Digitale Codierungen | DigCod

Zusammenfassung

INHALTSVERZEICHNIS

1. VORWORT

Credit where credit is due [1]

2. STELLENWERTSYSTEM

Begriff	*Definition*
Basis (Radix)	$E \in \mathbb{N}, R \geq 2$
Ziffernmenge	$T_R = \{0, 1, ..., R-1\}$
Darstellung einer Zahl (Polynom)	$N_R = \sum_{i=0}^{n} d_i R^i$
Stellenwertigkeit an der Position i	$d_i \in Z_R, R^i$

Beispiele

Dezimalsystem – $R = 10$ – $Z_{10} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ – $110_{10} = 0\text{d}110$	Oktalsystem – $R = 8$ – $Z_{10} = \{0, 1, 2, 3, 4, 5, 6, 7\}$ – $110_8 = 0\text{o}110 = 72_{10}$
Dualsystem – $R = 2$ – $Z_{10} = \{0, 1\}$ – $0110_2 = 0\text{b}0110 = 6_{10}$	Hexadezimalsystem – $R = 16$ – $Z_{10} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$ – $2\text{D}_{16} = 0\text{x}2\text{D} = 45_{10}$

***Beobachtungen** 1:*

Das Stellenwertsystem basiert auf der Position der Ziffern und verwendet eine Basis (wie 10), während das römische System feste Symbole für bestimmte Werte verwendet, ohne eine Basis oder Stellenwerte zu berücksichtigen. Die Position dient dort für die Entscheidung der Addition oder Subtraktion der Zahl.

2.1. KONVERSION

Dezimal- zu Dualsystem:
– Rechts shift ist eine Division durch 2
– Es entsteht nur dann ein Rest, wenn die Bitstelle $2^0 = 1$ ist.

Beispiel: 25_{10}
Resultat: $0001\ 1001_2$

Zähler	*Nenner*	*Resultat*	*Rest*	
25	2	12	**1**	
12	2	6	**0**	
6	2	3	**0**	↑
3	2	1	**1**	
1	2	0	**1**	

Dual- zu Dezimalsystem bei Nachkommastellen:
– Links shift ist eine Multiplikation mit 2

Beispiel: $0.187'5_{10}$
Resultat: $0.001\ 1_2$

a	*b*	*Resultat*	*Rest*	
0.1875	2	0.375	**0**	↓
0.375	2	0.75	**0**	
0.75	2	1.5	**1**	
0.5	2	1	**1**	

2.2. NACHKOMMASTELLEN

Erweiterung der Darstellung einer Zahl: $N_R = d_n R^n + ... + d_{10} R^0 + d_{-1} R^{-1} + ... + d_{-m} R^{-m}$

Beispiel: $(101.01)_2 = 1 * 2^2 + 0 * 2^1 + 1 * 2^0 + 0 * 2^{-1} + 1 * 2^{-2} = 4 + 1 + {}^1/_4 = 5.25_{10}$

2.3. SUBTRAKTION DURCH ADDITION

Beispiel: $620_{10} - 247_{10}$

1) Komplement bestimmen: $\underbrace{1000}_{\text{Frei wählbar}} - 247 = 753$
2) $620 - 247 = 620 + 753 = 1373 \equiv 373 \bmod 1000$

3. DUALSYSTEM

3.1. ARITHMETIK

3.1.1. Komplementbildung (Zweierkomplement)

1) Rechnen in n Bit $\Rightarrow \bmod 2^n$
   – Übertrag aus dem MSB wird verworfen (gerechnet wird in $\mathbb{Z}_{2^n}$)
2) $-b$ als Zweierkomplement
   – Additives Inverses von $b \bmod 2^n$ ist: $-b \equiv 2^n - b \bmod 2^n$
   – Praktische Berechnung:
     – Bits invertieren: $\sim b$
     – $+1$ addieren
     – $2K(b) = \sim b + 1$

3.1.2. Subtraktion

Subtraktion wird durch Addition des Komplements ersetzt

$$a - b \equiv a + 2K(b) \bmod 2^n$$

Beispiel 1: $13_{10} - 5_{10}$

8 Bit, wobei $13_{10} = 0000\ 1101_2, 5_{10} = 0000\ 0101_2$
Zweierkomplement von 5_{10}:
– invertieren: $1111\ 1010_2$
– $+1 : 1111\ 1011_2$

Addieren: $0000\ 1101_2 + 1111\ 1011_2 = 0001\ 0000\ 1000_2$

MSB-Übertrag weg: $0000\ 1000_2 = 8_{10}$

Also: $13_{10} - 5_{10} = 8_{10}$

3.1.3. Multiplikation

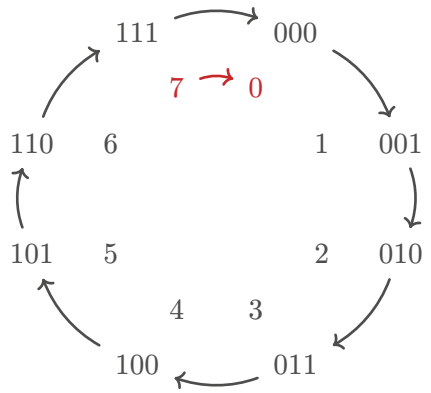
$$
\begin{aligned}
&13 \quad \cdot 9 \\
&11\,01 \cdot 1001 \\
+\ &\quad 1001 \\
+\ &\quad 0000 \\
+\ &\quad 1001 \\
+\ &1\,001 \\
=\ &1\,110101 \\
=\ &\quad 117
\end{aligned}
$$

3.1.4. Division

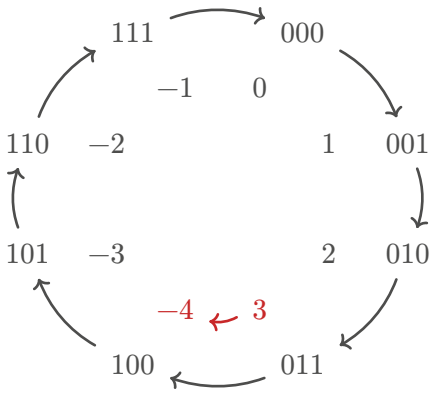
$$
\begin{aligned}
&1000010 \div 11 = 010110 \\
-\ &0 \\
&100 \\
-\ &11 \\
&10 \\
-\ &0 \\
&100 \\
-\ &11 \\
&11 \\
-\ &11 \\
&00 \\
-\ &0 \\
&0
\end{aligned}
$$

3.2. WERTEBEREICH

Graphische Veranschaulichung für Wortbreite von 3 Bit



Unsigned: Überlauf von 7 auf 0



Signed: Überlauf von 3 auf -4

3.3. BIT

Begriff	***Definition***
set bit (gesetztes Bit)	1
cleared bit (gelöschtes Bit)	0
LSB	Least Significant Bit (Bit 0)

Begriff	Definition
MSB	Most Significant Bit (Bit $n-1$) in einer n-stelligen Binärzahl
Nibble	Binärzahl mit 4 Bit
Oktett	Binärzahl mit 8 Bit
Byte	Oktett

3.4. PRÄFIXE

Binär	Näherung	Binärpräfix	Dezimal	Dezimalpräfix
2^{10}	$1.024 \cdot 10^3$	Ki - Kibi	10^3	K - Kilo
2^{20}	$1.049 \cdot 10^6$	Mi - Mebi	10^6	M - Mega
2^{30}	$1.074 \cdot 10^9$	Gi - Gibi	10^9	G - Giga
2^{40}	$1.100 \cdot 10^{12}$	Ti - Tebi	10^{12}	T - Tera
2^{50}	$1.126 \cdot 10^{15}$	Pi - Pebi	10^{15}	P - Peta
2^{60}	$1.153 \cdot 10^{18}$	Ei - Exbi	10^{18}	E - Exa

3.4.1. Multiplikation/Division

1) Umformen in Potenzschreibweise
2) Addition der Exponenten
3) Umformen in Präfixschreibweise

Beispiel: $128K \cdot 64M = 2^7 \cdot 2^{10} \cdot 2^6 \cdot 2^{20} = 2^{17+26} = 2^{43} = 2^3 \cdot 2^{40} = 8T$

4. CODIERUNGEN

Erst wenn man die Codierung kennt, kann man Daten richtig interpretieren.

Definition 1: **Codierung**

Σ und Π seien zwei endliche Mengen von Grundsymbolen. Unter einer ***Codierung*** verstehen wir eine injektive Abbildung der Form

$$c : \Sigma^+ \to \Pi^+$$

Σ ist das ***Quellenalphabet*** und Π das ***Codealphabet*** von c.

In diesem Modul befassen wir uns hauptsächlich mit binären Codierungen (Codierungen mit dem Codealphabet $\Pi = \{0, 1\}$).

4.1. VORZEICHEN

Jede Codierung optimiert eine andere Eigenschaft

– Betrag mit Vorzeichen: Intuition
– Einerkomplement: Einfache Negation
– Zweierkomplement: Arithmetische Effizienz
– Exzess: Vergleichbarkeit/Sortierung

Binär	Betrag mit V.	EinerK.	ZweierK.	$C_{ex,-8,4}$
0000	0	0	0	-8
0001	1	1	1	-7
0010	2	2	2	-6
0011	3	3	3	-5
0100	4	4	4	-4
0101	5	5	5	-3
0110	6	6	6	-2
0111	7	7	7	-1
1000	0	-7	-8	0
1001	-1	-6	-7	1
1010	-2	-5	-6	2
1011	-3	-4	-5	3
1100	-4	-3	-4	4
1101	-5	-2	-3	5
1110	-6	-1	-2	6
1111	-7	0	-1	7

4.1.1. Betrag mit Vorzeichen

Erstes Bit signalisiert, ob Zahl positiv (0) oder Negativ (1) ist.

Problem: Bekannte Rechenregeln funktionieren nicht.

$$\begin{aligned} -19_{10} \quad &+ 1_{10} \quad &&= -18_{10} \\ 1001\ 0011_2 &+ 0000\ 0001_2 = 1001\ 0100_2 &&= -20_{10} \end{aligned}$$

Beispiel 2: **Codierung Betrag mit Vorzeichen**

Gegeben	Dezimal → Binär	Ergebnis
$x = -3_{10}$	$3_{10} = x011_2$	$-3_{10} = 1011_2$

Beispiel 3: **Decodierung Betrag mit Vorzeichen**

Gegeben	Binär → Dezimal	Ergebnis
$x = 1001_2$	$x001_2 = 1_{10}$	$1001_2 = -1_{10}$

4.1.2. (b-1) Komplement / Einerkomplement

Von allen bits wird das Komplement gebildet.

Problem: $0000\ 00000_2 = 1111\ 1111_2 = 0_{10}$

Beispiel 4: **Codierung (b-1) Komplement**

Gegeben	Dezimal → Binär	Komplement	Ergebnis
$x = -5_{10}$	$5_{10} = 0101_2$	$0101_2 \Rightarrow 1010_2$	$-5_{10} = 1010_2$

Beispiel 5: **Decodierung (b-1) Komplement**

Gegeben	Komplement	Binär → Dezimal	Ergebnis
$x = 1011_2$	$1011_2 \Rightarrow 0100_2$	$0100_2 = 4_{10}$	$1011_2 = -4_{10}$

4.1.3. (b) Komplement / Zweierkomplement

Nach der Komplementbildung wird 1 addiert.

Beispiel 6: **Codierung (b) Komplement**

Gegeben	Dezimal → Binär	Komplement	+1	Ergebnis
$x = -5_{10}$	$5_{10} = 0101_2$	$0101_2 \Rightarrow 1010_2$	$0101_2 \Rightarrow 1011_2$	$-5_{10} = 1011_2$

Beispiel 7: **Decodierung (b) Komplement**

Gegeben	−1	Komplement	Binär → Dezimal	Ergebnis
$x = 1111_2$	$1111_2 \Rightarrow 1110_2$	$1110_2 \Rightarrow 0001_2$	$0001_2 = 1_{10}$	$1111_2 = -1_{10}$

4.1.4. Exzess-Codierung (Bias-Code)

Darstellung vorzeichenbehafteter Zahlen durch **Verschiebung des Wertebereichs**.

Statt negativer Zahlen wird ein **Offset (Bias)** addiert

Codierung	Verschiebung	Code							
		000	001	010	011	100	101	110	111
Exzess-0	0	0	1	2	3	4	5	6	7
Exzess-1	1	−1	0	1	2	3	4	5	6
Exzess-2	2	−2	−1	0	1	2	3	4	5
Exzess-3	3	−3	−2	−1	0	1	2	3	4
Exzess-4	4	−4	−3	−2	−1	0	1	2	3

Definition 2: $C_{ex,-B,n}(x)$

C_{ex} = Exzess-Codierung
$-B$ = Bias (negativer Überhang zur 0)
n = Länge der binären Schreibweise
x = zu codierende Zahl

Gespeichert wird: $c = x + B$

Decodierung: $x = c - B$

Gegeben	Codierung	Decodierung
Basis $b = 2$ Wortbreite n Bias B typischerweise: $B = 2^{n-1} - 1$	$C_{ex,-B,n}(x) = x + B$ mit $x \in [-B, 2^n - 1 - B]$	$x = c - B$

Beispiel 8: **Codierung Bias-Code**

Gegeben

Wortbreite: $n = 8$ Bit
Bias: $B = 2^7 - 1 = 127$
Zu codierende Zahl: $x = -5$

Bias addieren

$c = x + B = -5 + 127 = 122$

Dezimal → Binär

$122_{10} = 0111\ 1010_2$

Ergebnis

$C_{ex,-127,8}(-5) = 0111\ 1010_2$

Beispiel 9: **Decodierung Bias-Code**

Gegeben

Wortbreite: $n = 8$ Bit
Bias: $B = 2^7 - 1 = 127$
Bitmuster: $C_{ex,-127,8} = 0111\ 1010_2$

Binär → Dezimal

$0111\ 1010_2 = 122_{10}$

Bias subtrahieren

$x = c - B = 122 - 127 = -5$

Ergebnis

$0111\ 1010_2 \Rightarrow -5_{10}$

4.2. GLEITKOMMAZAHLEN

4.2.1. Fixkommazahl

Skalierte Ganzzahl

Pro	Contra
Einfache Implementierung	Fester Wertebereich
Deterministische Genauigkeit	Feste Auflösung
Keine Exponentendarstellung	Überlauf bei grossen Zahlen
Schnelle Berechnung	Ungeeignet für stark unterschiedliche Grössenordnungen

Definition 3: $C_{FK,k,n}(x) = x \cdot 2^k$

C_{FK} = Fixkomma-Codierung
k = Anzahl Nachkommabits
n = Länge der binären Schreibweise - daraus ergibt sich die Anzahl der Vorkommastellen: $n - k$
x = zu codierende Zahl
I = Ganzzahl

Beispiel 10: Codierung Fixkommazahl

Gegeben

Wortbreite: $n = 8$ Bit
Nachkommabits: $k = 4$
Zu codierende Zahl: $x = 3.25$

Skalieren

$I = x \cdot 2^k = 3.25 \cdot 16 = 52$

Dezimal → Binär

$52_{10} = 0011\ 0100_2$

Ergebnis

$C_{FK,4,8}(3.25) = 0011\ 0100_2$

Beispiel 11: Decodierung Fixkommazahl

Gegeben

Wortbreite: $n = 8$ Bit
Nachkommabits: $k = 4$
Bitmuster: $C_{FK,4,8}(3.25)$

Binär → Dezimal

$0011\ 0100_2 = 52_{10} = I$

Skalieren

$x = {}^{I}/_{2^k} = {}^{52}/_{16} = 3.25$

Ergebnis

$0011\ 0100_2 \Rightarrow 3.25_{10}$

4.2.2. Allgemeiner Wertebereich

Wertebereich bei n Bit und k Nachkommabits

	Ganzzahlbereich	Reeller Bereich
unsigned	$I \in [0, 2^n - 1]$	$x \in \left[0, \frac{2^n - 1}{2^k}\right]$
signed	$I \in [-2^{n-1}, 2^{n-1} - 1]$	$x \in \left[-\frac{2^{n-1}}{2^k}, \frac{2^{n-1} - 1}{2^k}\right]$

Beispiel *12:*

Gegeben

Wortbreite: $n = 8$ Bit
Nachkommabits: $k = 3$
Darstellung im Zweierkomplement (= inkl. Vorzeichen)

Ganzzahlbereich

$I \in [-2^{8-1}, 2^{8-1} - 1] = [-128, 127]$

Reeller Bereich

$$x = \frac{I}{2^k} = \frac{I}{8}$$
$$x_{\min} = \frac{-128}{8} = -17$$
$$x_{\max} = \frac{127}{8} = 15.875$$

Ergebnis

$x \in [-16, 15.875]$

4.2.3. Auflösung

Kleinster darstellbarer Schritt: $\Delta x = 2^{-k}$

Absoluter Fehler: $E_{\text{abs}} = |x_{\text{korrekt}} - x_{\text{gerundet}}|$

Relativer Fehler: $E_{\text{rel}} = {}^{|x_{\text{korrekt}} - x_{\text{gerundet}}|}\!/_{x_{\text{korrekt}}}$

4.2.4. Arithmetik

Addition, Subtraktion und Zweierkomplement sind wie bei Ganzzahlen

– Addiert man zwei Fixkommazahlen $z_0 + z_1$ mit $k_0 > k_1$, so muss z_1 um $k_0 - k_1$ Bits nach rechts geschoben werden.

Multiplikation und Division verschieben das Komma

– Multipliziert man zwei Fixkommazahlen $z_0 \cdot z_1 = z_2$, dann ist $k_2 = k_0 + k_1$

k muss für jede Zahl berücksichtigt werden

– Wird von den meisten Programmiersprachen kaum unterstützt
– Limitiert die Zahlen auf Bereiche, die zur Compilezeit bekannt sind
– unflexibel und fehleranfällig, aber performant

4.3. GLEITKOMMA

Definition 4: $\pm m \cdot 2^e$

$m =$ Mantisse (Signifikand)
$e =$ Exponent

$\leftarrow 32$ Bit $\rightarrow$		
$\pm$	$8-$Bit Exponent	$23-$Bit Mantisse

$$\pm(1 + \text{Mantisse}) \cdot 2^{\text{Exponent} - 127}$$

Der Exponent wird im Exzessformat gespeichert.

