999
- [108] Ziv, J.; Lempel, A.: A universal algorithm for sequential data compression. In: *IEEE Transactions on Information Theory* (1977), Nr. 23, S. 337–343
- [109] Ziv, J.; Lempel, A.: Compression of individual sequences via variable-rate coding. In: *IEEE Transactions on Information Theory* (1978), Nr. 24, S. 530–536

# Namensverzeichnis

## A

- Abel, Niels Henrik, 91  
Agamemnon, 15  
Aineias Taktikos, 17, 74  
Aischylos, 15  
Ampère, André-Marie, 28  
Aschoff, Volker, 73

## B

- Barbier de La Serre, Charles, 182, 220  
Bardeen, John, 62, 63  
Baudot, Jean-Maurice-Émile, 77, 593  
Bell, Alexander Graham, 46, **47**  
Berlekamp, Elwyn R., 424  
Berrou, Claude, 72  
Bose, Raj Chandra, 394  
Braille, Louis, 182, 220  
Brattain, Walter, 62, 63  
Braun, Karl Ferdinand, 58  
Burrows, Michael, 72, 289, 313

## C

- Cantor, Georg, 166  
Carroll, Lewis, 298  
Cayley, Arthur, 97  
Chap  e, Claude, 19  
Clausius, Rudolf Julius Emanuel, 322  
Coe, Lewis, 73  
Cooke, William F., 30, **32**, 224

## D

- Daemen, Joan, 120  
Duda, Jaros  aw, 72, 73, 261

## E

- Eaton, Ashael K., 54  
Edison, Thomas Alva, 45  
Elias, Peter, 71  
Euripides, 15

## F

- Faggin, Federico, 64  
Fano, Robert M., 71, 245, 249  
Fessenden, Reginald Aubrey, 60  
Fibonacci, 302  
Field, Cyrus W., 40  
Forest, Lee de, 60

## G

- Gale, Leonard D., 36, 38  
Galois,   variste, 100, 102  
Gamble, John, 23, 24  
Gau  , Carl Friedrich, 29, 74, 75  
Gilbert, William, 26  
Gisborne, Frederic N., 41  
Glavieux, Alain, 72  
Golay, Marcel J. E., 522  
Gold, Robert, 614  
Gray, Elisha, 48, **49**  
Grove, Andrew, 63

## H

- Hadamard, Jacques S., 445  
Hamming, Richard W., 71, 353, **354**  
Ha  y, Valentin, 182  
Henry, Joseph, **31**, 36, 38, 54  
Hocquenghem, Alexis, 394  
Hubbard, Greene Gardiner, 46

- Huffman, David A., 71, **249**, 250, 299  
Hughes, David Edward, 45  
Huth, J. S. Gottfried, 54  
Huurdeman, Anton A., 73

## J

- Jackson, Charles T., 34

## K

- Kasami, Tadao, 621  
Kilby, Jack, 63  
Kraft, Leon Gordon, 176

## L

- Lagrange, Joseph-Louis, 155  
Lempel, Abraham, 72, 239, 274  
Lilienfeld, Edgar, 63  
Luhn, Hans Peter, 350

## M

- MacKay, David J. C., 541, 563  
Marconi, Guglielmo, 58  
Maury, Matthew F., 41  
Maxwell, James Clerk, 56  
McElroy, Theodore Roosevelt, 203  
Meucci, Antonio, 52  
Milet, Thales von, 26  
Moore, Gordon, 63, 64  
Morse, Samuel F. B., 33, **34**  
Muller, David E., 71  
Munke, Georg Wilhelm, 30  
Murray, Lord George, 23, 25

**N**

- Nelson, Mark, 290  
Neumann, John von, 322  
Noyce, Robert, 63

**O**

- Ohm, Georg Simon, 28  
Ørsted, Hans Christian, 28  
Orton, William, 51

**P**

- Paley, Raymond E. A. C., 449, 636  
Pasco, Richard C., 72  
Pisa, Leonardo da, 302  
Polybius, 17, 18, 74  
Prange, Eugene, 71  
Preece, Sir William, 58, 59  
Primrose, Frank J., 77

**R**

- Rademacher, Hans Adolph, 635  
Ray-Chaudhuri, Dijen Kumar, 394  
Reed, Irving S., 71