Beispiel 13: **Codierung Gleitkommazahl**

Gegeben

Zu codierende Zahl: $x = -42.625$

Vorzeichenbit bestimmen

x ist negativ, somit eine 1

Mantisse Dezimal $\rightarrow$ Binär

$\underbrace{101010}_{42}.\underbrace{101}_{0.625}$

Komma verschieben

$101010.101_2 \cdot 2^0 = 1.01010101_2 \cdot 2^5$

Runden

Bei Überfluss: falls vorherige Bit eine 1 ist, aufrunden, ansonsten kürzen (in diesem Beispiel nicht nötig).

Bias addieren

$5 + 127 = 132$

Exponent Dezimal $\rightarrow$ Binär

$132_{10} \Rightarrow 1000\ 0100_2$

Zusammensetzen

$1 \mid 1000\ 0100 \mid (1)010\ 1010\ 1000\ 0000\ 0000\ 0000 \Rightarrow 1100\ 0010\ 0010\ 1010\ 1000\ 0000\ 0000\ 0000_2$

Beispiel 14: **Decodierung Gleitkommazahl**

Gegeben

Bitmuster: $0100\ 0001\ 0011\ 0110\ 0000\ 0000\ 0000\ 0000_2$

Komponenten extrahieren

$0 \mid 1000\ 0010 \mid (1)011\ 0110\ 0000\ 0000\ 0000\ 0000$

Erstes Bit

$0 =$ Zahl ist positiv

Exponent Binär → Dezimal

$1000\ 0010_2 \Rightarrow 130_{10}$

Bias subtrahieren

$130 - 127 = 3$

Mantisse Binär → Dezimal

$011\ 0110\ 0000\ 0000\ 0000\ 0000_2 \Rightarrow 0.40625_{10}$

Zusammensetzen

$+1 \cdot (1 + 0.40625) \cdot 2^3 = 11.25$

4.3.1. Sonderfälle

Fall	*Vorzeichenbit*	*Exponent*	*Mantisse*	*Beispielwert*
Positive Null	0	$0000\ 0000$	$000\ 0000\ 0000\ 0000\ 0000\ 0000$	$+0$
Negative Null	1	$0000\ 0000$	$000\ 0000\ 0000\ 0000\ 0000\ 0000$	-0
Positive Unendlichkeit	0	$1111\ 1111$	$000\ 0000\ 0000\ 0000\ 0000\ 0000$	$+\infty$
Negative Unendlichkeit	1	$1111\ 1111$	$000\ 0000\ 0000\ 0000\ 0000\ 0000$	$-\infty$
NaN	0 oder 1	$1111\ 1111$	mindestens ein Bit 1	Nicht darstellbare Werte
Subnormale Zahlen	0 oder 1	$0000\ 0000$	mindestens ein Bit 1	$\approx 1.4 \cdot 10^{-45}$ (positiv)
Normalisierte Zahlen	0 oder 1	alles ausser $0000\ 0000$ und $1111\ 1111$	beliebig	Alle regulären Werte

4.3.2. Addition

1) Hidden Bit ergänzen
2) Exponenten vergleichen
   – Wenn unterschiedlich: Bei kleinerer Zahl Mantisse nach rechts schieben
3) Vorzeichen berücksichtigen
   – Gleiche Vorzeichen: Addieren
   – Unterschiedliche Vorzeichen: Subtrahieren
4) Addition/Subtraktion der Signifikanden
5) Falls Carry = 1: Normalisieren
   – Erhöhe Exponent um 1
   – Schiebe Signifikand um 1 nach rechts
6) Hidden Bit entfernen

Beispiel 15: **Gleitkomma addition:** $x = 1.5, y = 0.75$

1) Hidden Bit ergänzen

$$x' = 0 \mid 0111\ 1111 \mid (1)100\ 0000\ 0000\ 0000\ 0000\ 0000$$
$$y' = 0 \mid 0111\ 1110 \mid (1)100\ 0000\ 0000\ 0000\ 0000\ 0000$$

1) Exponent verschieben bei y'

$$x'' = 0 \mid 0111\ 1111 \mid (1)100\ 0000\ 0000\ 0000\ 0000\ 0000$$
$$y'' = 0 \mid 0111\ 1111 \mid (0)110\ 0000\ 0000\ 0000\ 0000\ 0000$$

1) Vorzeichen: beide positiv → Addition
2) Addition $z'' = x'' + y''$

$$z'' = 0 \mid 0111\ 1111 \mid (10)010\ 0000\ 0000\ 0000\ 0000\ 0000$$

1) Carry = 1: Normalisieren

$$z' = 0 \mid 1000\ 0000 \mid (1)001\ 0000\ 0000\ 0000\ 0000\ 0000$$

1) Hidden Bit entfernen

$$z = 0 \mid 1000\ 0000 \mid 001\ 0000\ 0000\ 0000\ 0000\ 0000$$

4.3.3. Präzision

Viele Zahlen können nicht präzise dargestellt werden

$0.1_{10} = 0.0001100110011...$
– Abbruch nach 23 Bits
– $0.1 + 0.2 \neq 0.3$

4.3.4. IEEE 754 - Single vs. Double

Format	*Bits*	*Vorzeichen*	*Exponent*	*Mantisse*	*Bias*
Single	32	1	8	23	−127
Double	64	1	11	52	−1023

Grösserer Exponent führt zu grösserem Wertebereich. Grössere Mantisse ermöglicht höhere Präzision.
– Single Precision: $\varepsilon \approx 2^{-23} \approx 10^{-7}$
  – ≈ 7 signifikante Dezimalstellen
– Double Precision: $\varepsilon \approx 2^{-52} \approx 10^{-16}$
  – ≈ 16 signifikante Dezimalstellen

4.4. ASCII

American Standard Code for Information Interchange

Grösse des Zeichensatzes: $2^7 = 128 \rightarrow 7\ \text{Bit}$. 8-Bit-ASCII sind Extended ASCII-Varianten und nicht standardisiert, sondern Codepages, bspw.: ISO-8859-1 (Latin-1). ASCII Enthält druckbare (darstellbare) Zeichen und (nicht darstellbare) Steuerzeichen (0x00=NUL, 0x07=BEL, …).

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	**NUL** 00	SOH 01	STX 02	ETX 03	EOT 04	ENQ 05	ACK 06	BEL 07	**BS** 08	**TAB** 09	**LF** 0A	VT 0B	FF 0C	**CR** 0D	SO 0E	SI 0F
1	DLE 10	DC1 11	DC2 12	DC3 13	DC4 14	NAK 15	SYN 16	ETB 17	CAN 18	EM 19	SB 1A	ESC 1B	FS 1C	GS 1D	RS 1E	US 1F
2	**SP** 20	**!** 21	**“** 22	**#** 23	**$** 24	**%** 25	**&** 26	**‘** 27	**(** 28	**)** 29	***** 2A	**+** 2B	**,** 2C	**–** 2D	**.** 2E	**/** 2F
3	**0** 30	**1** 31	**2** 32	**3** 33	**4** 34	**5** 35	**6** 36	**7** 37	**8** 38	**9** 39	**:** 3A	**;** 3B	**<** 3C	**=** 3D	**>** 3E	**?** 3F
4	**@** 40	**A** 41	**B** 42	**C** 43	**D** 44	**E** 45	**F** 46	**G** 47	**H** 48	**I** 49	**J** 4A	**K** 4B	**L** 4C	**M** 4D	**N** 4E	**O** 4F
5	**P** 50	**Q** 51	**R** 52	**S** 53	**T** 54	**U** 55	**V** 56	**W** 57	**X** 58	**Y** 59	**Z** 5A	**[** 5B	**** 5C	**]** 5D	**^** 5E	**_** 5F
6	**`** 60	**a** 61	**b** 62	**c** 63	**d** 64	**e** 65	**f** 66	**g** 67	**h** 68	**i** 69	**j** 6A	**k** 6B	**l** 6C	**m** 6D	**n** 6E	**o** 6F
7	**p** 70	**q** 71	**r** 72	**s** 73	**t** 74	**u** 75	**v** 76	**w** 77	**x** 78	**y** 79	**z** 7A	**{** 7B	**|** 7C	**}** 7D	**~** 7E	**DEL** 7F

4.5. UNICODE

Globale Zeichennummerierung

– ASCII-kompatibel
– 1 bis 4 Byte pro Zeichen
– Häufige (westliche) Zeichen kurz, seltene (westliche) Zeichen länger
– Das erste Byte verrät, wie viele Bytes folgen
– Unicode = Codepoint, UTF-8 = Bytecodierung

4.5.1. Struktur

Codepoint-Bereich	*Bytes*	*Bitmuster*
U+0000 - U+007F	1	0xxxxxxx
U+0080 - U+07FF	2	110xxxxx 10xxxxxx
U+0800 - U+FFFF	3	1110xxxx 10xxxxxx 10xxxxxx
U+10000 - U+10FFFF	4	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

– Erstes Byte zeigt Länge
– Folgebytes beginnen mit 10
– ASCII-Zeichen (0-127) bleiben unverändert

4.5.2. Codierung

Gegeben: Unicode-Codepoint U

1) Bestimme den Wertebereich → Anzahl Bytes.
2) Schreibe U in Binärdarstellung.
3) Fülle die Bits in die x-Positionen des passenden Musters ein.
4) Wandle in Hex um.

***Beispiel** 16:*

Gegeben

Zeichen: ä

Unicode

$U = \mathrm{E4}_{16} = 228_{10} = 1110\ 0100_2$

Benötigt 8 Bit $\rightarrow$ 2 Bytes

Muster einfügen

`110xxxxx 10xxxxxx` $\Rightarrow$
`11000011 10100100` $\Rightarrow 1100\ 0011\ 1010\ 0100_2 = \mathrm{C3}\ \mathrm{A4}_{16}$

UTF-8(ä) = $\mathrm{C3}\ \mathrm{A4}_{16}$

4.5.3. Decodierung

Gegeben: Bytefolge

1) Lies das erste Byte.
2) Bestimme anhand der führenden Bits die Länge.
3) Entferne die Strukturpräfixe (110, 10, etc.).
4) Füge die restlichen Bits zusammen.
5) Interpretiere als Codepoint.

***Beispiel** 17:*

Gegeben

$1110\ 0010\ 1000\ 0010\ 1010\ 1100_2$

Muster entfernen

`11100010 10000010 10101100`
`1110xxxx 10xxxxxx 10xxxxxx` $\Rightarrow$
`    0010   000010   101100` $\Rightarrow 0010\ 0000\ 1010\ 1100_2 = 20\ \mathrm{AC}_{16} = €$

4.5.4. Encodings

UTF-16

– Meist 2 Bytes pro Zeichen
– Zeichen ausserhalb der BMP (Basic Multilingual Pane) benötigen in UTF-16 sogenannte Surrogate Pairs (4 Bytes)
– Nicht ASCII-kompatibel

Vorteil: Häufig effizient für asiatische Sprachen

Nachteil: Komplexere Handhabung

UTF-32

– Immer 4 Bytes pro Zeichen
– Direkter Zugriff auf Codepoints
– Sehr speicherintensiv

Encoding	Länge	Vorteil	Nachteil
UTF-8	1-4 Byte	ASCII-Kompatibel, effizient	Variable Länge
UTF-16	2-4 Byte	Teils effizient	Surrogate-Komplexität
UTF-32	4 Byte	Sehr einfach	Hoher Speicherbedarf

4.5.5. Basic Multilingual Plane (BMP)

Unicode ist als Zahlenraum definiert: $0 \leq U \leq 10\ \mathrm{FF}\ \mathrm{FF}_{16}$

Das sind $1'114'111_{10}$ mögliche Codepoints. Dieser Raum ist in Planes (Ebenen) aufgeteilt.

Die BMP umfasst $U + 0000$ bis $U + \mathrm{FFFF}$, also: $0 \leq U \leq 65535$. Das sind genau 16 Bit.

In der BMP liegen ASCII, Lateinische Schriftzeichen, Griechisch, Viele asiatische Zeichen, Mathematische Symbole etc.

5. BOOLSCHE LOGIK

– Eine Aussage ist entweder wahr (1) oder falsch (0)
– Aussagen können logisch verknüpft werden

($x \wedge y$ schreibt man auch als xy oder $x \cdot y$ oder $x \cap y$, $x \vee y$ als $x + y$ oder $x \cup y$)

x	y	$x \wedge y$	$x \vee y$	$x \oplus y$	$\neg x$
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Dualitätsprinzip: Ersetze in einer wahren Gleichung $\wedge \leftrightarrow \vee$ und $0 \leftrightarrow 1$, Ergebnis bleibt wahr.

5.1. RECHENGESETZE

$(A \Rightarrow B) \Leftrightarrow (\neg B \Rightarrow \neg A)$ $(A \Leftrightarrow B) \Leftrightarrow (A \wedge B) \vee (\neg A \wedge \neg B)$ $A \vee (\neg A \wedge B) \Leftrightarrow A \vee B$

$(A \Rightarrow B) \Leftrightarrow (\neg A \vee B)$ $\neg(A \Rightarrow B) \Leftrightarrow A \wedge \neg B$ $(A \Rightarrow B \Rightarrow C) \Leftrightarrow (A \Rightarrow B) \wedge (B \Rightarrow C)$

Begriff	Definition
Abtrennungsregel	$(A \wedge (A \Rightarrow B)) \Rightarrow B$
Kommutativität	$(A \wedge B) \Leftrightarrow (B \wedge A)$ $(A \vee B) \Leftrightarrow (B \vee A)$
Assoziativität	$A \wedge (B \wedge C) \Leftrightarrow (A \wedge B) \wedge C$ $A \vee (B \vee C) \Leftrightarrow (A \vee B) \vee C$
Distributivität	$A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$ $A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$
Absorption	$A \vee (A \wedge B) \Leftrightarrow A$ $A \wedge (A \vee B) \Leftrightarrow A$
Idempotenz	$A \vee A = A$ $A \wedge A = A$

Begriff	Definition
Doppelte Negation	$\neg(\neg A) \Leftrightarrow \neg\neg A \Leftrightarrow A$
Konstanten	$W = \text{wahr}$ $F = \text{falsch}$
de Morgan	$\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B$ $\neg(A \vee B) \Leftrightarrow \neg A \wedge \neg B$

5.2. TERME

Begriff	Definition
Literal	Variable oder Negation einer Variablen: x_3 (positiver Literal), $\neg x_3$ oder $\overline{x_3}$ (negatives Literal)
Konjunktionsterm	Konjunktion von Literalen: $\overline{x_1}x_3x_4 = \overline{x_1} \wedge x_3 \wedge x_4$
Disjunktionsterm	Disjunktion von Literalen: $\overline{x_1} \vee x_3 \vee x_4$
Minterm	Konjunktionsterm, der alle Parameter der Funktion enthält: $x_0\overline{x_1}\overline{x_2}x_3x_4$
Maxterm	Disjunktionsterm, der alle Parameter der Funktion enthält: $x_0 \vee \overline{x_1} \vee \overline{x_2} \vee x_3 \vee x_4$

5.3. NORMALFORMEN

Standardisierte Schreibweise. Vorteile:

– aus Wahrheitstabelle konstruierbar
– Vereinfachung systematisch möglich
– Umsetzung als Schaltung (AND-OR-Structure) direkt ableitbar

5.3.1. Disjunktive Normalform

in Term ist in DNF, wenn er eine ODER-Verknüpfung von UND-Verknüpfungen ist, z.B.:

$$f(x,y,z) = (\neg x \wedge y) \vee (x \wedge z) = \neg x \wedge y \vee x \wedge z$$

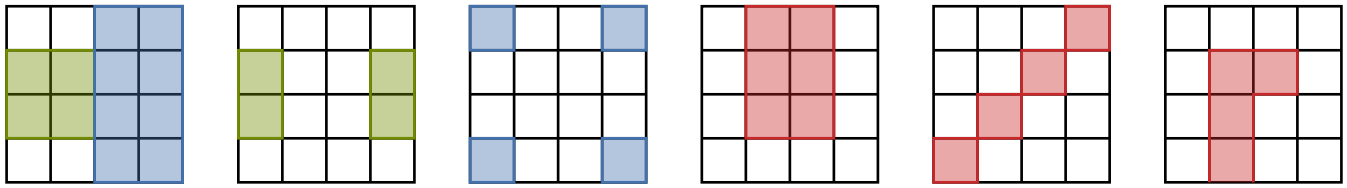
Da $\wedge$ eine höhere Bindungsstärke als $\vee$ hat, sind die Klammern nicht zwingend notwendig. Die kanonische DNF (KDNF) ist die Disjunktion der Minterme.

5.3.2. KV-Diagramm

Je grösser die Blöcke, desto einfacher wird das Ergebnis. Dabei müssen aber bestimmte Regeln eingehalten werden:

– Die Blöcke müssen immer rechteckig sein und
– die Grösse einer Zweierpotenz haben, also 2, 4, 8, 16, 32, …
– Die Blöcke können auch über den Rand hinaus gehen und mit der gegenüberliegenden Seite verbunden werden,
– Blöcke können sich teilweise überlappen. Das kann sinnvoll sein, wenn dadurch grössere Blöcke entstehen.
– Werte, die sowohl einfach als auch negiert vorkommen, werden gestrichen
– Ein Block wird nur berücksichtigt, wenn seine Einsen nicht vollständig in anderen Blöcken enthalten sind. Andernfalls entsteht ein nichtessentieller Term, der redundant ist, da andere Terme bereits die gleichen Variablenbelegungen abdecken und nicht weiter vereinfacht werden können.

OK **NOK**



5.4. NAND (NOT AND)

Universalbaustein

$x \underbrace{\mid}_{\text{Sheffer stroke}} y = \overline{x \wedge y} = \overline{xy}$

– praktisch (in CMOS schnell sowie einfach aufzubauen)
– funktional vollständig: jede Boolesche Funktion lässt sich nur mit NAND realisieren.

x	y	$x \wedge y$	$x \mid y$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

OP	***impl***
NOT	$\overline{x} = \overline{x \wedge x} = x \mid x$
AND	$x \wedge y = \overline{x \mid y} = (x \mid y) \mid (x \mid y)$
OR	$x \vee y = \overline{\overline{x \vee y}} = \overline{\overline{x} \wedge \overline{y}} = \overline{(x \mid x) \wedge (y \mid y)} = (x \mid x) \mid (y \mid y)$

5.5. VON LOGIK ZUR ARITHMETIK

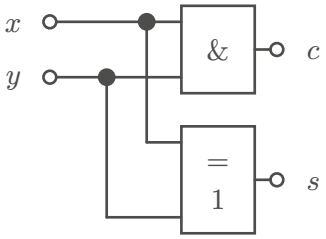
Boolesche Logik realisiert binäre Addition

XOR bildet die Addition zweier Bits ab (im Zahlenraum mit nur 0 und 1, also $\bmod 2$), AND bildet den Übertrag ab
– $s = x \oplus y \rightarrow$ Addition mod 2
– $c = x \wedge y \rightarrow$ Übertrag

x	y	$x + y$	$x \wedge y$	$x \oplus y$
0	0	00	0	0
0	1	01	0	1
1	0	01	0	1
1	1	10	1	0

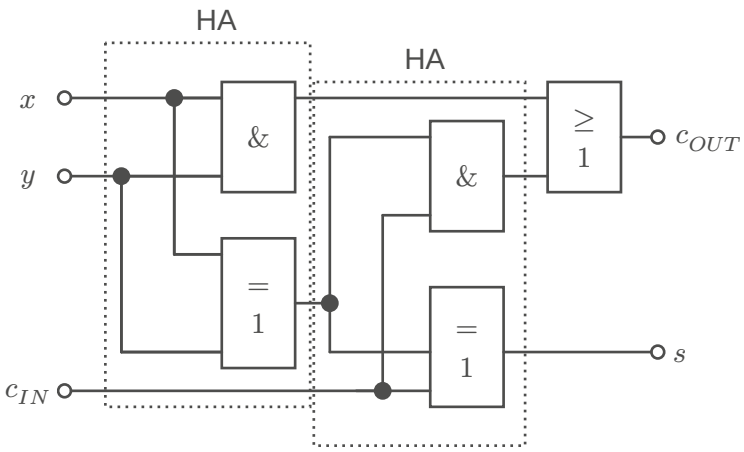
Halbaddierer (half adder)

– Kann 2 einstellige Binärzahlen addieren
– Hat 2 Eingänge (x, y) und 2 Ausgänge (s, c)



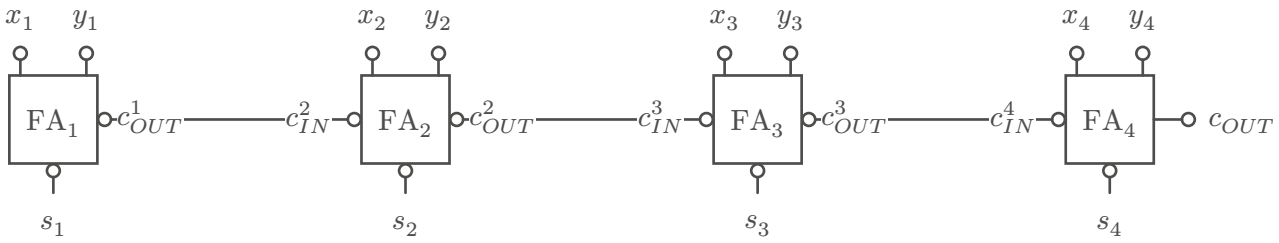
Volladdierer (full adder)

– Hat 3 Eingänge (x, y, c_{IN})



4-Bit Ripple carry adder

– Jedes Bit muss auf das Carry-Bit des letzten Volladierers warten



Carry Look-Ahead adder

– Reduziert propagations-delay durch komplexere hardware

5.5.1. Bitfolge Darstellungen

Darstellung	*Beispiel*
Zahl	$0101_2 = 5_{10}$
Vektor	$\begin{pmatrix}1\\0\\1\end{pmatrix}$
Polynom	$u^2 + u^0 = u^2 + 1$
Tupel	$(1, 0, 1)$
Menge	$\{0, 1\}$
Zustand	Könnte ein Zustand eines Speicherregisters (Adressierung, Anweisung, usw.) sein.

5.5.1.1. Vektor

***Beispiel** 18:* **Bitfolge als Vektor in $\mathbb{Z}_2^n$**

$$v, w \in \mathbb{Z}_2^4, v = \begin{pmatrix}1\\0\\1\\1\end{pmatrix}, w = \begin{pmatrix}1\\1\\0\\1\end{pmatrix}$$

$$v + w = \begin{pmatrix}1\\0\\1\\1\end{pmatrix} + \begin{pmatrix}1\\1\\0\\1\end{pmatrix} \equiv \begin{pmatrix}0\\1\\1\\0\end{pmatrix} \bmod 2$$

5.5.1.2. Polynom

Polynome erlauben

– Verschiebungen elegant zu beschreiben
– Redundanz strukturiert zu erzeugen
– Generatorpolynome (zyklische Codes)
– CRC-Rechnung als Polynomdivision

Beispiel *19:* **Bitfolge als Polynom in $\mathbb{Z}_2$**

$$\begin{aligned}&[1\ 0\ 0\ 1\ 0\ 1]\\&= 1u^5 + 0u^4 + 0u^3 + 1u^2 + 0u^1 + 1u^0\\&= u^5 + u^2 + 1\end{aligned}$$

u ist keine Zahl, u^k beschreibt «Bitposition k»

Polynomaddition in $\mathbb{Z}_2$	**Polynommultiplikation in $\mathbb{Z}_2$**
$(u^3 + u + 1) + (u^3 + u^2)$	$(u^2 + u + 1)(u + 1)$
$= \cancel{2u^3} + u^2 + u + 1$	$= u^3 + \cancel{2u^2} + \cancel{2u} + 1$
$= u^2 + u + 1$	$= u^3 + 1$

5.6. FUN GAMES

NAND Game

nand2tetris

6. KOMBINATORIK

Kombinatorik befasst sich mit der Bestimmung aller möglichen Anordnungen/Kombinationen bestimmter Objekte oder Ereignisse. Laplace-Wahrscheinlichkeit erfordert Berechnung von Anzahlen. Kombinatorik ist die Mathematische Technik hierfür.

	Wiederholung	***Keine Wiederholung***
Anordnung	Permutation mit Wiederholung	Permutation ohne Wiederholung
Geordnete Auswahl	Variation mit Wiederholung	Variation ohne Wiederholung
Ungeordnete Auswahl	Kombination mit Wiederholung	Kombination ohne Wiederholung

Beispiel *20:*

5 Männer und 4 Frauen sollen in einer Reihe sitzen, und zwar so, dass die Frauen auf den geraden Plätzen sitzen. Wie viele solche Anordnungen sind möglich?

$$4! \cdot 5! = 2880$$

6.1. PERMUTATION MIT WIEDERHOLUNG

Anordnung von n Objekten, die **nicht** alle voneinander unterscheidbar sind (s Subsets k mit gleichen Objekten).

$$\frac{n!}{k_1!k_2!...k_s!}$$

Beispiel 21: **Anordnung drei roter und zwei blauer Kugeln**

$$s = 2, n = 5, k_1 = 3, k_2 = 2, \text{Anzahl} = \frac{(5 \cdot 4 \cdot 3 \cdot 2 \cdot 1)}{(3 \cdot 2 \cdot 1)(2 \cdot 1)} = 10$$

6.2. PERMUTATION OHNE WIEDERHOLUNG

Anordnung von n Objekten, die alle unterscheidbar sind.

$$n!$$

Beispiel 22: **Anordnung fünf verschiedenfarbiger Kugeln**

$$\text{Anzahl} = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 5!$$

6.3. VARIATION MIT WIEDERHOLUNG

- Reihenfolge spielt eine Rolle
- Elemente dürfen mehrfach vorkommen

$$n^k$$

Beispiel 23: **PIN-Code mit 4 Ziffern**

```
0000
1234
9876
```

Jede Position hat 10 Möglichkeiten

$$\text{Anzahl} = 10 \cdot 10 \cdot 10 \cdot 10 = 10^4$$

6.4. VARIATION OHNE WIEDERHOLUNG

- Reihenfolge spielt eine Rolle
- Elemente dürfen **nicht** mehrfach vorkommen

$$\frac{n!}{(n-k)!} = \prod_{n}^{n-k+1} n$$

Beispiel 24: **1., 2. und 3. Platz aus 10 Teilnehmern**

$$\text{Anzahl} = 10 \cdot 9 \cdot 8$$

6.5. KOMBINATION MIT WIEDERHOLUNG

- Reihenfolge spielt **keine** Rolle
- Elemente dürfen mehrfach vorkommen

$$\binom{n+k-1}{k} = \frac{(n+k-1)!}{(n-1)! \cdot k!}$$

***Beispiel** 25:* **Es sollen drei von fünf verschiedenfarbigen Kugeln gezogen und wieder zurückgelegt werden.**

$$\text{Anzahl} = \binom{5+3-1}{3} = \binom{7}{3} = 35$$

6.5.1. Kombination ohne Wiederholung

– Reihenfolge spielt **keine** Rolle
– Elemente dürfen **nicht** mehrfach vorkommen

$$\binom{n}{k} = \frac{\frac{n!}{(n-k)!}}{k!} = \frac{n!}{k!(n-k)!} = \frac{\prod_{n}^{n-k+1} n}{k!}$$

***Beispiel** 26:* **Lotto 6 aus 42**

$$\text{Anzahl} = \binom{42}{6}$$

6.6. ÜBERSICHT AUSWAHL

Anzahl A der Möglichkeiten			
n Optionen k Auswählen		Beachtung der Reihenfolge	
		✓	✗
Wiederholung erlaubt	✓	$A = n^k$	$A = \binom{n+k-1}{k}$
	✗	$A = n(n-1)...(n-k+1) = \frac{n!}{(n-k)!}$	$A = \frac{n!}{k!(n-k)!} = \binom{n}{k}$

6.7. HYPERGEOMETRISCHE VERTEILUNG

Die hypergeometrische Verteilung ist ein Modell, das verwendet wird, um die Wahrscheinlichkeit $P(k)$ von k Erfolgen (gewünschten Ergebnissen) aus den K gesamt möglichen Erfolgen aus N Objekten in n Ziehungen zu berechnen.

$$P(k) = \frac{\binom{K}{k} \cdot \binom{N-K}{n-k}}{\binom{N}{n}}$$

***Beispiel** 27:* **Beim Lotto 6 aus 49 die Wahrscheinlichkeiten 3, 4, 5 oder 6 Richtige zu erhalten**

$$K = 6$$
$$N = 49$$
$$n = 6$$
$$P(3) = \frac{\binom{6}{3} \cdot \binom{43}{3}}{\binom{49}{6}} \approx 0.0176504039$$

$$P(4) = \frac{\binom{6}{4} \cdot \binom{43}{2}}{\binom{49}{6}} \approx 0.0009686197$$

$$P(5) = \frac{\binom{6}{5} \cdot \binom{43}{1}}{\binom{49}{6}} \approx 0.0000184499$$

$$P(6) = \frac{\binom{6}{6} \cdot \binom{43}{0}}{\binom{49}{6}} \approx 0.0000000715$$

7. WAHRSCHEINLICHKEIT

In realen Systemen trifft Unsicherheit auf. Diese lässt sich nicht exakt vorhersagen, sondern nur statistisch beschreiben. Dafür verwenden wir ***Wahrscheinlichkeit***.

Begriff	***Definition***
Disjunke Ereignisse	Ereignisse, die sich gegenseitig ausschliessen. Beispiel: z ist gerade oder ungerade
Zufallsvorgang	Ein Vorgang mit mehreren möglichen Ergebnissen, dessen Ausgang nicht sicher vorhergesagt werden kann
Zufallsexperiment	Zufallsvorgang, der geplant ist und kontrolliert ablauft

7.1. ERGEBNISMENGE

Begriff	***Definition***
Ergebnismenge	Die ***Ergebnismenge*** eines Zufallsvorgangs umfasst **alle möglichen Ausgänge** des Experiments. Sie wird mit dem Symbol Ω (Omega) bezeichnet. Die Anzahl aller möglichen Ergebnisse der Ergebnismenge wird mit $\lvert\Omega\rvert$ bezeichnet.
Ergebnis	Ein einzelner möglicher Ausgang $\omega \in \Omega$ heisst ***Ergebnis***
Ereignis	Ein ***Ereignis*** ist eine Teilmenge der Ergebnismenge $A \subset \Omega$

***Beispiel** 28:*

Würfel: $\Omega = \{1, 2, 3, 4, 5, 6\}$

Werfen einer Münze so lange, bis Kopf erscheint: $\Omega = \{K, ZK, ZZK, ZZZK, ...\}$

7.2. EIGENSCHAFTEN

Für jedes Ereignis A gilt $0 \leq P(A) \leq 1$.

Begriff	***Eigenschaft***
Sicheres Ereignis	Die Wahrscheinlichkeit der gesamten Ergebnismenge ist $P(\Omega) = 1$
Unmögliches Ereignis	Die Wahrscheinlichkeit der leeren Menge ist $P(\varnothing) = 0$

Begriff	*Eigenschaft*
Additionsregel	Für zwei Ereignisse (Wahrscheinlichkeit, dass A **oder** B auftritt): $$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$ Für drei Ereignisse: $$P(A \cup B \cup C) = P(A) + P(B) + P(C) - P(A \cap B) - P(A \cap C) - P(C \cap B) + P(A \cap B \cap C)$$ Für **disjunkte** Ereignisse ($A \cap B = \varnothing$): $$P(A \cup B) = P(A) + P(B)$$
Unabhängigkeit	Wahrscheinlichkeit, dass A_1 und A_2 auftritt: $$P(A_1 \cap A_2) = P(A_1) \cdot P(A_2)$$
Gegenereignis	Wird notiert als $\overline{A}$ und hat die Eigenschaft $P\left(\overline{A}\right) = 1 - P(A)$.

7.3. WAHRSCHEINLICHKEITSDEFINITION NACH LAPLACE

Wenn ein Experiment eine Anzahl verschiedener und gleich möglicher Ausgänge hervorbringen kann und einige davon als günstig anzusehen sind, dann ist die Wahrscheinlichkeit eines günstigen Ausgangs gleich dem Verhältnis der Anzahl der günstigen zur Anzahl der möglichen Ausgänge.

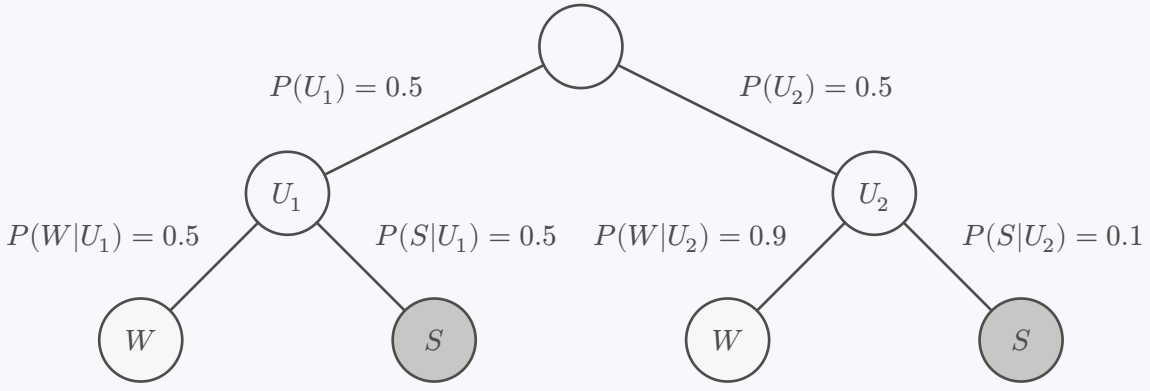
$$P(E) = \frac{\text{Anzahl günstiger Ergebnisse}}{\text{Anzahl aller möglichen Ergebnisse}} = \frac{|E|}{|\Omega|}$$

Dabei gilt:

– Ω: Ergebnismenge (alle möglichen Ergebnisse)
– $E \subseteq \Omega$: betrachtetes Ereignis
– $|E|$: Anzahl günstiger Ergebnisse
– $|\Omega|$: Anzahl aller möglichen Ergebnisse

***Beispiel** 29:* **Urnen**

Es gibt zwei Urnen, U_1 und U_2. Urne U_1 enthält 5 schwarze und 5 weisse Kugeln und Urne U_2 enthält 9 schwarze und 1 weisse Kugel. Die Wahrscheinlichkeit eine Kugel aus U_1 oder U_2 zu ziehen, ist gleich gross (50/50). Wie gross ist nun die Wahrscheinlichkeit eine weisse Kugel zu ziehen?



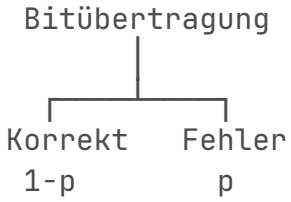
$$P(W|U_1) \wedge P(U_1) \vee P(W|U_2) \wedge P(U_2) = P(W|U_1) \cdot P(U_1) + P(W|U_2) \cdot P(U_2)$$

$$
\begin{aligned}
&= 0.5 \cdot 0.5 + 0.5 \cdot 0.1 \\
&= 0.25 + 0.05 \\
&= 0.3
\end{aligned}
$$

Die Wahrscheinlichkeit beträgt also 30%, eine weisse Kugel zu ziehen unter der Annahme, dass jede Urne mit gleicher Wahrscheinlichkeit gewählt wird.

7.4. BITFEHLER

Ein Bitfehler bezeichnet die inkorrekte Wiedergabe eines Bits während der Datenübertragung oder -speicherung. Das bedeutet, dass ein Bit von 0 zu 1 oder von 1 zu 0 fälschlicherweise geändert wird.