- Reis, Philipp, 31, 52, **55**  
Righi, Augusto, 58  
Rijmen, Vincent, 120  
Rissanen, Jorma J., 72

**S**

- Sanders, Thomas, 46  
Schilling von Cannstatt, Paul Ludwig, 30  
Shannon, Claude, 7, **66**, 71, 322  
Shockley, William, 62, 63  
Singleton, Richard C., 512  
Soemmerring, Samuel Thomas, 27, 28, 74  
Solomon, Gustave, 71  
Sophokles, 15  
Standage, Tom, 73  
Stearns, Joseph B., 45  
Steinheil, Carl August von, 75  
Storer, James, 279  
Stroustrup, Bjarne, 290  
Sturgeon, William, 28  
Sylvester, James Joseph, 445  
Szymanski, Thomas, 279

**T**

- Thitimajshima, Punya, 72  
Thompson, Silvanus P., 56

**V**

- Vail, Alfred L., **37**, 38  
Vandermonde, Alexandre-Théophile, 396  
Viterbi, Andrew James, 72, 476  
Volta, Alessandro, 27

**W**

- Walsh, Joseph Leonard, 636  
Watson, Thomas A., 46  
Weber, Wilhelm Eduard, 29, 74, 75  
Welch, Lloyd R., 424  
Welch, Terry A., 72, 239, 281  
Wheatstone, Charles, 31, **32**, 45, 224  
Wheeler, David J., 72, 313  
White, Samuel S., 48  
Whitehouse, E. O. Wildman, 43  
Wiles, Andrew, 409

**Z**

- Ziv, Jacob, 72, 239, 274

# Sachwortverzeichnis

## Symbole

2421-Code, 187  
4004-Prozessor, 64  
8421-Code, 187

## A

Abbildung  
  bijektive, 85  
  identische, 94  
  injektive, 85  
  surjektive, 85  
ABC-Telegraf, 45  
Abel'sche Gruppe, 91  
Abstand  
  Hamming-, 355  
Abzählbare Menge, 166  
Acknowledge-Bit, 583  
Additive Gruppe, 91  
Adenin, 178, 179  
Advanced Encryption Standard, 120  
AES, 120  
Äther, 56  
Agamemnon, 15  
Aiken-Code, 187  
Aineias-Telegraf, 17  
Algebraische Struktur, 89  
Algorithmus  
  Berlekamp-Welch-, 424  
  euklidischer, 86  
    für Polynome, 115  
  Viterbi-, 475, **476**  
ALOHA-Protokoll, 594  
Alphabet  
  Code-, 168  
  Devanagari-, 184  
  Quellen-, 168

Alternate Mark Inversion, 571  
AMI-Codierung, 571  
  modifizierte, 571  
Aminosäure, 178  
Amplitude, 586  
Amplitudenmodulation, 45  
Anallagmatic pavement, 445  
Analog-Digital-Wandler, 435  
Anode, 60  
ANS-Codierung, 260  
Arithmetik  
  modulare, 83  
Arithmetische Codierung, 72, 254  
Artikelnummer  
  europäische, 346  
ASCII, 182  
Assoziativität, 89  
Attention-Signal, 79  
Audion, 60  
Ausrollen  
  eines endlichen Automaten, 475  
Autokorrelation, 606  
Axiome  
  von Huntington, 156

**B**

Barcode-Scanner, 346  
Base, 179  
Basic Multilingual Plane, 184, 186  
Basis, 142  
Baud, 78  
Baudot  
  -Code, 78  
  -Telegraf, 77, 593  
BCD-Code, 187  
BCH-Code, 394  
BD, 123, 435  
Berlekamp-Welch-Algorithmus, 424

**C**

CAN-Bus, 593  
CD, 123, 435  
  -DA, 435  
  Digital Audio, 435  
CDMA, 597  
  asynchrones, 604  
  synchrones, 604  
Charakteristik, 102, 157  
Charakteristische Gleichung, 201  
Chip, 597