Die Wahrscheinlichkeit, dass ein Datenblock der Grösse n Bits bei einer Fehlerrate von p fehlerhaft ist, kann mit folgender Formel berechnet werden:

$$P(\text{fehlerhaft}) = 1 - (1-p)^n$$

Wahrscheinlichkeit für genau k Fehler in einem Datenblock mit n Bits bei Bitfehlerrate p (Binomialverteilung):

$$P(k) = \binom{n}{k} \cdot p^k \cdot (1-p)^{n-k}$$

***Beispiel** 30:* **Bitfehler**

Gegeben sei eine Bitfehlerrate von 10^{-5}. Wie gross ist die Wahrscheinlichkeit, dass ein Datenblock mit einer Grösse von 150 kbit fehlerhaft ist?

$$P(\text{fehlerhaft}) = 1 - \left(1 - 10^{-5}\right)^{150 \cdot 10^3} \approx 0.77687$$

Was ist die Wahrscheinlichkeit für genau 2 Fehler?

$$P(2) = \binom{150 \cdot 10^3}{2} \cdot \left(10^{-5}\right)^2 \cdot \left(1 - 10^{-5}\right)^{150 \cdot 10^3 - 2} \approx 0.25102$$

***Beispiel** 31:* **Die Grösse eines Datenblocks sei 150 bit. Die Bitfehler-Wahrscheinlichkeit ist 10^{-2}. Wie gross ist die Wahrscheinlichkeit, dass höchstens 3 Fehler auftreten? Wie gross ist die Restfehlerwahrscheinlichkeit für einen Datenblock?**

$$
\begin{aligned}
P(0) &= \binom{150}{0} \cdot (0.99)^{150} && \approx 0.221451 \\
P(1) &= \binom{150}{1} \cdot 0.01 \cdot (0.99)^{149} && \approx 0.335533 \\
P(2) &= \binom{150}{2} \cdot 0.0001 \cdot (0.99)^{148} && \approx 0.252497 \\
P(3) &= \binom{150}{3} \cdot 0.000001 \cdot (0.99)^{147} && \approx 0.125823
\end{aligned}
$$

$$P(0) + \mathbb{P}(1) + \mathbb{P}(2) + \mathbb{P}(3) \approx 0.935304$$

$$P(\text{Restfehler}) \approx 1 - 0.935304 = 0.064696$$

7.5. GESAMTWAHRSCHEINLICHKEIT

Jede mögliche Fehleranordnung hat die Wahrscheinlichkeit

$$p^k(1-p)^{n-k}$$

Es gibt $\binom{n}{k}$ solche Anforderungen. Darum gilt

$$P(X=k) = \overbrace{\binom{n}{k}}^{\text{Anzahl Kombinationen}} \underbrace{p^k}_{k\times \text{ Erfolg}} \overbrace{(1-p)^{n-k}}^{n-k\times \text{ Misserfolg}}$$

***Beispiel** 32:* **Schraubenproduktion**

Bei einer Schraubenproduktion ist jede 10e Schraube fehlerhaft. Was ist die Wahrscheinlichkeit, dass 2 von 3 Schrauben OK sind?

$$P(X=2) = \binom{3}{2} \cdot \left(\frac{9}{10}\right)^2 \cdot \left(1-\frac{9}{10}\right)^{3-2} = \binom{3}{2} \cdot \frac{81}{100} \cdot \frac{1}{10} = 3 \cdot \frac{81}{1000} = 0.243$$

7.6. BINOMIALVERTEILUNG

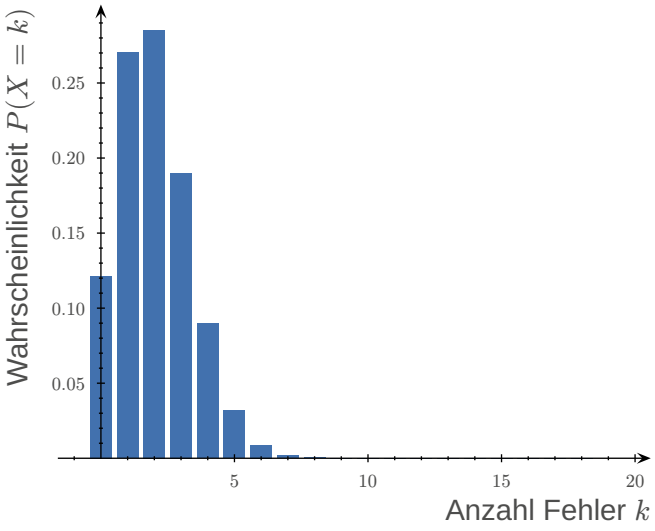
$$E(X) = np$$

Die Binomialverteilung beschreibt, wie wahrscheinlich es ist, dass bei n unabhängigen Versuchen genau k Erfolge auftreten.

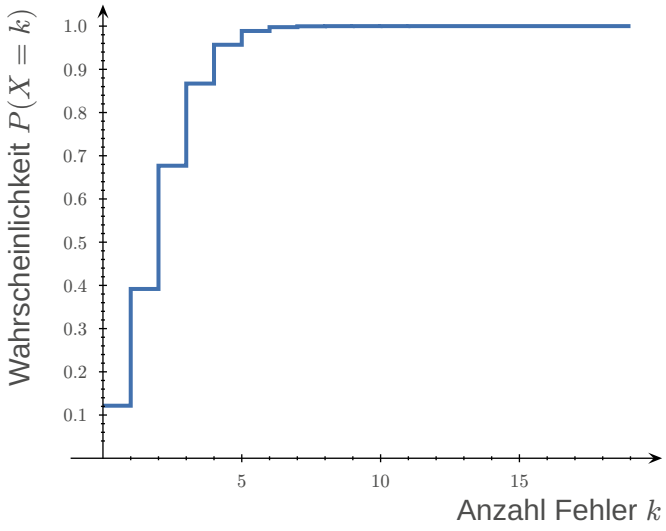
In unserem Kontext bedeutet das:

- n = Anzahl übertragener Bits
- p = Bitfehlerwahrscheinlichkeit
- k = Anzahl Fehler

Binomialverteilung von Bitfehlern
($n = 20$ bits, $p = 0.1$)



Kumulative Binomialverteilung
($n = 20$ bits, $p = 0.1$)



***Beispiel** 33:* **Bitfehler**

Ein Übertragungskanal überträgt Bits (0 oder 1). Die Bitfehlerwahrscheinlichkeit beträgt $p = 0.02$ Die gesendeten Bits sind gleich wahrscheinlich:

$$P(\text{gesendet} = 0) = 0.5$$
$$P(\text{gesendet} = 1) = 0.5$$

Wie gross ist die Wahrscheinlichkeit, dass der Empfänger eine 1 liest?

$$\begin{aligned} P(\text{empfangen} = 1 \mid \text{gesendet} = 1) &= 0.98 \\ P(\text{empfangen} = 1 \mid \text{gesendet} = 0) &= 0.02 \\ P(\text{empfangen} = 1) &= P(\text{empfangen} = 1 \mid \text{gesendet} = 1) \cdot P(\text{gesendet} = 1) \\ &\quad + P(\text{empfangen} = 1 \mid \text{gesendet} = 0) \cdot P(\text{gesendet} = 0) \\ &= 0.98 \cdot 0.5 + 0.02 \cdot 0.5 \\ &= 0.5 \end{aligned}$$

7.7. BEDINGTE WAHRSCHEINLICHKEIT

Wie wahrscheinlich ist Ereignis A, wenn Ereignis B bereits eingetreten ist? Formelle Schreibweise: $P(A|B)$.

Allgemein gilt für die bedingte Wahrscheinlichkeit:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Voraussetzung ist dabei, dass gilt:

$$P(B) > 0$$

Die Formel bedeutet:
– Im Nenner steht die Wahrscheinlichkeit der Bedingung B.
– Im Zähler steht die Wahrscheinlichkeit, dass sowohl A als auch B eintreten.

Man betrachtet also nur noch den Teil des Wahrscheinlichkeitsraums, in dem B gilt, und fragt dann, wie gross darin der Anteil der Fälle ist, in denen zusätzlich A eintritt.

***Beispiel** 34:* **Würfelwurf**

$$\begin{aligned} A &= \text{Die gewürfelte Zahl ist gerade} &&= \{2, 4, 6\} \\ B &= \text{Die gewürfelte Zahl ist grösser als } 3 &&= \{4, 5, 6\} \\ P(A) = P(B) &= \frac{1}{2} \\ A \cap B &= \{4, 6\} \\ P(A \mid B) &= \frac{2}{3} \end{aligned}$$

Die Wahrscheinlichkeit für «gerade Zahl» ist also unter der Bedingung «grösser als 3» nicht mehr $^1/_2$, sondern $^2/_3$.

7.7.1. Zusammenhang mit Multiplikationsregel

Aus der Definition folgt direkt eine wichtige Umformung, nämlich die ***Multiplikationsregel***:

$$P(A \cap B) = P(A|B) \cdot P(B)$$

Ebenso gilt auch:

$$P(A \cap B) = P(B|A) \cdot P(A)$$

7.7.2. Bayes-Theorem

Setzt man die beiden Ausdrücke gleich, erhält man

$$P(A|B) \cdot P(B) = P(B|A) \cdot P(A)$$

Durch Umstellen ergibt sich das ***Bayes-Theorem***:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Diese Formel erlaubt es, Wahrscheinlichkeiten umzukehren: Aus der Wahrscheinlichkeit von B unter der Bedingung A kann die Wahrscheinlichkeit von A unter der Bedingung B berechnet werden.

7.7.2.1. Satz der totalen Wahrscheinlichkeit

Angenommen ein Ereignis B kann durch mehrere verschiedene Ursachen entstehen. Seien

$$A_1, A_2, ..., A_n$$

Ereignisse, die
– sich gegenseitig ausschliessen
– zusammen den gesamten Ereignisraum bilden

Dann gilt für jedes Ereignis B:

$$P(B) = P(B|A_1) \cdot P(A_1) + P(B|A_2) \cdot P(A_2) + ... + P(B|A_n) \cdot P(A_n)$$

Diese Formel nennt man den ***Satz der totalen Wahrscheinlichkeit***. Sie beschreibt die Gesamtwahrscheinlichkeit eines Ereignisses als Summe aller möglichen Fälle, in denen es entstehen kann.

7.8. UNABHÄNGIGKEIT VON EREIGNISSEN

Zwei Ereignisse A und B heissen unabhängig, wenn das Eintreten des einen Ereignisses keinen Einfluss auf die Wahrscheinlichkeit des anderen hat. Dann gilt:

$$P(A|B) = P(A)$$

Das bedeutet: Auch wenn B bereits eingetreten ist, bleibt die Wahrscheinlichkeit von A unverändert. Setzt man das in die Multiplikationsregel ein, erhält man:

$$P(A \cap B) = P(A) \cdot P(B)$$

8. INFORMATIONSTHEORIE

Die Informationstheorie versucht, mathematisch zu messen:

– Wie viel Information Ereignisse enthalten (Erwartbares Ereignis liefert wenig Information, überraschendes Ereignis liefert viel Information)
– Wie unsicher eine Informationsquelle ist

– Wie effizient Informationen codiert werden können.

Information ist das, was Ungewissheit beseitigt oder reduziert. Die ***Entropie*** ist die Grösse, die die durchschnittliche Informationsmenge einer Quelle beschreibt. Die Einheit der Information ist das ***Shannon*** (benannt nach Claude Shannon). Um Verwirrung zwischen Datenmenge und Informationsgehalt zu stiften, wird in vielen Lehrmitteln ***Bit*** verwendet.

8.1. INFORMATIONSGEHALT

Der ***Informationsgehalt*** $I(x_k)$ eines Symbols x_k ist ein Mass dafür, wie viel **Information** das Symbol trägt, basierend auf seiner Wahrscheinlichkeit des Auftretens $p(x_k)$.

$$I(x_k) = \log_2\left(\frac{1}{p(x_k)}\right) = -\log_2(p(x_k))$$

Lies: n Möglichkeiten auf m Möglichkeiten einzugrenzen entspricht einer Information vom $\log_2(n/m)$ Bit.

Information ist:

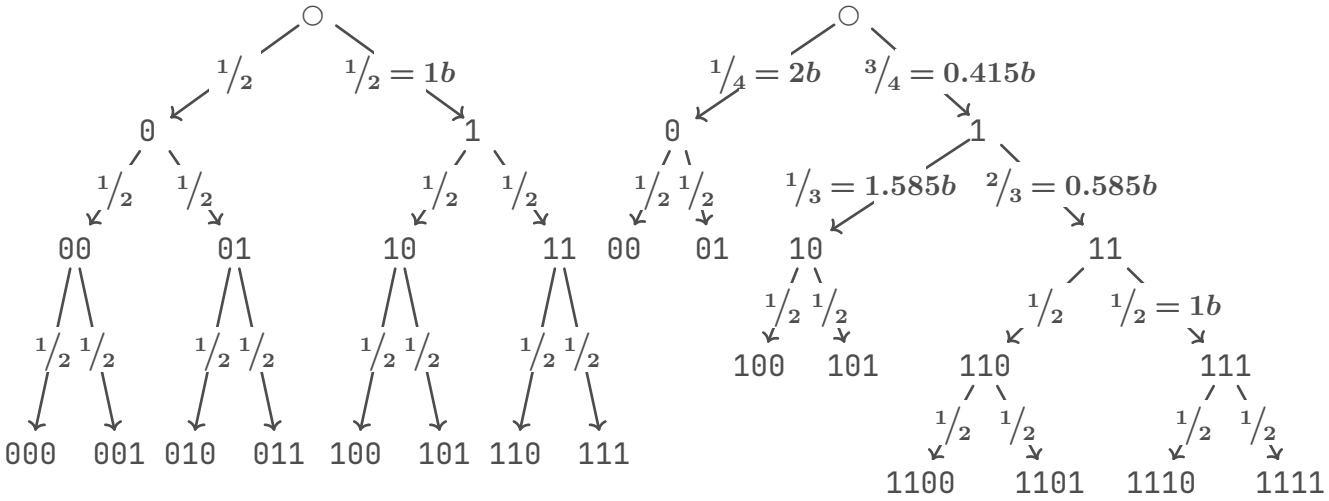
Nie negativ: $I(x) \geq 0$

Stetig: $p(x_k) \to p(x) \Rightarrow I(x_k) \to I(x)$

Monoton: $p(x_1) < p(x_2) \Rightarrow I(x_1) > I(x_2)$

Additiv: $I(x_1 x_2) = I(x_1) + I(x_2)$

Der Informationsgehalt kann auch als die theoretisch ideale (kürzeste) Länge des Codeworts interpretiert werden.



8.2. ENTSCHEIDUNGSGEHALT

Der ***Entscheidungsgehalt*** H_0 einer Quelle gibt an, wie viel Information im Durchschnitt benötigt wird, um ein Ereignis aus N **gleich wahrscheinlichen unterschiedlichen Ereignissen** zu identifizieren.

$$\mathrm{H}_0 = \log_2(N) \quad [\text{bit}]$$

> ***Bemerkung 1:*** **Entscheidungsgehalt und Informationsgehalt**
>
> Angenommen eine Quelle besitzt N mögliche und gleichwahrscheinliche Symbole.
>
> Wahrscheinlichkeit eines Symbols

– Bei gleichwahrscheinlichen Symbolen gilt: $p(x) = {}^1/_N$

Informationsgehalt eines Ereignisses: $I(x) = -\log_2(p(x))$
– Einsetzen von $p(x) = {}^1/_N$ ergibt $I(x) = -\log_2({}^1/_N) = \log_2(N)$

Ergebnis: $I(x) = \log_2(N) = \mathrm{H}_0$ für gleichwahrscheinliche Ereignisse.

⇒ Der Entscheidungsgehalt H_0 ist ein Spezialfall des Informationsgehalts $I(x)$ für gleichwahrscheinliche Ereignisse.

8.2.1. Entscheidungsfluss

Der ***Entscheidungsfluss*** (oft Entscheidungsrate oder Informationsfluss) beschreibt, wie viel Entscheidungsinformation pro Zeiteinheit t bereitgestellt wird.

$$H_0^* = \frac{\log_2(N)}{t} \quad \left[\frac{\text{bit}}{\text{zeit}}\right]$$

8.3. ENTROPIE

Die Entropie $\mathrm{H}(X)$ beschreibt den durchschnittlichen Informationsgehalt / durchschnittliche Unsicherheit einer Quelle.

$$\mathrm{H}(X) = \sum_{k=1}^{N} p(x_k) \cdot I(x_k) = \sum_{k=1}^{N} p(x_k) \cdot \log_2\left(\frac{1}{p(x_k)}\right) = -\sum_{k=1}^{N} p(x_k) \cdot \log_2(p(x_k)) \quad \left[\frac{\text{bit}}{\text{zeichen}}\right]$$

$p(x) = 1$	$p(x) = 0$	$0 \leq \mathrm{H}(X) \leq \log_2 N$
– Ergebnis ist sicher – $I(x) = 0$	– Ergebnis tritt nie auf – Trägt 0 zur Entropie bei	– Minimum: deterministische Quelle – Maximum: gleichverteilte Quelle

***Beispiel** 35:* $\mathrm{H}(X) = -(0.99 \cdot \log_2 0.99 + 0.01 \cdot \log_2 0.01) = 0.014 + 0.066 \approx 0.080$ bit

– A liefert fast keine Information, weil es praktisch immer kommt.
– B liefert sehr viel Information, weil es extrem selten ist.
  – Aber: B kommt nur 1% der Zeit, deshalb ist der durchschnittliche Informationsgehalt klein.