- Chipsequenz, 597  
 CIRC, 435  
 Code, 168, **170**
  - Distanz, 356
  - alphabet, 168
  - multiplexverfahren, 597
  - rate, 470, 541, 575
  - wort, 170
  - wortlänge
    - mittlere, 242
  - 2421-, 187
  - 8421-, 187
  - Aiken-, 187
  - AMI-, 571
    - modifizierter, 571
  - ANS-, 260
  - Baudot-, 78
  - BCD-, 187
  - BCH-, 394
  - binärer, 168
  - Biphase-Mark-, 573
  - Biphase-Space-, 573
  - Bit-, 569
  - Block-, 178, 574
  - dualer, 193
  - EFM-, 579
  - einschrittiger, 188
  - erweiterter, 360
  - Excess-3-, 187
  - Faltungs-, 72, **469**
  - Fano-, 245
  - genetischer, 178
  - Golay-, 510, **522**
    - erweiterter, 529
    - ternärer, 531
    - zyklischer, 524
  - Gold-, 617
  - Gray-, 187, 188, 575
  - Hadamard-, 445
  - Hamming-, 71, 192, **370**
  - Huffman-, 247
  - Katastrophen-, 474
  - längenvariabler, 171
  - linearer, 188
  - Manchester-, 572
  - Manchester-II-, 573
  - MDS-, 512, 513
    - triviale, 513
  - trivialer, 513
  - MMS43-, 576
  - NRZ-, 567
  - OVSF-, 602
  - Paritäts-, 189, 353
  - perfekter, 510, **514**, 518
    - trivialer, 520
  - präfixreier, 171, 172, 251
  - progressiver, 188
  - Prüfziffer-, 341
  - pseudoternärer, 571
  - Rademacher-, 635
  - rANS-, 261
  - redundanzfreier, 325
  - Reed-Muller-, 460
    - höherer Ordnung, 466
  - Reed-Solomon-, 414
  - RLL-, 578
  - RZ-, 571
    - bipolarer, 572
    - unipolarer, 572
  - selbstdualer, 194, 195
  - Shannon-, 242
  - Simplex-, 194, **456**
  - Spreiz-, 602
  - Stibitz-, 187
  - ternärer, 571
  - Turbo-, 73
  - uABS-, 271
  - umschaltbarer, 78
  - Walsh-, 636
  - Zahlen-, 186
  - Zeichen-, 18, 181
  - zyklischer, 192, **377**
  - Codierung, 168
    - 2421-, 187
    - 8421-, 187
  - Aiken-, 187
  - AMI-, 571
    - modifizierte, 571
  - ANS-, 260
  - arithmetische, 254
  - Baudot-, 78
  - BCD-, 187
  - BCH-, 394
  - binäre, 168
  - Biphase-Mark-, 573
  - Biphase-Space-, 573
  - Bit-, 569- Block-, 178, 574
- Blockweise, 330
- EFM-, 579
- einschrittige, 188
- Entropie-, 242
- Excess-3-, 187
- Faltungs-, 72, **469**
- Fano-, 245
- genetische, 178
- gleichstromfreie, 568
- Golay-, 510, **522**
  - erweiterte, 529
  - ternäre, 531
  - zyklische, 524
- Gold-, 617
- Gray-, 187, 188, 575
- Hadamard-, 445
- Hamming-, 71, 192, **370**
- Huffman-, 247
- Katastrophen-, 474
- längenvariable, 171
- lineare, 188, 189
- Manchester-, 572
- Manchester-II-, 573
- MDS-, 512, 513
  - triviale, 513
- MMS43-, 576
- Move-to-front-, 295
- NRZ-, 567
- OVSF-, 602
- Paritäts-, 189, 353
- perfekte, 510, **514**, 518
  - triviale, 520
- präfixfreie, 171, 172, 251
- progressive, 188
- Prüfziffer-, 341
- pseudoternäre, 571
- Rademacher-, 635
- rANS-, 261
- redundanzfreie, 325
- Reed-Muller-, 460
  - höherer Ordnung, 466
- Reed-Solomon-, 414
- RLL-, 578
- RZ-, 571
  - bipolare, 572
  - unipolare, 572
- Shannon-, 242