Zeichen	*Wahrscheinlichkeit*
A	0.99
B	0.01

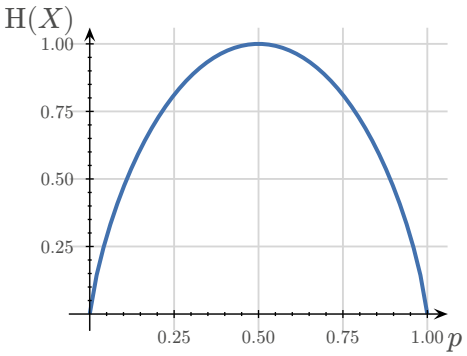
⇒ Ergebnis: ca. 0.08 Bit pro Symbol

8.4. REDUNDANZ DER QUELLE

Die Redundanz R_q beschreibt den Anteil vorhersehbarer Information / den Unterschied zwischen dem maximal möglichen Entscheidungsgehalt und der tatsächlichen Entropie der Quelle.

$$R_q = R_{\text{abs}} = \mathrm{H}_0 - \mathrm{H}(X) \quad \left[\frac{\text{bit}}{\text{zeichen}}\right]$$

$$R_{\text{rel}} = \frac{R_{\text{abs}}}{\mathrm{H}_0} = \frac{\mathrm{H}_0 - \mathrm{H}(X)}{\mathrm{H}_0} = 1 - \frac{\mathrm{H}(X)}{\mathrm{H}_0} \quad [\%]$$



Hohe Redundanz: Quelle stark vorhersehbar

Geringe Redundanz: Quelle nahe maximaler Unsicherheit

Eine Quelle mit hoher Redundanz enthält viele regelmässige Strukturen oder Abhängigkeiten. Diese können genutzt werden, um Daten effizienter zu codieren oder zu komprimieren.

8.5. CODIERUNG DER ZEICHEN

8.5.1. Codewortlänge

Die ***Codewortlänge*** ist die tatsächliche Symbollänge der Codewörter. Idealerweise sollte die Länge eines Codeworts dem Informationsgehalt des Symbols entsprechen:

$$L(x_k) \approx I(x_k) \approx -\log_2 p(x_k)$$

Da Codewörter aus einer **ganzen Anzahl Bits** bestehen müssen, wird aufgerundet:

$$L(x_k) = \lceil I(x_k) \rceil = \lceil -\log_2 p(x_k) \rceil \quad [\text{bit}]$$

8.5.2. Mittlere Codewortlänge

Die ***mittlere Codewortlänge*** $L(X)$ ist definiert als der gewichtete Durchschnitt der Längen der Codewörter, wobei jedes Gewicht der Auftrittswahrscheinlichkeit des entsprechenden Symbols gleich ist.

$$L(X, c) = \sum_{k=1}^{N} p(x_k) \cdot L(x_k) \quad \left[\frac{\text{bit}}{\text{zeichen}}\right]$$

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.

Es gilt für jeden Code $\mathrm{H}(X) \leq L(X)$

***Beispiel** 36:*

Folgende Zeichen mit entstprechenden Codewörter, deren Länge und ihren Wahrscheinlichkeiten sind gegeben:

Zeichen	*Codewort*	*Wahrscheinlichkeit p*	*Länge*
a	0	0.3	1
b	110	0.1	3
c	1111	0.1	4
d	1110	0.2	4
e	10	0.3	2

Nun berechnen wir die mittlere Codewortlänge L mit der oberigen Formel:

$$\begin{aligned} L &= (0.3 \cdot 1) + (0.1 \cdot 3) + (0.1 \cdot 4) + (0.2 \cdot 4) + (0.3 \cdot 2) \\ &= 0.3 + 0.3 + 0.4 + 0.8 + 0.6 \\ &= 2.4 \text{ bit} \end{aligned}$$

8.5.3. Redundanz des Codes

Die Redundanz des Codes R_c ist die Differenz zwischen der mittleren Codewortlänge und der Entropie der Quelle.

$$R_c = L - \mathrm{H}(x) \quad \left[\frac{\text{bit}}{\text{zeichen}}\right]$$

Interpretation: Beschreibt, wie ineffizient die Codierung ist / Verlust durch nicht-optimalen Code.

Eine Codierung mit $R = 0$ ist ***redundanzfrei***.

8.6. SHANNON'SCHES CODIERUNGSTHEOREM

Das ***Shannon-Theorem*** beschreibt die **theoretischen Grenzen der Datenkompression**. Für jede Informationsquelle mit mittlerer Codewortlänge $L(X)$ und deren Entropie $\mathrm{H}(X)$ gilt:

$$\mathrm{H}(X) \leq L(X) < \mathrm{H}(X) + 1$$

Entropie ist die theoretische Mindestlänge eines Codes / Grenze der Kompression. Praktische Codes können dieser Grenze sehr nahe kommen.

8.7. DISKRETE QUELLEN

Eine ***diskrete Quelle*** X mit dem Quellenalphabet $\Sigma = \{x_1, ..., x_n\}$ emittiert einen unendlichen Strom von Symbolen aus der Menge Σ. Jedes Symbol x_n besitzt eine Auftrittswahrscheinlichkeit $p(x_n) > 0$.

8.7.1. Quellen ohne Gedächtnis

Eine diskrete Quelle über dem Alphabet $\Sigma = \{x_1, ..., x_n\}$ heisst ***gedächtnislos***, wenn sie die Symbole $x_1, ..., x_n$ **stochastisch unabhängig voneinander** mit den Wahrscheinlichkeiten $p(x_1), ..., p(x_n)$ emittiert.

Heisst: Diskrete Quellen ohne Gedächtnis haben Symbole, die unabhängig von vorherigen Symbolen auftreten. Die Wahrscheinlichkeit für die beiden Zeichen x_i und x_{i+1} lautet damit:

Die ***Bigrammwahrscheinlichkeit*** oder ***Verbundwahrscheinlichkeit*** $p(x_i, x_{i+1})$ ist die Wahrscheinlichkeit, dass die Textsequenz $x_i x_{i+1}$ emittiert wird.

$$p(x_i, x_{i+1}) = p(x_i) \cdot p(x_{i+1})$$

Eine gedächtnislose Quelle mit $\Sigma = \{0, 1\}$ wird auch ***Bernoulli-Quelle*** genannt.

8.7.2. Quellen mit Gedächtnis (Markov-Quellen)

In der Praxis sind nur wenige Datenquellen vollständig gedächtnislos. Häufig hängt die Wahrscheinlichkeit eines Zeichens vom vorhergehenden Zeichen ab. Solche Kontextabhängigkeiten lassen sich mit bedingten Wahrscheinlichkeiten beschreiben:

Die ***Transitionswahrscheinlichkeit*** oder ***bedingte Wahrscheinlichkeit*** $p(x_{i+1}|x_i)$ ist die Wahrscheinlichkeit, dass auf das Symbol x_i das Symbol x_{i+1} folgt.

$$p(x_{i+1}|x_i) = \frac{p(x_i, x_{i+1})}{p(x_{i+1})}$$

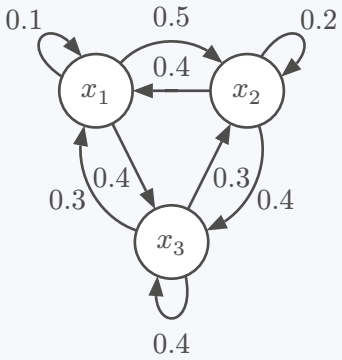
Beispiel 37: **Zeichenfolgen**

In deutschen und englischen Texten folgt auf den Buchstaben «q» praktisch immer «u».

$$p(u|q) \approx 1$$

***Beispiel** 38:*

Gegeben sei eine Markov-Quelle erster Ordnung mit drei Symbolen, die alle T Sekunden ein Symbol (x_1, x_2, x_3) erzeugt. Diese Zustände mit den Übertragungswahrscheinlichkeiten im stationären Fall sind im nachstehenden Markov-Diagramm wiedergegeben.



$$
\begin{aligned}
P(x_1) &= P(x_1)\cdot 0.1 + P(x_2)\cdot 0.4 + P(x_3)\cdot 0.3 \\
P(x_2) &= P(x_1)\cdot 0.5 + P(x_2)\cdot 0.2 + P(x_3)\cdot 0.3 \\
P(x_3) &= P(x_1)\cdot 0.4 + P(x_2)\cdot 0.4 + P(x_3)\cdot 0.4 \\
P(x_1) + P(x_2) + P(x_3) &= 1 \\
&\Rightarrow \\
P(x_1) &= 0.28 \\
P(x_2) &= 0.32 \\
P(x_3) &= 0.4
\end{aligned}
$$

$$
\begin{aligned}
H(X,Y) &= -\sum_{i=1}^{N}\sum_{j=1}^{N} P(x_i,y_k)\cdot \log_2(P(x_i,y_k)) \\
&= 0.144 + 0.397 + 0.354 \\
&\quad +0.38 + 0.254 + 0.38 \\
&\quad +0.367 + 0.267 + 0.423 \\
&= 3.066 \\
H(Y|X) &= H(X,Y) - H(X) \\
&= 3.066 - 1.57 \\
&= 1.496
\end{aligned}
$$

$P(Y\|X)$	x_1	x_2	x_3
x_1	0.1	0.5	0.4
x_2	0.4	0.2	0.4
x_3	0.3	0.3	0.4

$P(X,Y)$	x_1	x_2	x_3
x_1	0.028	0.14	0.112
x_2	0.128	0.064	0.128
x_3	0.12	0.12	0.16

x	$P(x)$	$I(x)$	$P(x)I(x)$
x_1	0.28	1.84	0.52
x_2	0.32	1.64	0.52
x_3	0.4	1.32	0.53
		$H(X)$	1.57

8.7.2.1. Entropie

Für zwei aufeinanderfolgende Zeichen gilt:

$$\mathrm{H}(X,Y) = \sum_{k=1}^{N}\sum_{i=1}^{N} p(x_k,y_i)\cdot \log_2\left(\frac{1}{p(x_k,y_i)}\right)$$

Mit

$$p(x_k,y_i) = p(x_k)\cdot p(y_i \mid x_k)$$

ergibt sich

$$\mathrm{H}(X,Y) = \mathrm{H}(X) + \mathrm{H}(Y \mid X)$$

Interpretation:

– $\mathrm{H}(X)$: Unsicherheit über das erste Zeichen
– $\mathrm{H}(X,Y)$: Unsicherheit über zwei aufeinanderfolgende Zeichen
– $\mathrm{H}(Y \mid X)$: Verbleibende Unsicherheit über Y wenn X bereits bekannt ist

Da Kontextinformation Unsicherheit reduziert, gilt:

$$\mathrm{H}(Y \mid X) \leq \mathrm{H}(Y) \leq \mathrm{H}_0$$

8.7.2.2. Redundanz

$$R_{\text{abs,oG}} = \mathrm{H}_0 - \mathrm{H}(Y)$$
$$R_{\text{abs,mG}} = \mathrm{H}_0 - \mathrm{H}(Y \mid X)$$
$$R_{\text{abs,oG}} \leq R_{\text{abs,mG}}$$

Interpretation: beschreibt, wie viel «überflüssige Struktur» in der Quelle steckt / Potenzial für Kompression.

Kontext reduziert Unsicherheit, somit sinkt die Entropie und die Redundanz steigt.

9. GRUPPENTHEORIE

9.1. GRUPPE

Definition 5: **Gruppe**

Eine ***Gruppe*** ist ein Paar $(G, \boxplus)$, bestehend aus einer nichtleeren Menge G und einer Abbildung

$$\boxplus = \begin{cases} G \times G \to G \\ (a, b) \quad \mapsto a \boxplus b \end{cases}$$

sodass folgende Eigenschaften erfüllt sind:

- Abgeschlossenheit bzgl. $\boxplus$ $\quad \forall a, b \in G.\ (a \boxplus b = c \Rightarrow c \in G)$
- $\boxplus$ ist assoziativ $\quad (a \boxplus b) \boxplus c = a \boxplus (b \boxplus c)$
- Es existiert ein neutrales Element bzgl. $\boxplus$ (id) $\quad a \boxplus \mathbb{1} = a$
- Es existiert zu jedem Element ein Inverses bzgl. $\boxplus$ $\quad a^{-1} \boxplus a = \mathbb{1}$

Beispiel: $(\mathbb{Z}, +)$

Wenn kommutativ: abel'sche Gruppe $\quad \forall a, b \in G.\ (a \boxplus b = b \boxplus a) \Rightarrow G \text{ abelian}$

Eine Gruppe ohne Invertierbarkeit heisst ***Halbgruppe***

9.2. KÖRPER (FIELD)

Definition 6: **Körper**

Ein ***Körper*** ist ein Triple $(F, \boxplus, \boxtimes)$ bestehend aus einer nichtleeren Menge F und zwei Abbildungen $\boxplus$ und $\boxtimes$ sodass folgende Eigenschaften erfüllt sind

- $(F, \boxplus)$ ist eine abel'sche Gruppe mit neutralem Element 0
- $(F, \boxtimes)$ ist eine abel'sche Gruppe mit neutralem Element 1
- Das Distributivgesetz gilt $\quad a \boxtimes (b \boxplus c) = (a \boxtimes b) \boxplus (a \boxtimes c) \ \wedge \ (a \boxplus b) \boxtimes c = (a \boxtimes c) \boxplus (b \boxtimes c)$

Beispiel: $(\mathbb{Z}_2, +, \cdot), (\mathbb{R}, +, \cdot)$

Seien $(M, \boxplus, \boxtimes)$ und $(M', \boxplus, \boxtimes)$ zwei Körper

- 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***.

9.3. RING

Definition 7: **Ring**

Ein ***Ring*** ist ein Triple $(R, \boxplus, \boxtimes)$, bestehend aus einer nichtleeren Menge R und zwei Abbildungen $\boxplus$ und $\boxtimes$ sodass folgende Eigenschaften erfüllt sind
- $(R, \boxplus)$ ist eine abel'sche Gruppe
- $(R, \boxtimes)$ ist eine Halbgruppe
- Das Distributivgesetz gilt $a \boxtimes (b \boxplus c) = (a \boxtimes b) \boxplus (a \boxtimes c) \ \wedge \ (a \boxplus b) \boxtimes c = (a \boxtimes c) \boxplus (b \boxtimes c)$

Beispiel: $(\mathbb{Z}, +, \cdot)$

Sei $(M, \boxplus, \boxtimes)$ 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.

Jeder Ring $(M, \boxplus, \boxtimes)$ 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***.

9.3.1. Division im Polynomring

Ziel: $p(x)$ in der Form $p(x) = q(x)s(x) + r(x)$ darzustellen mit $\deg r(x) < \deg q(x)$

Vorgehen: Ein Vielfaches von $q(x)$ von $p(x)$ subtrahieren

Beispiel 39: $(x^5 + x^2 + x + 1) : (x^3 + x^2 + x - 1)$

$$
\begin{array}{rccccccl}
 & x^5 & & & + & x^2 & +x & +1 & \\
- & x^5 & -x^4 & -x^3 & + & x^2 & & & \mid -(x^3 + x^2 + x - 1) \cdot x^2 \\
\hline
 & & -x^4 & -x^3 & + & x^2 & +x & & \\
+ & & x^4 & +x^3 & + & x^2 & -x & & \mid -(x^3 + x^2 + x - 1) \cdot (-x) \\
\hline
 & & & & & 3x^2 & & +1 & \\
\end{array}
$$

$$= (x^3 + x^2 + x - 1)(x^2 - x) + (3x^2 + 1)$$

9.3.2. Irreduzible polynome

Definition 8: **Irreduzibel**

Es sei R ein Integritätsring. Dann heißt ein Polynom $p \in R$ ***irreduzibel***, wenn $r \neq 0$ nicht invertierbar in R ist und für $g, h \in R$ und $p = gh$ entweder g oder h invertierbar ist.

Ein irreduzibles Polynom ein Polynom, das sich nicht als Produkt zweier nicht invertierbarer Polynome schreiben lässt und somit nicht in «einfachere» Polynome zerfällt.

Ob ein Polynom irreduzibel ist, hängt von der zugrundeliegenden algebraischen Struktur ab, die man betrachtet. Beispielsweise ist das Polynom x^2+1 im Polynomring über den reellen Zahlen $\mathbb{R}_x$ irreduzibel, da $x^2+1=(x+a)(x+b)$ keine reellen Lösungen hat

9.4. VEKTORRÄUME

***Definition** 9:* **Vektorraum**

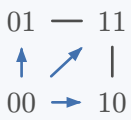
Sei $\mathbb{K}$ ein Körper. Eine Menge V bildet mit den beiden Abbildungen

$$\boxplus : V \times V \to V, \quad \boxtimes : \mathbb{K} \times V \to V$$

einen ***Vektorraum*** über $\mathbb{K}$, oder kurz einen $\mathbb{K}$-Vektorraum, wenn die folgenden Eigenschaften gelten:
- $(V, \boxplus)$ ist eine kommutative Gruppe
- $\boxplus$ und $\boxtimes$ erfüllen die Distributivgesetze
- $\boxtimes$ ist assoziativ
- 1 ist ein neutrales multiplikatives Element

***Beispiel** 40:* **Vektorraum $\mathbb{Z}_2^2$**

$$\mathbb{Z}_2^2 = \left\{ \begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}0\\1\end{pmatrix}, \begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}1\\1\end{pmatrix} \right\}$$



9.4.1. Generatormatrix

Untervektorräume von $\mathbb{K}^n$ besitzen die Eigenschaft, dass wir alle ihre Elemente mit einer ***Generatormatrix*** erzeugen können. Dies geschieht durch die Multiplikation der möglichen Kombinationen von λ_i mit G.

***Beispiel** 41:* **2-dimensionaler Vektorraum**

$$x = \begin{pmatrix}x_0\\x_1\\x_2\end{pmatrix} = \lambda_1 \begin{pmatrix}0\\1\\1\end{pmatrix} + \lambda_2 \begin{pmatrix}1\\1\\0\end{pmatrix} = \begin{pmatrix}0&1\\1&1\\1&0\end{pmatrix} \cdot \begin{pmatrix}\lambda_1\\\lambda_2\end{pmatrix}$$

$$x^\top = (x_0, x_1, x_2) = (\lambda_1, \lambda_2) \cdot \underbrace{\begin{pmatrix}0&1&1\\1&1&0\end{pmatrix}}_{\text{Generatormatrix } G}$$

9.5. GALOIS FELDER ($GF(2^3)$)

Frage: Hat das Polynom x^3+x+1 in $\mathbb{Z}_2$ eine Lösung?

9.5.1. Körpererweiterung

Ziel: Einen Körper bauen, in dem unsere Gleichungen lösbar sind.

Beispiel *42:*

$$\begin{aligned}\mathbb{R} &\to \mathbb{C}: & x^2+1=0 &\to i^2=-1\\ \mathbb{Z}_2 &\to GF(2^3): & x^3+x+1=0 &\to a^3+a+1=0\end{aligned}$$

Ausgangslage: Körper $\mathbb{Z}_2=\{0,1\}$. Mit erzeugendem Element a ergibt sich:

$$\begin{aligned}g(p) &= a^3+a+1=0\\ &\Leftrightarrow a^3=-a-1\equiv a+1 \bmod 2\end{aligned}$$

$a^0=1$
$a^1=a$
$a^2=a^2$
$a^3=a^3=a+1$
$a^4=a^2+a$
$a^5=a^3+a^2=a^2+a+1$
$a^6=a^3+a^2+a=a^2+2a+1=a^2+1$
$a^7=a^3+a=2a+1=1=a^0$
a^7 führt somit wieder zu a^0

a^n	a^0	a^1	a^2	b
0	1	0	0	100
1	0	1	0	010
2	0	0	1	001
3	1	1	0	110
4	0	1	1	011
5	1	1	1	111
6	1	0	1	101

So kommen wir zum Erweiterungskörper:

$$\begin{aligned}GF(2^3) &= \{0, & 1, & a, & a^2, & a+1, & a^2+a, & a^2+a+1, & a^2+1\}\\ &\to \{000, & 001, & 010, & 100, & 011, & 110, & 111, & 101\ \}\end{aligned}$$

– $\mathbb{Z}_2 \subset GF(2^3)$
– Basis: $\{1,a,a^2\}\to$ alle Linearkombinationen $c_0+c_1a+c_2a^2, c_i\in\{0,1\}$ führen zur Struktur des Erweiterungskörpers

9.5.2. Zyklische Gruppen

Die Nicht-Null-Elemente von $GF(2^3)\{1,a,a^2,a^3,a^4,a^5,a^6\}$ bilden eine zyklische Gruppe bezüglich der Multiplikation

– Abgeschlossenheit: $a^i\cdot a^j=a^{i+j} \bmod 7$
– Assoziativität: $(a^i\cdot a^j)\cdot a^k=a^i\cdot(a^j\cdot a^k)$
– Neutrales Element: $1\cdot a^i=a^i$
– Inverses Element: $\forall a^i.\ (a^i\cdot a^{7-i}=a^7=1)$
– Kommutativ: $a^i\cdot a^j=a^j\cdot a^i$

$\Rightarrow$ bilden eine abel'sche Gruppe

10. QUELLENCODIERUNG

Die Quellencodierung verfolgt das Ziel, die zu übertragende Nachricht in einer Form an den Sender zu übergeben, die einen möglichst hohen Informationsgehalt aufweist.

$$R_c\to 0\Rightarrow L\approx \mathrm{H}(X)$$

Anwendungen:

Datenkomprimierung	*Verschlüsselung*
– Verlustfrei (bijektiv) – Verlustbehaftet (surjektiv)	– Symmetrisch – Asymmetrisch

10.1. PRÄFIXFREIE CODES

Ein Code besitzt die **Präfixeigenschaft**, wenn kein Codewort der Anfang eines anderen Codeworts ist und somit die ***Fano-Bedingung*** ist.

Präfixcodes sind wichtig, weil sie eine **eindeutige und sofortige Decodierung** ermöglichen.

Symbol	*Code*
A	0
B	10
C	110
D	111

10.1.1. Kraft'sche Ungleichung

Für jede Folge $l_1, ..., l_n$ von natürlichen Zahlen gilt: Es existiert ein präfixfreier Code mit dem Codealphabet Π und den Codewortlängen $l_1, ..., l_n$

$$\Leftrightarrow \sum_i \frac{1}{|\Pi|^{l_i}} \leq 1$$

10.2. DATENKOMPRIMIERUNG

Kompression ist nur möglich, wenn **Redundanz** vorhanden ist.

Art der Redundanz	*Beispiel*	*Verfahren*
Wiederholungen	AAAAAAAA	«Run-Length-Encoding (RLE)» (Seite 42)
Ungleiche Wahrscheinlichkeiten	häufige Zeichen	«Huffman-Codierung» (Seite 41)
Muster / Struktur	ABCABCABC	«Lempel-Ziv-Codierung (LZ77)» (Seite 43)

Verfahren zur Datenkomprimierung

Statistische Verfahren	z.B. Huffman-Codierung für die deutsche Sprache – nutzen bekannte Wahrscheinlichkeiten	Eigenart der Daten (Wahrscheinlichkeiten) werden berücksichtigt =Entropiecodierungen
Adaptive Verfahren	z.B. Huffman-Codierung mit gemessener Häufigkeitsverteilung – lernen Wahrscheinlichkeiten während der Codierung	
Wörterbuchbasierte Verfahren	z.B. LZ77, LZ78, LZW, DEFLATE – nutzen wiederkehrende Muster im Datenstrom	Eigenart der Daten (Wahrscheinlichkeiten) werden **nicht** explizit berücksichtigt – stattdessen Muster =Substitutionscodierungen

10.2.1. Huffman-Codierung

Ein Verfahren zur Entwicklung eines präfixfreien Codes mit minimaler mittlerer Codewortlänge L. Es gilt also immer $\mathrm{H} \leq L < \mathrm{H} + 1$, heisst die mittlere Codewortlänge kann nie kleiner sein als die Entropie.

Kerngedanke

– Häufige Zeichen $\rightarrow$ kurze Codes

– Seltene Zeichen → lange Codes

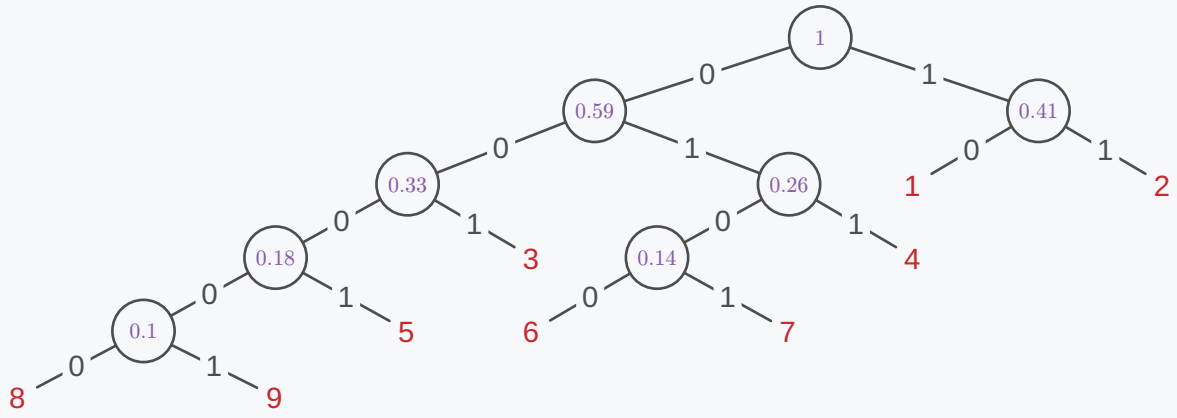
Verfahren

1) Sortiere Zeichen nach Wahrscheinlichkeit
2) Kombiniere die zwei kleinsten
3) Ersetze sie durch neuen Knoten
4) Wiederhole, bis ein Baum entsteht
5) Weise 0 / 1 entlang der Kanten zu

Der Huffman-Code ist nicht eindeutig aber immer optimal.

***Beispiel** 43:*

x_i	1	2	3	4	5	6	7	8	9
$p(x_i)$	0.22	0.19	0.15	0.12	0.08	0.07	0.07	0.06	0.04



$\Rightarrow$

2	1	4	3	7	6	5	9	8
11	10	011	001	0101	0100	0001	00001	00000

Entropie

$$\mathrm{H}(X) \approx 2.89 \text{ Bit}$$

10.2.2. Run-Length-Encoding (RLE)

– Wiederholungen werden nicht mehrfach gespeichert, sondern gezählt.
– wird bei vielen Bildformaten benutzt zum Beispiel BMP, PCX und TIFF.
– gehört zu musterbasierten Verfahren
– Spezialfall von Lempel-Ziv
– sehr einfach und schnell
– keine Wahrscheinlichkeiten nötig

Kerngedanke

– Wiederholungen ersetzen durch: `Symbol,Anzahl`

Gut geeignet für

– lange Wiederholungen
– Bilder (z.B. einfarbige Flächen)
– einfache Muster

Schlecht geeignet für

– zufällige Daten
– häufige Wechsel

Beispiel *44:*

Quelltext	Codiert
$w =$ agggbbehfffgggg	$w_c =$ a3g2beh3f4g
$\|w\| = 15$	$\|w_c\| = 11$

Beispiel *45:* **bits**

Quelltext	Codiert
$w =$ 1111 1110 0000 1000 0001 1111	$w_c =$ 111 101 001 110 101
$\|w\| = 24$	$\|w_c\| = 15$

10.2.3. Lempel-Ziv-Codierung (LZ77)

Kerngedanke

Muster werden wiedererkannt und ersetzt durch Verweis auf vorherige Sequenz.

Grundidee

Zwei Bereiche

– Search Buffer → Vergangenheit
– Look-Ahead Buffer → Zukunft

Grosser Buffer	Kleiner Buffer
Bessere Kompression, weil längere Muster erkennbar	Schneller, weil weniger speicher
Langsamer	Schlechtere Kompression

Codierung

Statt Zeichen: `Distanz,Länge,Symbol`
– Distanz → wie weit zurück
– Länge → wie lang das Muster
– Symbol → nächstes Zeichen

Ein nach Lempel-Ziv codierter Code kann grösser sein als der Originalcode, falls der Eingang nicht genug Wiederholungen enthält.

***Beispiel** 46:*

Search Buffer							CP	Lookahead Buffer					Out
							a	b	r	a	c	ada…	(0,**0**,a)
						a	b	r	a	c	a	dab…	(0,**0**,b)
					a	b	r	a	c	a	d	abr…	(0,**0**,r)
				a	b	r	a	c	a	d	a	bra	(3,**1**,c)
		a	b	r	**a**	c	a	d	a	b	r	a	(2,**1**,d)
a	**b**	**r**	**a**	c	a	d	a	b	r	a			(7,**4**,eof)
c	a	d	a	b	r	a							
7	6	5	4	3	2	1							

Tokens

– Distanz = 3Bit
– Länge = 3Bit
– Symbol = 8Bit (ASCII)

Effizienzberechnung

– 14 Bit / Token: $6 \cdot 14 = 84$ Bit
– ASCII: $11 \cdot 8 = 88$ Bit

```
000 000 01100001    (0,0,a)
000 000 01100010    (0,0,b)
000 000 01110010    (0,0,r)
011 001 01100011    (3,1,c)
010 001 01100100    (2,1,d)
111 100 00000000    (7,4,eof)
```

Der Lempel-Ziv-Algorithmus und die Huffman-Codierung kann wie folgt kombiniert werden: Zuerst werden die Daten Lempel-Ziv-Komprimiert, anschliessend das Wörterbuch Huffman-codiert, d.h. die gefundenen Phrasen werden entsprechend ihrer Häufigkeit codiert.

10.2.4. Lempel-Ziv-Welch-Komprimierung (LZW)

Die Lempel-Ziv-Welch-Komprimierung ist ein Verfahren zur verlustfreien Datenkompression, das auf der Lempel-Ziv- Komprimierung aufbaut. Sie verwendet ein Wörterbuch, um wiederkehrende Sequenzen zu identifizieren und durch kürzere Codes zu ersetzen.

10.2.4.1. Funktionsweise

1) Beginne mit einem Wörterbuch, das bereits alle möglichen Einzelzeichen enthält.
2) Suche im Wörterbuch die längste Sequenz, die mit den nächsten Zeichen $(n+1)$ der Eingabe übereinstimmt.
3) Speichere den Index des gefundenen Wörterbucheintrags.
4) Erstelle einen neuen Eintrag im Wörterbuch mit der gefundenen Sequenz, gefolgt vom nächsten Zeichen der Eingabe.
5) Verschiebe das Eingabefenster um die Anzahl der codierten Zeichen.
6) Wiederhole den Prozess, bis alle Zeichen codiert sind.

***Beispiel** 47:*

Gegeben

Zu kodierende Zeichenabfolge: 123123123123

Wörterbuch:

Index	Eintrag
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9

Vorgehen

Buffer	Erkannte Zeichen (Index)	Index: Neuer Eintrag
123123123123	1 (1)	10: 12
123123123123	2 (2)	11: 23
123123123123	3 (3)	12: 31
123123123123	12 (10)	13: 123
123123123123	31 (12)	14: 312
123123123123	23 (11)	15: 231
123123123123	123 (13)	-

Neues Wörterbuch:

Index	Eintrag
0-9	Wie bisher
10	12
11	23
12	31
13	123
14	312
15	231

Daraus folgt die codierte Nachricht: 1 2 3 10 12 11 13

10.2.5. Kompressionsrate

$$R(\%) = \frac{L_{\text{komprimiert}}}{L_{\text{original}}} \cdot 100$$

$$L_{\text{original}} = \text{Länge der ursprünglichen Sequenz}$$

$$L_{\text{komprimiert}} = \text{Länge der komprimierten Sequenz}$$

10.3. VERSCHLÜSSELUNG

Wenn eine Nachricht über einen offenen Kanal übertragen wird, entstehen sofort mehrere Probleme:

Vertraulichkeit: Nur der gewünschte Empfänger soll den Inhalt lesen können.

Integrität: Der Inhalt soll nicht unbemerkt verändert werden können.

Authentizität: Der Empfänger soll prüfen können, von wem die Nachricht stammt.

Ziel ist die gezielte Konstruktion einer leicht ausführbaren, aber schwer umkehrbaren Operation.

10.3.1. Substitutionsverfahren

– Caesar-Chiffre

10.3.1.1. Verschlüsselung

Ordnen wir Buchstaben Zahlen zu:

$A = 0, B = 1, ..., Z = 25$

Dann gilt:

$c = (m + k) \bmod 26$

– m: Klartext
– k: Schlüssel
– c: Chiffretext

$\Rightarrow$ Interpretation: Addition in $\mathbb{Z}_{26}$

10.3.1.2. Entschlüsselung

$m = c - k \bmod 26$

$\Rightarrow$ Rückwärtsoperation ist trivial

***Beispiel** 48:*

$$m = \text{bald sind sommerferien}$$
$$k = 4$$
$$c = \text{feph wmrh wsqqivjivmir}$$

10.3.1.3. Unsicher!

– Die statistischen Eigenschaften von Klar- und Chiffriertext sind nach wie vor unverändert
– Kennen wir die Sprache und ist unsere Probe gross genug, können wir den Schlüssel leicht ermitteln
– Schlüsselanzahl ist recht übersichtlich, hier braucht es keinen Computer, um alle auszuprobieren

10.3.2. Transpositionsverfahren

– Zeichen werden nicht verändert
– Nur ihre Position wird verändert

$(x_1, x_2, ..., x_n) \to \left(x_{\pi(1)}, x_{\pi(2)}, ..., x_{\pi(n)}\right),\ \pi = \text{Permutation}$

***Beispiel** 49:*

Klartext

DIE WORTE HOER ICH WOHL ALLEIN MIR FEHLT DER GLAUBE

Chiffretext

DTILNHGIECAMLLEHHLITAWOWLRDUOE-OEFEBRRHIERE

→ Erstellen einer Tabelle zeilenweise

← Auslesen spaltenweise

D	I	E	W	O	R
T	E	H	O	E	R
I	C	H	W	O	H
L	A	L	L	E	I
N	M	I	R	F	E
H	L	T	D	E	R
G	L	A	U	B	E

10.3.2.1. Unsicher!

– Zeichenhäufigkeiten bleiben exakt erhalten
– Typische Muster bleiben sichtbar
– Statistische Analyse möglich

10.3.3. Polyalphabetisches Verfahren

– Vigenère-Chiffre

Nicht nur ein Alphabet sondern mehrere. Zuordnung hängt von der Position ab.

$c_i = (m_i + k_i) \bmod 26$

– m_i: i-tes Klartextzeichen (als Zahl)
– k_i: i-tes Schlüsselzeichen (als Zahl)
– c_i: i-tes Zeichen im Chiffretext

⇒ jedes Zeichen wird unterschiedlich verschlüsselt

10.3.3.1. Unsicher!

– Der Schlüssel wird periodisch wiederholt, dadurch entstehen wiederkehrende Muster im Chiffretext
– Diese Muster erlauben es, die Schlüssellänge zu bestimmen

⇒ Der Text zerfällt in mehrere Teilfolgen → jede Teilfolge ist wieder eine Caesar-Chiffre
⇒ Mehr Komplexität – aber die gleiche Grundstruktur
⇒ Das Verfahren bleibt systematisch angreifbar

10.3.4. Probleme

Alle bisherigen Verfahren basieren auf:

– einfacher Algebra (Addition, Permutation)
– erkennbarer Struktur

⇒ Die Struktur ist zu einfach – und deshalb angreifbar.