- Simplex-, 194, **456**  
 Spreiz-, 602  
 Stibitz-, 187  
 Substitutions-, 274  
 systematische, 191  
 ternäre, 571  
 Turbo-, 73  
 uABS-, 271  
 verlustbehaftete, 169  
 verlustfreie, 169  
 Walsh-, 636  
 Zahlen-, 186  
 Zeichen-, 18, 181  
 Codierungstheorie, 65  
 Communication channel, 68, 198  
 Compact Disc, 123, 435  
 Convolutional code, 72  
 CRC, 72, **377**  
     -Algorithmus, 387  
 Cyclic Redundancy Check, 72, **377**  
 Cytosin, 178, 179
- D**
- Data Encryption Standard, 120  
 Datenkompression, 237  
 Datenquelle, 228, **229**  
 Datenrahmen, 583  
 De Magnete, 26  
 Degree, 111  
 Delimiter-Bit, 583  
 DES, 120  
 Desoxyribonukleinsäure, 179  
 Destination, 68  
 Devanagari-Alphabet, 184  
 Dictionary, 239  
 Differenz  
     symmetrische, 107  
 Digital Versatile Disc, 123, 435  
 Digitale Frequenzmodulation, 579  
 Dimension  
     eines Vektorraums, 143  
 Diode, 60  
 Disc  
     Blu-ray, 123, 435  
     Compact, 123, 435  
     Digital Versatile, 123, 435
- Diskreter Logarithmus, 130  
 Disparität, 577  
 Distanz  
     Code-, 356  
     Hamming-, 354, 355  
 Distributivgesetz, 98, 156  
 Divisionsrest, 112  
 Dualer Code, 193  
 Duplexbetrieb, 45  
 DVD, 123, 435  
 Dynamische Programmierung, 477
- E**
- EAN, 346  
 Eaton-Spencer case, 54  
 echte Körpererweiterung, 102  
 echte Ringerweiterung, 108  
 Écriture nocturne, 220  
 EFM-Codierer, 443  
 EFM-Codierung, 579  
 Eight-to-Fourteen-Modulation, 443, 579  
 Einheitsmatrix, 107, 108  
 Einschrittiger Code, 188  
 Elektromagnetische Welle, 56, 605  
 Element  
     inverses, 156  
     linksinverses, 95  
     linksneutrales, 95  
     neutrales, 90, 156  
     primitives, 401  
     rechtsinverses, 95  
     rechtsneutrales, 95  
 Elemente  
     von Euklid, 86  
 Empfänger, 68, 198  
 End-of-Frame-Field, 583  
 Endlich erzeugbar, 140  
 Endlicher Körper, 99, **111**, 420  
     Konstruktion, 116  
     Logarithmus, 129  
     schnelles Rechnen, 123  
 ENIAC, 62  
 Entropie, 70, **317**, 320, 322, 323  
 Entropiecodierung, 242  
 Ergodizität, **236**, 328  
 Erweiterter Golay-Code, 529
- Erzeugendensystem, 140  
     minimales, 141  
 Euklidischer Algorithmus, 86  
     für Polynome, 115  
 Europäische Artikelnummer, 346  
 Excess-3-Code, 187
- F**
- F1-Frame, 436  
 F2-Frame, 442  
 F3-Frame, 442  
 Fackelcode, 18  
 Fackelpost, 15, 16  
 Faltungscode, 72, **469**  
 Faltungscodierung  
     systematische, 474  
 Fano-Bedingung, 172  
 Fano-Codierung, 245  
 Fast Frequency Hopping, 595  
 FDMA, 590  
 Fehler  
     -erkennung, 353  
     -korrektur, 353  
     -polynom, 425  
     -rahmen, 583  
     -vektor, 362  
 Fermat'scher Satz  
     großer, 409  
     kleiner, 409  
 Fernschreiber, 20  
 FFH, 595  
 FHSS, 594  
 Fibonacci  
     -Folge, 302  
     -Zahl, 302  
 Field, 98  
 Finite field, 99  
 Flexray-Bus, 593  
 FM, 579, 580  
 Folge  
     Fibonacci-, 302  
     Gold-, 614  
     Kasami-, 621  
     maximaler Länge, 637  
     pseudozufällige, 584, **609**  
 Frequency Hopping, 595

- Frequency Hopping Spread Spectrum, 594, 595  
 Frequenz, 586  
 Frequenzmodulation  
     digitale, 579  
     modifizierte, 580  
 Frequenzmultiplexing, 590  
 Frequenzspreizung, 595  
 Füllbit, 582  
 Funktion  
     bijektive, 85  
     injektive, 85  
     Rademacher-, 635  
     surjektive, 85
- G**
- Gallows telephone, 48  
 Galoisfeld, 100  
 Galoiskörper, 100  
 Gedächtnisbehaftete Quelle, 228  
 Gedächtnislose Quelle, 228, 230  
 Generatormatrix, 147  
     reduzierte Form, 148  
     systematische Form, 148  
 Generatorpolynom, 130, 377  
 Genetischer Code, 178  
 Gesetz  
     Distributiv-, 156  
     Kommutativ-, 156  
     Moore'sches, 64, 80  
 Gleichstromfreiheit, 568  
 Gleichung  
     charakteristische, 201  
 Global Positioning System, 596  
 Golay-Code, 510, **522**  
     erweiterter, 529  
     ternärer, 531, 532  
     zyklischer, 524  
 Gold-Code, 617  
 Gold-Folge, 614  
 GPS, 595, 617, 618  
     -Link, 618  
 Grad, 111  
     einer Gruppe, 94  
 Gray-Code, 187, 188, 575  
 Great Atlantic Bubble, 43  
 Größter gemeinsamer Teiler, 85  
 Großzelle, 592  
 Gruppe, 89, 90  
     Abel'sche, 91  
     additive, 91  
     kommutative, 91  
     multiplikative, 91  
     symmetrische, 94, 97  
 Guanin, 178, 179  
 Guttapercha, 41
- H**
- Hadamard  
     -Code, 445  
     -Matrix, 445  
     -Vermutung, 449  
 Halbgruppe, 90  
 Hamming  
     -Abstand, 355  
     -Code, 71, 192, **370**, 520  
         erweiterter, 194  
         zyklischer, 197  
     -Distanz, 354, 355  
     -Gewicht, 362  
     -Kugel, 357, 537  
     -Medaille, 370  
     -Würfel, 354, 482  
 Harmonischer Telegraf, 46  
 Hauptdiagonale, 107  
 High  
     -Surrogate, 186  
     Byte, 436  
 Hochintegration, 63  
 Hop, 595  
 Horner-Schema, 128  
 Hütchenspiel, 207  
 Huffman-Codierung, 247  
     ternäre, 301  
 Hughes printer, 45  
 Huntington'sche Axiome, 156
- I**
- Ideal, 110  
     echtes, 110  
     triviales, 110
- Identische Abbildung, 94  
 Indikator, 20  
 Indikatorbit, 279  
 Information, 68, 70, **203**, **317**  
 Information source, 68  
 Informations  
     -gehalt, 320  
     -quelle, 228, **229**  
     -senke, 68  
     -theorie, 65, 67  
 Inneres Produkt, 148, 164  
 Integrierter Schaltkreis, 63  
 Intel, 63  
     4004-Prozessor, 64  
 Interleaver, 438  
 Inverses Element, 156  
 Irreduzibles Polynom, 114
- J**
- JPEG-XL, 261
- K**
- Kanal, 198  
 Kanalcode  
     linearer, 361  
 Kanalcodierung, 340  
 Kanalcodierungstheorem, 71, **541**  
 Kanalkapazität, **199**  
 Kasami-Folge, 621  
 Katastrophencodierung, 474  
 Kathode, 60  
 Klappentelegraf, 23  
 Kleinzelle, 592  
 Körper, 97, 98  
     endlicher, 99, **111**, 420  
         Konstruktion, 116  
         Logarithmus, 129  
         schnelles Rechnen, 123  
 Körpererweiterung, 102  
 Kommunikationsquelle, 68  
 Kommutative Gruppe, 91  
 Kommutative Verknüpfung, 91  
 Kommutativgesetz, 156  
 Kompression, 237  
     LZ77-, 72, 275