10.3.5. Moderne symmetrische Verschlüsselung

	DES (historisch)	*AES (heute Standard)*
Chiffre	Blockchiffre (64 Bit Blöcke)	Blockchiffre (128 Bit Blöcke)
Schlüssel	56 Bit	128/192/256 Bit
Basiert auf	– Substitution (S-Boxen) – Permutation – Viele Runden (Feistel-Struktur)	– Substitution (S-Box) – Lineare Transformationen – Mehrere Runden
Sicherheit	Heute nicht mehr sicher (Schlüssel zu kurz)	Aktuell nicht gebrochen

⇒ Sicherheit entsteht durch komplexe, nichtlineare Transformationen

10.3.5.1. Warum moderne symmetrische Verfahren sicher sind

10.3.5.1.1. Eigenschaften moderner Blockchiffren

– Kombination vieler einfacher Operationen
– keine einfache algebraische Struktur sichtbar
– starke Durchmischung von Bits (Diffusion) führt zu Avalanche-Effekt: kleine Änderungen → grosse Auswirkungen

10.3.5.1.2. Angriffe

– keine einfachen statistischen Methoden mehr möglich
– nur noch:
  – brute force
  – sehr komplexe Spezialangriffe

⇒ Die Struktur ist so komplex, dass sie praktisch nicht mehr ausgenutzt werden kann

Angriffsmöglichkeiten

Brute Force:
– alle Schlüssel ausprobieren
– schauen, welcher sinnvollen Klartext ergibt

Brute Force ignoriert komplett:
– Struktur
– Mathematik

Differenzielle Kryptanalyse:
– Unterschiede im Input
– Unterschiede im Output

Analysieren, wie sich diese durch das System bewegen Ziel: nicht alle Schlüssel testen, sondern Information über den Schlüssel gewinnen

10.3.5.2. Probleme

– Sender und Empfänger brauchen denselben Schlüssel
– dieser Schlüssel muss vorher ausgetauscht werden
– Für n Teilnehmer werden $\binom{n}{2} = {}^{n(n-1)}/_{2}$ verschiedene Schlüssel benötigt

$\rightarrow$ quadratisches Wachstum (Schlüsselexplosion)

10.3.6. Asymmetrische Verschlüsselung

– Zwei unterschiedliche Schlüssel
– öffentlicher Schlüssel (darf bekannt sein)
– privater Schlüssel (bleibt geheim)

$\Rightarrow$ Wir lösen das Problem nicht durch mehr Komplexität, sondern durch ein neues mathematisches Konzept

10.3.6.1. RSA

Ziel:
– Verschlüsselung mit einem öffentlichen Schlüssel
– Entschlüsselung mit einem privaten Schlüssel

Eigenschaften:
– Jeder kann verschlüsseln
– Nur der Besitzer des privaten Schlüssels kann entschlüsseln
– Der private Schlüssel lässt sich aus dem öffentlichen nicht einfach berechnen

Für $a \in \mathbb{Z}_q$ ist $b \in \mathbb{Z}_q$ das ***multiplikative inverse*** von a, wenn $a \cdot b \equiv 1 \bmod q$

10.3.6.1.1. Berechnung

1) Wähle zwei Primzahlen p, q
2) Berechne $n = p \cdot q$
3) Berechne $\varphi(n) = (p-1)(q-1)$
4) Wähle a, b so, dass $a \cdot b \equiv 1 \bmod \varphi(n)$
5) Vergesse $p, q, \varphi(p \cdot q)$. Brauchen wir nicht und riskieren nur, dass uns jemand hackt

Public key ist nun n, b, Private key ist n, a

– Verschlüsseln: $z = c^a \bmod n$
– Entschlüsseln: $c = z^b \bmod n = c^{a^b} \bmod n$

10.3.6.1.2. Satz von Euler

Sei $n \in \mathbb{N} \setminus \{0\}$ und $z \in \mathbb{Z}$ mit $\mathrm{ggT}(z, n) = 1$. Dann ist $z^{\varphi(n)} \equiv 1 \bmod n$.

10.3.6.1.2.1. Euler'sche φ-Funktion

Sei $n \in \mathbb{N} \setminus \{0\}$ und $\mathbb{Z}_n^* = \{x \in \mathbb{Z}_n \mid x \text{ hat ein multiplikatives Inverses in } \mathbb{Z}_n\}$.
Dann heisst $\varphi(n)$:

$$\begin{aligned}\varphi(n) &= \text{Anz. Elemente in } \mathbb{Z}_n \text{ mit mult. Inversen} \\ &= \text{Anz. Zahlen } 1 \leq q \leq n \text{ mit } \mathrm{ggt}(q,n) = 1 \\ &= |\mathbb{Z}_n^*|\end{aligned}$$

Rechenregeln

1) Sei $n \in \mathbb{N}$ eine Primzahl, dann $\varphi(n) = n - 1$
2) Sei $n \in \mathbb{N}$ eine Primzahl und $p \in \mathbb{N} \setminus \{0\}$, dann $\varphi(n^p) = n^{p-1} \cdot (n-1)$
3) Seien $m, n \in \mathbb{N} \setminus \{0\}$ und $\mathrm{ggT}(m,n) = 1$, dann $\varphi(n \cdot m) = \varphi(n) \cdot \varphi(m)$

10.3.6.1.3. Erweiteter Euklidscher Algorithmus

Seien $a, b \in \mathbb{N}, a \neq b, a \neq 0, b \neq 0$
Initialisierung: Setze $x := a, y := b, q := x \div y, r := x - q \cdot y, (u, s, v, t) = (1, 0, 0, 1)$ (d.h. bestimme q und r so, dass $x = q \cdot y + r$ ist)
Wiederhole bis $r = 0$ ist
Ergebnis: $y = \mathrm{ggT}(a,b) = s \cdot a + t \cdot b$
Wenn $\mathrm{ggT}(a,b) = 1$ ist, dann folgt: $t \cdot v \equiv 1 \bmod a$

***Beispiel** 50:* $\mathrm{ggT}(99, 79)$

i	$x = y_{-1}$	$y = r_{-1}$	$q = x \div y$	$r = x - q \cdot y$	$u = s_{-1}$	$s = u_{-1} - q_{-1} \cdot s_{-1}$	$v = t_{-1}$	$t = v_{-1} - q_{-1} \cdot t_{-1}$
$i = 0$	99	79	1	20	1	0	0	1
$i = 1$	79	20	3	19	0	1	1	-1
$i = 2$	20	19	1	1	1	-3	-1	4
$i = 3$	19	1	19	0	-3	4	4	-5

Daraus folgend:
– $\mathrm{ggT}(99,79) = 4 \cdot 99 + (-5) \cdot 79 \Leftrightarrow 396 - 395 = 1$
– $99 + (-5) = 94$ ist mult. Inv. von 79 in $\mathbb{Z}_{99}$
– $79 + 4 = 83 \equiv 4$ ist mult. Inv. von 99 in $\mathbb{Z}_{79}$

10.3.6.2. Grösse

1024-Bit RSA: zwei Primzahlen mit je 512 Bit
– Anzahl möglicher Primzahlen: $\approx 10^{151}$

2048-Bit RSA: zwei Primzahlen mit je 1024 Bit
– Anzahl möglicher Primzahlen: $\approx 10^{305}$

4096-Bit RSA: zwei Primzahlen mit je 2048 Bit
– Anzahl möglicher Primzahlen: $\approx 10^{613}$

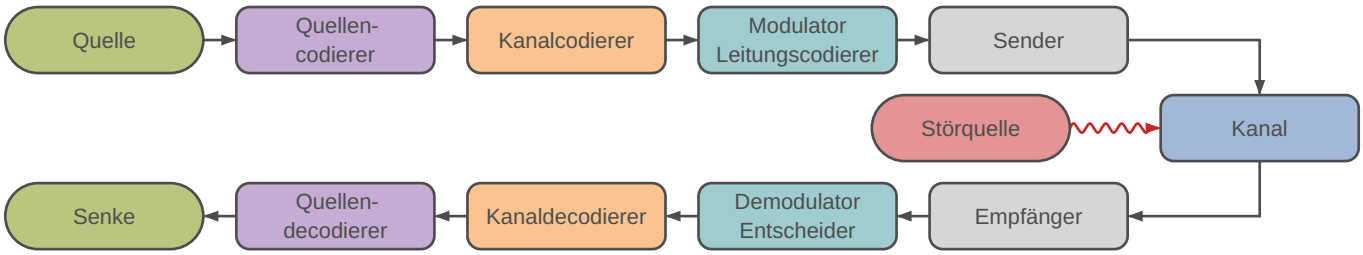
Die Gesamtzahl der möglichen Kombinationen von zwei Primzahlen aus 10^{151} Primzahlen (1024-Bit RSA-Schlüssel) ist gegeben durch die Kombination ohne Wiederholung:

$$t = \frac{\text{Kombinationen}}{\text{Faktorisierungen pro Sekunde}} = \frac{\frac{1}{2}(10^{151})^2 s}{100 \cdot 10^9} = \frac{10^{302} s}{2 \cdot 10^{11}} \approx 10^{290} s$$

11. KANALMODELL

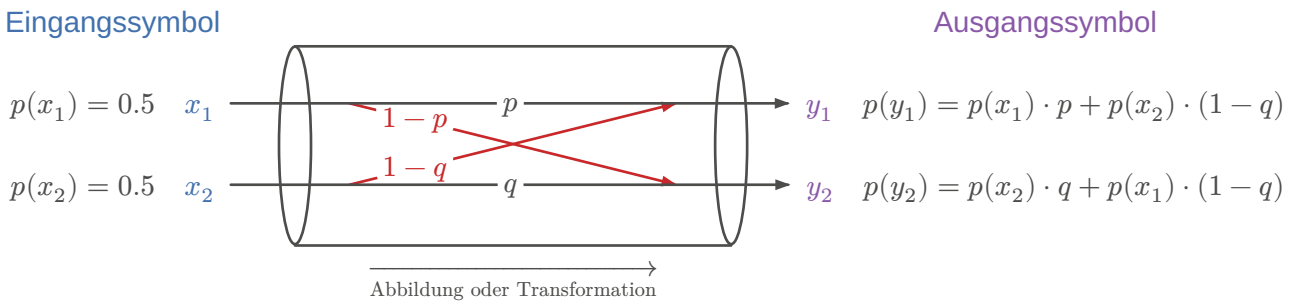
Mathematisches Modell, das beschreibt, wie Informationen durch einen Kommunikationskanal übertragen werden.

Kanal: Medium der Übertragung



11.1. KANALMATRIX

Auch ***Übergangsmatrix*** genannt. Gibt an, wie die Eingangssignale (gesendete Symbole) durch den Kanal in Ausgangssignale (empfangene Symbole) überführt werden. Diese Matrix beinhaltet die bedingten Wahrscheinlichkeiten $P(Y|X)$, wobei jedes Element der Matrix die Wahrscheinlichkeit darstellt, dass ein bestimmtes Ausgangssymbol y empfangen wird, gegeben ein gesendetes Eingangssymbol x.



Gegeben x

$$P(Y|X) = \begin{pmatrix} p & 1-p \\ 1-q & q \end{pmatrix}, \quad \text{Rot} \rightarrow \text{Fehlerwahrscheinlichkeit}$$

Resultierendes y

$$P(Y|X) = \underbrace{\begin{pmatrix} p(y_1|x_1) & p(y_2|x_1) \\ p(y_1|x_2) & p(y_2|x_2) \end{pmatrix}}_{\text{Kanalmatrix}}$$

$$p(y_1) = p(x_1) \cdot p(y_1|x_1) + p(x_2) \cdot p(y_1|x_2)$$

$$p(y_2) = p(x_1) \cdot p(y_2|x_1) + p(x_2) \cdot p(y_2|x_2)$$

Um die Kanalmatrix eines Kanals zu ermitteln, sendet man wiederholt ein bekanntes Zeichen, zeichnet die Empfänge auf und berechnet aus diesen Daten die Häufigkeiten, um die Matrix zu erstellen.

11.1.1. Binary Symmetric Channel (BSC)

Ein BSC ist ein binärer Kanal (2 inputs, 2 outputs), bei dem die Wahrscheinlichkeit für einen Bitflip für beide Zeichen gleich ist.

***Beispiel** 51:*

$$P(Y|X) = \begin{pmatrix} 0.95 & 0.05 \\ 0.05 & 0.95 \end{pmatrix}$$

Uncodierte Übertragung:

$$\text{Rate} \quad R = \frac{4}{4} = 1$$
$$\text{OK} \quad P_{\text{OK}} = 0.95^4 \approx 0.815$$
$$\text{Restfehlerwahrscheinlichkeit} \quad P_{\text{Rest}} = 1 - 0.815 = 0.185$$

Codiert mit Hamming(7,4):

$$\text{Rate} \quad R = \frac{4}{7} \approx 0.571$$
$$\text{OK} \quad P_{\text{OK}} = 0.95^7 + \binom{7}{1} 0.05 \cdot 0.95^6 \approx 0.955$$
$$\text{Restfehlerwahrscheinlichkeit} \quad P_{\text{Rest}} = 1 - 0.955 = 0.045$$

11.1.1.1. Kanalkapazität

Kanalkapazität (für BSC): $C = 1 - \mathrm{H}(Y|X) =$ Maximale Informationsrate, bei der Fehler gegen 0 möglich sind

– $R < C$: Fehlerfreie Übertragung möglich
– $R > C$: Fehler unvermeidbar

$\rightarrow$ Redundanz hinzufügen, um korrekte Übertragung sicherzustellen

11.1.2. Generalisierung

$$P(Y|X) = \underbrace{\left.\begin{pmatrix} p(y_1|x_1) & p(y_2|x_1) & \dots & p(y_n|x_1) \\ p(y_1|x_2) & p(y_2|x_2) & \dots & p(y_n|x_2) \\ \vdots & \vdots & \ddots & \vdots \\ p(y_1|x_m) & p(y_2|x_m) & \dots & p(y_n|x_m) \end{pmatrix}\right\} \text{Zeilen} = \text{gesendete Symbole}}_{\text{Spalten} = \text{empfangene Symbole}}$$

$$P(Y) = P(X)^\top \cdot P(Y|X)$$

$$p(y_k) = \sum_{i=1}^{m} p(x_i) \cdot p(y_k|x_i)$$

11.1.3. Nicht gestörter Kanal

= Einheitsmatrix

$$P(Y|X) = \mathbb{1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

11.1.4. Vollständig gestörter Kanal

= Matrix, bei der alle Elemente gleich sind

$$P(Y|X) = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix}$$

11.1.5. Verlustfreie (rauschbehaftete) Matrix

Summe jeder Zeile muss 1 ergeben.

$$P(Y|X) = \begin{pmatrix} 0.7 & 0.3 \\ 0.9 & 0.1 \end{pmatrix}$$

11.2. ENTSCHEIDER

Der «Entscheider» (auch Decider, Decoder) ist die Stelle, die nach Empfang eines Signals eine diskrete Entscheidung trifft, nämlich welches gesendete Codewort/ welche Nachricht am wahrscheinlichsten ist.

11.2.1. Maximum-Likelihood-Verfahren (ML)

Der Entscheider wählt (pro Spalte) das x, das die ***Likelihood*** maximiert: «Welche gesendete Möglichkeit würde das beobachtete y am ehesten erzeugen?»

Die Zuordnung erfolgt durch:

$$\hat{x}_{\text{ML}} = \arg\max_{x_i} p(y_i|x_i)$$

***Beispiel** 52:*

$$P(Y|X) = \begin{pmatrix} 0.2 & 0.5 & 0.3 \\ 0.7 & 0.2 & 0.1 \\ 0.4 & 0 & 0.6 \end{pmatrix} \Rightarrow \begin{cases} y_1 \to x_2 \\ y_2 \to x_1 \\ y_3 \to x_3 \end{cases}$$

11.2.2. Maximum-A-Posteriori-Verfahren (MAP)

Der Entscheider maximiert das ***Posterior***: «Welche gesendete Möglichkeit ist insgesamt am wahrscheinlichsten, unter Einbezug der Prior-Wahrscheinlichkeiten $p(x)$?»

Die Zuordnung erfolgt durch:

$$\hat{x}_{\text{MAP}} = \arg\max_{x_i} p(x_i) \cdot p(y_i|x_i)$$

***Beispiel** 53:*

$$P(X) = \begin{pmatrix} 0.9 \\ 0.09 \\ 0.01 \end{pmatrix} \quad P(Y|X) = \begin{pmatrix} 0.2 & 0.5 & 0.3 \\ 0.7 & 0.2 & 0.1 \\ 0.4 & 0 & 0.6 \end{pmatrix} \Rightarrow \begin{cases} y_1 \to x_1 \\ y_2 \to x_1 \\ y_3 \to x_1 \end{cases}$$

11.3. BERECHNUNGEN

Beispiele bauen auf folgenden Daten auf, falls nicht explizit erwähnt:

$$P(Y|X) = \begin{pmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{pmatrix}$$

$$p(x_1) = 0.3$$

$$p(x_2) = 0.7$$

$$\text{Übertragungsrate} = 1\frac{\text{kbit}}{s}$$

11.3.1. Eingangswahrscheinlichkeit

$$p(x_i) = \frac{p(x_i|y_j) \cdot p(y_j)}{p(y_j|x_i)}$$

Beispiel *54:*

$$p(x_1) = 0.3$$
$$p(x_2) = 0.7$$

11.3.2. Ausgangswahrscheinlichkeit

$$p(y_j) = \sum_{i=1}^{m} p(x_i) \cdot p(y_j|x_i)$$

Beispiel *55:*

$$p(y_1) = 0.3 \cdot 0.9 + 0.7 \cdot 0.2 = 0.41$$
$$p(y_2) = 0.3 \cdot 0.1 + 0.7 \cdot 0.8 = 0.59$$

11.3.3. Gemeinsame Entropie / Verbundentropie

$$\begin{aligned} \mathrm{H}(X,Y) = &- \sum_{i}^{m} \sum_{j}^{n} p(x_i, y_j) \cdot \log_2 p(x_i, y_j) \\ &- \sum_{i}^{m} \sum_{j}^{n} p(x_i) \cdot p(x_i \mid y_j) \cdot \log_2 (p(x_i) \cdot p(x_i \mid y_j)) \end{aligned}$$

Beispiel *56:*

$$\begin{aligned} \mathrm{H}(X,Y) &= -(0.3 \cdot 0.9 \cdot \log_2(0.3 \cdot 0.9) + 0.3 \cdot 0.1 \cdot \log_2(0.3 \cdot 0.1) \\ &\quad +0.7 \cdot 0.2 \cdot \log_2(0.7 \cdot 0.2) + 0.7 \cdot 0.8 \cdot \log_2(0.7 \cdot 0.8)) \\ &\approx 1.527339244 \end{aligned}$$

11.3.4. Entropie am Kanaleingang

$$\mathrm{H}(X) = -\sum_{i=1}^{m} p(x_i) \cdot \log_2 p(x_i)$$

Beispiel *57:*

$$\begin{aligned} \mathrm{H}(X) &= -(0.3 \cdot \log_2(0.3) + 0.7 \cdot \log_2(0.7)) \\ &\approx 0.8812908992 \text{ bit/Zeichen} \end{aligned}$$

11.3.5. Entropie am Kanalausgang

Durchschnittliche Unsicherheit der empfangenen Symbole.