- LZ78-, 72, 280  
 LZFSE, 261  
 LZSS-, 279  
 LZW-, 284  
 Kompressionsrate, 239  
 Kongruenz, 83, 118  
 Kontinuum, 166  
 Kontrollmatrix, 151, 364  
 Koppelbits, 443  
 Korrekturvektor, 362  
 Korrelationsprodukt  
     diskretes, 605  
 Kraft'sche Ungleichung, 175, **176**, 318  
 Kreuzkorrelation, 606
- L**
- Längenvariabler Code, 171  
 Land, 444, 501  
 Latin-Zeichensatz, 184  
 Lauflängencodierung, 169  
 Leitkoeffizient, 133  
 Leitungscodierung, 567  
 Lempel-Ziv  
     -77-Kompression, 72, 275  
     -78-Kompression, 72, 280  
     -Storer-Szymanski-Kompression,  
         279  
     -Welch-Kompression, 284  
 Linear  
     abhängig, 141  
     unabhängig, 141  
 Lineare Codierung, 189  
 Linearer  
     Code, 188  
     Kanalcode, 361  
 Linearkombination, 140  
 Link (GPS), 618  
 Linksideal, 110  
 Links inverses Element, 95  
 Links neutrales Element, 95  
 Liquid transmitter, 49  
 Lock-up State, 609  
 Logarithmus  
     diskreter, 130  
     dualis, 211  
 Logik
- negative, 567  
 positive, 567  
 Long code, 607  
 Look-ahead buffer, 275  
 Low  
     -Surrogate, 186  
     Byte, 436  
 Luhn-Test, 351  
 LZ77-Kompression, 72, 275  
 LZ78-Kompression, 72, 280  
 LZFSE-Kompression, 261  
 LZSS-Kompression, 279  
 LZW-Kompression, 284
- M**
- Magnetic Drum, 573  
 Magnetic Telegraph Company, 38  
 Manchester Mark I, 573  
 Manchester-Codierung, 572, 573  
 Manchester-II-Codierung, 573  
 Manhattan-Projekt, 370  
 Mariner 9, 453  
 Mark, 570, 573  
 Markov-Quelle, 231, 234  
 Matrix  
     Generator-, 147  
     Kontroll-, 151  
     orthogonale, 445, 503  
     Rotations-, 291  
     Vandermonde-, 396  
 Maximum Length Sequence, 610  
 Maximum-Likelihood-Decoder, **362**,  
     424, 478, 537, 564  
 MDS-Code, 512, 513  
     trivialer, 513  
 Mehrheitsentscheider, 460  
 Menge  
     abzählbare, 166  
     überabzählbare, 166  
 Microcom Network Protocol, 217  
 Mikrozelle, 592  
 Minimalpolynom, **409**, 410, 611  
 Mittlere Codewortlänge, 242  
 MMS43-Codierung, 576  
 MNP-5, 217  
 Mobilfunkzelle, 592
- Modifizierte Frequenzmodulation, 580  
 Modul, 83  
 Modularre Arithmetik, 83  
 Modulation, 585  
     digitale, 586  
 Monoid, 90  
 Moore'sches Gesetz, 64, 80  
 Morse  
     -Code  
         amerikanischer, 38  
         internationaler, 38  
         kontinentaler, 38  
     -Telegraf, **33**, 37  
 Move-to-front-Codierung, 295  
 Multiplexverfahren, 590  
 Multiplikative Gruppe, 91  
 Mutterfolge, 614
- N**
- Nachrichtenübertragung  
     elektrische, 26  
 Nachtschrift, 220  
 Nadeltelegraf, 29  
 Nebenklasse, 154  
 Negative Logik, 567  
 Neunerkomplement, 187  
 Neutrales Element, 90, 156  
 Nibble, 582  
 Nichtnegative Zahl, 82  
 Noise source, 68  
 Noisy-Channel Coding Theorem, 71, **541**  
 Non-return-to-zero, 567  
     inverted, 569  
 Normierter Teiler, 114  
 NRZ-Codierung, 567  
 NRZI-Codierung, 569  
 Nukleotid, 179  
 Nullteilerfrei, 158  
 Nullvektor, 137
- O**
- Obergruppe, 96  
     echte, 96  
 Oberkörper, 102

echter, 102  
 Oberring, 108  
     echter, 108  
 Ohm'sches Gesetz, 28  
 On-Off Keying, 587  
 One-Time-Pad, 584  
 OOK, 587  
 Orestie, 15  
 Orthogonale Matrix, 445, 503  
 Orthogonalraum, 148, 149  
 OVSF-Code, 602

**P**

Paritätsbit, 183, 190  
 Paritätscode, 189, 353  
 Patent, 259, 261  
 Perfekter Code, 510, **514**, 518  
     trivialer, 520  
 Permutation, 94  
 Permutationsgruppe, 97  
 Phasenmodulation, 45  
 Phasenversatz, 586  
 Pik, 261  
 Pilotkanal, 622  
 Pit, 444, 501  
 Planartechnik, 63  
 Polare Koordinaten, 163  
 Polynom, 111  
     -division, 112  
     -funktion, 112  
     -ring, 111  
         Generator-, 130, 377  
         irreduzibles, 114  
         primitives, 611  
 Positiv definit, 164  
 Positive Logik, 567  
 Potenzmenge, 106  
 Präfixfreie Codierung, 172  
 Präfixfreier Code, 171, 172, 251  
 Primitives Element, 401  
 Primitives Polynom, 611  
 Primkörper, 102  
 Programmierung  
     dynamische, **477**  
 Progressiver Code, 188  
 Prüfbit, 361

Prüfziffercode, 341  
 Pseudo-Tetrade, 186  
 Pseudoternäre Codierung, 571  
 Pseudozufallsfolge, 584, **609**

**Q**

QAM, 589  
 Quadraturamplitudenmodulation, 589  
 Quadruplex-Telegraf, 45  
 Quelle, 228, **229**  
     Bernoulli'sche, 230  
     gedächtnisbehaftete, 228  
     gedächtnislose, 228, 230  
     Markov-, 231  
 Quellenalphabet, 168  
 Quellencodierung, 227, 237  
 Quellencodierungstheorem, 325

**R**

Rademacher-Code, 635  
 Rademacher-Funktion, 635  
 rANS-Codierung, 261  
 Raummultiplexverfahren, 592  
 Receiver, 68, 198  
 Rechenschieber, 129  
 Rechtsideal, 110  
 Rechtsinverses Element, 95  
 Rechtsneutrales Element, 95  
 Redundanz, **317**, 325  
 Redundanzfreie Codierung, 325  
 Redundanzprüfung  
     zyklische, 72  
 Reed-Muller-Code, 72, 460, 462  
     höherer Ordnung, 466  
 Reed-Solomon-Code, 123, 414  
     Cross-interleaved, 435  
 Regulator, 20  
 Rekurrenzgleichung, 201  
 Relais office, 39  
 Restfehlerwahrscheinlichkeit, 533, 541  
     bei der Fehlererkennung, 533  
     bei der Fehlerkorrektur, 537  
 Restklassenkörper, 99  
 Restklassenring, 116  
 Resynchronisation, 568

externe, 582  
 Return-to-zero, 572  
 Rijndael-Verschlüsselung, 120  
 Ring, 105  
     kommutativer, 105  
     mit 1, 105, 158  
     Polynom-, 111  
 Ringerweiterung, 108  
 RLL-Codierung, 578  
 Rotationsmatrix, 291  
 RS232-Schnittstelle, 567  
 Run-Length Encoding, 169  
 RZ-Codierung, 571  
     bipolare, 572  
     unipolare, 572

**S**

Satz  
     von Cayley, 97  
     von Lagrange, 155

Schaltelement  
     digitales, 62

Schaltkreis  
     integrierter, 63

Schnellschreiber, 20

Schranke  
     von Singleton, 511, 512

Scrambler, 584  
 SDMA, 592  
 Search window, 275  
 Sektor  
     einer Audio-CD, 444

Selbstdualer Code, 194, 195  
 Selbstkorrelation, 606  
 Selbsttaktender Code, 567  
 Semaphoren-Telegraf, 19  
 Sender, 68, 198  
 Shannon-Codierung, 242  
 Shannon-Limit, 543  
 Shell game, 207  
 Shift-and-add Property, 612  
 Short code, 607  
 Shutter-Telegraf, 23  
 Signalraumdiagramm, 589  
 Simplex-Code, 194, **456**  
 Singleton-Schranke, 511, 512

Skalarprodukt, 164  
 Slow Frequency Hopping, 595  
 Space, 570, 574  
 Spaltenvektor, 136  
 Spreizcode, 602  
 Spreizfaktor, 602  
 Standardbasis, 142  
 Start-of-Frame-Bit, 583  
 Steinheil-Telegraf, 75  
 Stibitz-Code, 187  
 Stirling-Formel, 548, 563  
 Stopfbit, 582  
 Streumenge, 550  
 Struktur  
     algebraische, 89  
 Störquelle, 68  
 Subchannel, 442  
 Subcode  
     einer Audio-CD, 442  
 Substitutionscodierung, 274  
 Suchfenster, 275  
 Surrogate, 186  
 Symmetrische  
     Differenz, 107  
     Gruppe, 94, 97  
 Synchronelegraf, 17, 74  
     von Aineias, 17  
 Syndrom, 364, 365  
 Syndromdecodierung, 361  
 Systematische  
     Codierung, 191  
     Faltungscodierung, 474

**T**

Tachygraf, 20  
 TDMA, 593  
 Teiler  
     normierter, 114  
     trivialer, 112  
     von Polynomen, 112  
 Telefonie, 45  
 Telegraf, 20  
     ABC-, 45  
     Aineias-, 17  
     Baudot-, 77, 593  
     Harmonischer, 46

Klappen-, 23  
 Morse-, 33, 37  
 Nadel-, 29  
 Semaphoren-, 19  
 Shutter-, 23  
 Steinheil-, 75  
 Typendruck-, 45  
 Telegrafenplateau, 41  
 Télégraphe, 20  
 Telefono, 52  
 Ternäre Codierung, 571  
 Ternärer  
     Code, 532  
     Golay-Code, 531  
     Huffman-Code, 301  
 Tetrade, 186  
 Tetraden-Code, 186  
 Theorem  
     Kanalcodierungs-, 71, **541**  
     Quellencodierungs-, 325  
 Thymin, 178, 179  
 Trägermenge, 106  
 Transformation  
     Burrows-Wheeler-, 73, 239, **288**  
 Transistor, 63  
 Transitionswahrscheinlichkeit, 231  
 Transmitter, 68, 198  
 Trellis-Diagramm, 475, 477  
 Triadische Darstellung, 278  
 Triodenröhre, 60  
 Triplet, 178  
 Trivialer  
     Code  
         perfekter, 520  
     Teiler, 112  
 Turbo-Code, 73  
 Turing-Maschine, 62  
 Typendrucktelegraf, 45

**U**

uABS-Codierung, 271  
 Überzählbare Menge, 166  
 Übertragungskanal, 68, 198  
 Umschaltcode, 78  
 Ungleichung  
     von Kraft, 175, **176**, 318

Universal Transformation Format, 185  
 Untergruppe, 96  
     echte, 96  
 Untergruppenkriterium, 96, 154  
 Unterkörper, 102  
     -kriterium, 102  
     echter, 102  
 Unterraumkriterium, 143  
 Unterring, 108  
     -kriterium, 109  
     echter, 108  
     mit 1, 108, 110  
 Untervektorraum, 143  
     trivialer, 144  
 UTF-*n*, 185

**V**

Vandermonde-Matrix, 396  
 Vektor  
     orthonormaler, 503  
     Spalten-, 136  
     Zeilen-, 136  
 Vektorraum, 136, 138  
     endlich dimensionaler, 143  
     endlich erzeugbarer, 140  
     orthogonaler, 148  
     unendlich dimensionaler, 143  
 Verknüpfung  
     kommutative, 91  
 Verlustbehaftete Codierung, 169  
 Verlustfreie Codierung, 169  
 Verstärker, 61  
 Vertauschungsfehler, 344  
 Viterbi-Algorithmus, 72, 475, **476**  
 Volksempfängers, 61  
 Volta'sche Säule, 27  
 Vorschaupuffer, 275

**W**

Wahrscheinlichkeit  
     Bigramm-, 231  
     Transitions-, 231  
 Walsh-Code, 636  
 Wechselwegnahme, 86  
 Welle

elektromagnetische, 56, 605  
Wiederholungscode, 356  
Wiegeproblem, 211  
Winkeralphabet, 79  
Wörterbuch, 239  
WpM, 202

## Z

Zahl

Fibonacci-, 302  
nichtnegative, 82  
positive, 82  
Zahlencode, 186  
Zeichencode, 18, 181  
Zeichensatz, 184  
Zeigertelegraf, 32  
Zeilenvektor, 136  
Zeitmultiplexverfahren, 77, 593  
synchrones, 593

Zellatmung, 592  
Zelle  
    Mobilfunk-, 592  
Zstandard, 261  
Zufallsfolge, 584, **609**  
Zyklenschreibweise  
    für Permutationen, 95  
Zyklische Redundanzprüfung, 72  
Zyklicher Code, 192, **377**  
Zyklus, 95

$\varphi_g(\bar{g})$

Dirk W. Hoffmann

# Grenzen der Mathematik

Eine Reise durch die Kerngebiete  
der mathematischen Logik

*3. Auflage*

 Springer Spektrum

**Jetzt bestellen:**

[link.springer.com/978-3-662-56616-9](http://link.springer.com/978-3-662-56616-9)