$$\mathrm{H}(Y) = -\sum_{j=1}^{n} p(y_j) \cdot \log_2 p(y_j)$$

***Beispiel** 58:*

$$\begin{aligned}\mathrm{H}(Y) &= -(0.41 \cdot \log_2(0.41) + 0.59 \cdot \log_2(0.59)) \\ &\approx 0.9765004688 \text{ bit/Zeichen}\end{aligned}$$

11.3.6. Äquivokation vs. Irrelevanz

Äquivokation	***Irrelevanz***
Empfänger $\rightarrow$ Sender	Sender $\rightarrow$ Empfänger
$\mathrm{H}(X\|Y)$: Wie unsicher ist das gesendete Signal X, obwohl das empfangene Signal Y bekannt ist?	$\mathrm{H}(Y\|X)$: Wie unsicher ist das empfangene Signal Y, obwohl das gesendete Signal X bekannt ist?
beschreibt den **Verlust an Information**	beschreibt das **Rauschen im Kanal**
ein Ausgang kann von **mehreren Ursachen stammen**	ein Eingang führt zu **mehreren möglichen Ausgängen**

11.3.7. Äquivokation

$\mathrm{H}(X|Y)$ misst die verbleibende Unsicherheit über das tatsächliche X, nachdem Y bekannt ist. Auch ***Rückschlussentropie*** genannt. Ist der Kanal fehlerfrei, so ist $\mathrm{H}(X|Y) = 0$.

$$\begin{aligned}\mathrm{H}(X|Y) &= -\sum_{i}^{m}\sum_{j}^{n} p(x_i, y_j) \cdot \log_2 p(x_i|y_j) \\ &= -\sum_{i}^{m}\sum_{j}^{n} p(x_i) \cdot p(x_i|y_j) \cdot \log_2 p(x_i|y_j) \\ &= \mathrm{H}(X, Y) - \mathrm{H}(Y)\end{aligned}$$

***Beispiel** 59:*

$$\begin{aligned}\mathrm{H}(X|Y) = &-(0.3 \cdot 0.9 \cdot \log_2(0.9) + 0.3 \cdot 0.1 \cdot \log_2(0.1) \\ &+0.7 \cdot 0.2 \cdot \log_2(0.2) + 0.7 \cdot 0.8 \cdot \log_2(0.8)) \\ \approx &\ 0.6460483445\end{aligned}$$

11.3.8. Irrelevanz

$\mathrm{H}(Y|X)$ misst die verbleibende Unsicherheit über das Empfangssignal Y, obwohl das gesendete Signal X bekannt ist. Auch ***Streuentropie*** genannt. Ist der Kanal fehlerfrei, so ist $\mathrm{H}(Y|X) = 0$.

$$\begin{aligned}\mathrm{H}(Y|X) &= -\sum_{i}^{m}\sum_{j}^{n} p(x_i, y_j) \cdot \log_2 p(y_j|x_i) \\ &= -\sum_{i}^{m}\sum_{j}^{n} p(x_i) \cdot p(y_j|x_i) \cdot \log_2 p(y_j|x_i) \\ &= \mathrm{H}(X, Y) - \mathrm{H}(X)\end{aligned}$$

Beispiel *60:*

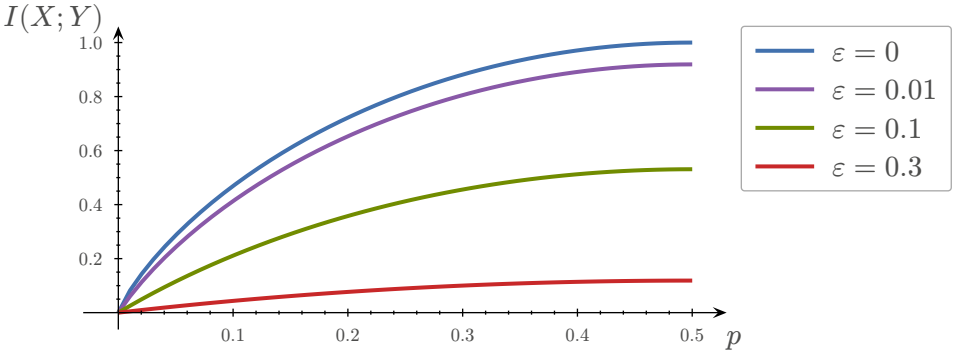
$$\begin{aligned}\mathrm{H}(Y|X) &= -(0.3 \cdot 0.9 \cdot \log_2(0.9) + 0.3 \cdot 0.1 \cdot \log_2(0.1) \\ &\quad +0.7 \cdot 0.2 \cdot \log_2(0.2) + 0.7 \cdot 0.8 \cdot \log_2(0.8)) \\ &\approx 0.6460483445\end{aligned}$$

11.3.9. Transinformation

Auch ***gegenseitige Information*** genannt. Gibt den maximalen und somit fehlerfreien Informationsfluss über einen gestörten Kanal an.

- Bei vollständig gestörtem Kanal ist $T = 0$, d.h. $\mathrm{H}(Y) = \mathrm{H}(Y|X) = 1$ und zwar unabhängig von der Entropie am Kanaleingang.
- Verändert sich die Entropie der Quelle, so verändert sich auch die Transinformation.
- Nimmt die Fehlerwahrscheinlichkeit zu, so verringert sich die Transinformation.
- Ein nicht gestörter Kanal (Einheitsmatrix) überträgt den mittleren Informationsfluss ohne weiteren Verlust, d.h. die Transinformation wird nur durch die Quelle bestimmt.

$$\begin{aligned}I(X;Y) &= \text{Ursprüngliche Information} - \text{Verlust} = \mathrm{H}(X) - \mathrm{H}(X|Y) \\ &= \text{Empfang} - \text{Rauschen} = \mathrm{H}(Y) - \mathrm{H}(Y|X) \\ &= \sum_i^m \sum_j^n p(x_i, y_j) \cdot \log_2\left(\frac{p(y_j|x_i)}{p(y_j)}\right) \\ &= \sum_i^m \sum_j^n p(x_i, y_j) \cdot \log_2\left(\frac{p(x_i|y_j)}{p(x_i)}\right)\end{aligned}$$



Beispiel *61:*

$$I(X;Y) = \mathrm{H}(Y) - \mathrm{H}(Y|X) = 0.9765004688 - 0.6460483445 = 0.3304521243$$

11.3.10. Maximale Symbolrate

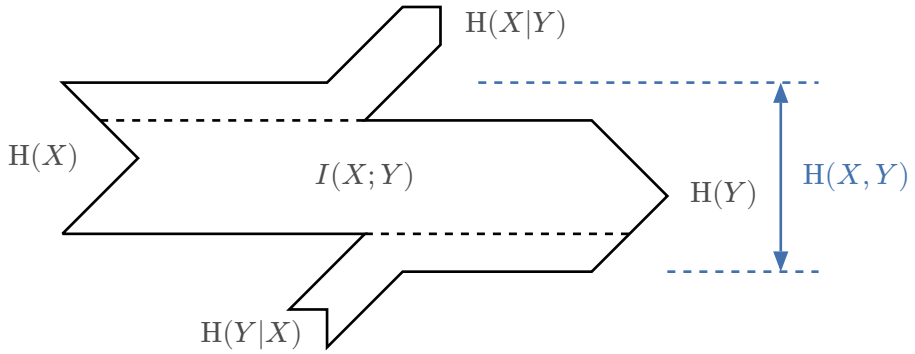
$$\begin{aligned}R_{\max}\left[\frac{\text{Zeichen}}{s}\right] &= \text{Übertragungsrate} \cdot I(X;Y) \\ &= \text{Bandbreite des Kanals} \cdot \mathrm{H}(X)\end{aligned}$$

Beispiel 62:

$$R_{\max}\left[\frac{\text{Zeichen}}{s}\right] = 0.3304521243 \cdot 1000 = 330.4521243$$

11.3.11. Zusammenhänge

11.3.11.1. Visuell



11.3.11.2. Rechnerisch

$$\begin{aligned}
\mathrm{H}(X) &\geq \mathrm{H}(X|Y) \\
\mathrm{H}(Y) &\geq \mathrm{H}(Y|X) \\
\mathrm{H}(X,Y) &= \mathrm{H}(X) + \mathrm{H}(Y|X) \\
&= \mathrm{H}(Y) + \mathrm{H}(X|Y) \\
&= \mathrm{H}(X) + \mathrm{H}(Y) - I(X;Y) \\
I(X;Y) &\geq 0
\end{aligned}$$

11.3.12. Zusammenfassung der Begriffe

Begriff	Formel	Bedeutung / Interpretation
Entropie der Quelle	$\mathrm{H}(X) = -\sum_{i}^{n} p(x_i) \cdot \log_2 p(x_i)$	Unsicherheit über das gesendete Symbol
Entropie des Ausgangs	$\mathrm{H}(Y) = -\sum_{i}^{n} p(y_i) \cdot \log_2 p(y_i)$	Unsicherheit über das empfangene Symbol
Verbundentropie	$\mathrm{H}(X,Y) = -\sum_{i}^{n}\sum_{j}^{n} p(x_i,y_j) \cdot \log_2 p(x_i,y_j)$	Gemeinsame Unsicherheit über Eingabe und Ausgabe
Äquivokation (Verlust)	$\mathrm{H}(X \mid Y) = \mathrm{H}(x,Y) - \mathrm{H}(Y)$	Unsicherheit über X, obwohl Y bekannt ist
Irrelevanz (Rauschen)	$\mathrm{H}(Y \mid X) = \mathrm{H}(X,Y) - \mathrm{H}(X)$	Unsicherheit über Y, obwohl X bekannt ist
Transinformation	$I(X;Y) = \mathrm{H}(X) - \mathrm{H}(X\|Y) = \mathrm{H}(Y) - \mathrm{H}(Y\|X)$	Tatsächlich übertragene Information

12. BLOCKCODES

Ein Blockcode C teilt das eingehende Nachrichtensignal in gleich lange Blöcke der Länge m auf und erzeugt daraus Blöcke der Länge n, wobei zusätzliche Redundanz beigefügt wird und damit $n > m$ ist. Wir sprechen immer dann von einem Blockcode, wenn alle Codewörter die gleiche Länge aufweisen.

***Beispiel** 63:*

$m = 2$
$k = 1$
$n = 3$

gültig:

$m = 2$		$k = 1$
0	0	0
0	1	1
1	0	1
1	1	0

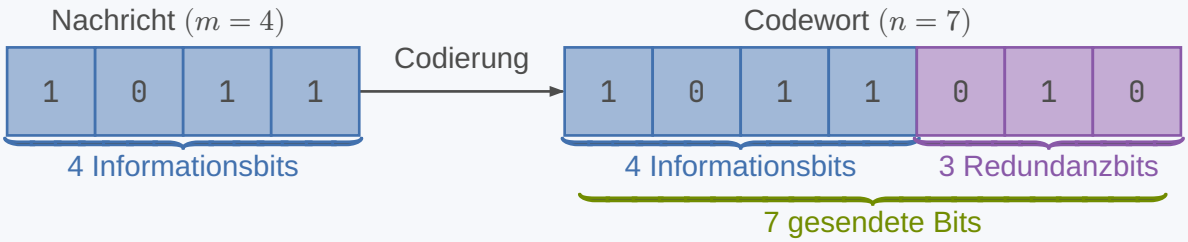
ungültig:

$m = 2$		$k = 1$
0	0	1
0	1	0
1	0	0
1	1	1

12.1. CODERATE

Die Coderate R_c eines binären (n, m)-Blockcodes C ist wie folgt definiert: $R_c = {}^{m}/_{n}$ und beschreibt, wie gross der Anteil der Nutzdaten in den Codeworten von C sind.

***Beispiel** 64:*



$$p = 0.95$$
$$\mathrm{H}(Y|X) \approx 0.286$$
$$R = \frac{4}{7} \approx 0.571$$
$$C = 1 - 0.286 = 0.714$$
$$0.571 < 0.714 \Rightarrow OK$$

12.2. LINEARE CODES

Sei F^n ein Vektorraum über einem endlichen Körper F. Die Menge $C \subseteq F^n$ heisst ***linearer** (n, k)-Code*, wenn sie ein k-dimensionaler Untervektorraum von F^n ist.

***Beispiel** 65:*

Linearer code

u	$c_1(u)$
00	000
01	011
10	101
11	110

Nicht linearer code

u	$c_2(u)$
00	000
01	001
10	010
11	100

12.2.1. Paritätscode

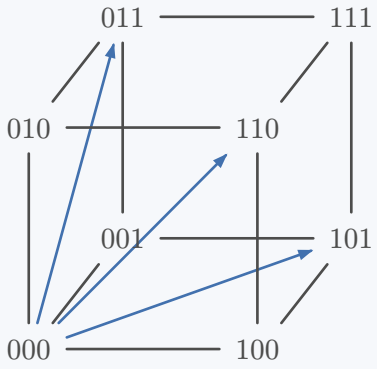
Gegeben ein Alphabet $\Sigma = \{\sigma_1, \sigma_2, ..., \sigma_{q-1}\}$ und Codewörtern der Länge n zur Basis q. Ein ***Paritätscode*** liegt vor, wenn für jedes Codewort $c_1, c_2, ..., c_n$ folgendes gilt: $c_1 + c_2 + ... + c_n \bmod q \equiv 0$

Jeder Paritätscode erkennt Einzelfehler.

***Beispiel** 66:*

Paritätscode, dargestellt als Untervektorraum von $\mathbb{Z}_2^3$

$$\left\{ \begin{pmatrix}0\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\1\end{pmatrix}, \begin{pmatrix}1\\0\\1\end{pmatrix}, \begin{pmatrix}1\\1\\0\end{pmatrix} \right\}$$



Aufgespannt durch die Basisvektoren

$$\left\{ \begin{pmatrix}1\\0\\1\end{pmatrix}, \begin{pmatrix}0\\1\\1\end{pmatrix} \right\}$$

Die Generatormatrix von c_1 ist dabei $G_1 := \begin{pmatrix}1&0&1\\0&1&1\end{pmatrix}$

12.2.1.1. 1D-Parity

Ein zusätzliches Bit macht die Gesamtzahl der 1en gerade (even parity).

Eigenschaften:
– Länge: $n = k + 1$
– Mindestabstand: $d_{\min} = 2$

Kann:
– ✓ 1-Bit-Fehler erkennen
– ✗ keinen Fehler korrigieren
– ✗ 2-Bit-Fehler nicht zuverlässig erkennen

Beispiel: $101 \to 1010$

12.2.1.2. 2D-Parity

Parität in zwei Dimensionen (Zeile + Spalte).

$$\text{Daten } r \times c \to \text{gesendet } (r+1) \times (c+1)$$

Eigenschaften:

– Produkt zweier Paritätscodes
– Mindestabstand: $d_{\min} = 4$

Kann:
– ✓ 1-Bit-Fehler erkennen
– ✓ bis 3 Bitfehler erkennen
– ✗ bestimmte 4-Bit-Fehler (Rechteck) bleiben unentdeckt

***Beispiel** 67:*

x_{11}	x_{12}	x_{13}	…	$x_1 n$	$p_{1(n+1)}$
x_{21}	x_{22}	x_{23}	…	$x_2 n$	$p_{2(n+1)}$
⋮	⋮	⋮	⋱	⋮	⋮
x_{k1}	x_{k2}	x_{k3}	…	x_{kn}	$p_{k(n+1)}$
$p_{(k+1)1}$	$p_{(k+1)2}$	$p_{(k+1)3}$	…	$p_{(k+1)n}$	$p_{(k+1)(n+1)}$

Sicher erkennbare Fehler: 3
Sicher korrigierbare Fehler: 1
Hammingdistanz: 4
(Konstant)

12.2.2. Hamming-Code

Ein Hamming-Code $H(m, k)$ ist ein linearer Fehlerkorrekturcode, der aus Datenbits zusätzliche Prüfbits bildet, sodass der Empfänger einzelne Bitfehler (und oft auch Fehler-Positionen) erkennen bzw. korrigieren kann.

Eigenschaften des binären Hamming-Codes:

Redundanz: $k \in \mathbb{N}$ (Paritätsbits)

Stellenzahl: $m = 2^k - k - 1, k \geq 2$

Gewicht: 3

Maximaldistanz: 3

Hamming-Abstand: $d_{\min} = 3$

***Beispiel** 68:* **Hamming-Blockcode für 32 Messwerte mit Fehlererkennung**

Für 32 Messwerte werden 5 Bit benötigt ($2^5 = 32$) $\Rightarrow m \geq 5$, geht aber nicht, da Hamming blockcodes $m = 2^k - k - 1$ benötigen.

$$\begin{aligned} k = 3: \quad & m = 2^3 - 3 - 1 = 4 \\ k = 4: \quad & m = 2^4 - 4 - 1 = 11 \\ & \Rightarrow k = 4 \end{aligned}$$
$$n = m + k \Rightarrow n = 11 + 4 = 15$$

Somit benötigt man einen $(11, 4)$-Hamming-Blockcode.

Die Hammingdistanz für diesen Code ist 3, wie das für alle Hamming codes der Fall ist. Die Anzahl der sicher erkennbaren Fehler ist damit $e = {}^{h-1}/_{2} = 1$. Ausserdem gilt:

$$2^m \cdot \sum_{w=0}^{e} \binom{n}{w} = 2^{11} \cdot \sum_{w=0}^{1} \binom{15}{w} = 2^{11} \cdot \left(\binom{15}{0} + \binom{15}{1} \right) = 32768 = 2^{15} = 2^n$$

Der Code ist also dichtgepackt.

12.2.3. Zyklische Codes

Ein Code heisst ***zyklisch***, wenn jede zyklische Verschiebung eines Codeworts ebenfalls ein Codewort ist.

***Beispiel** 69:*

$$1001010 \in C$$
$$\Rightarrow 0100101 \in C$$

12.2.3.1. Generatorpolynom

***Definition** 10:* **Generatorpolynom**

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

$$C = \{v(x) \mid \deg v(x) < n \wedge g(x) \mid v(x)\}$$

$g(x)$ heisst das ***Generatorpolynom*** von C.

Eigenschaften:
- Das Nullpolynom ist ein Vielfaches eines jeden Generatorpolynoms $\Rightarrow$ Die Symbolsequenz $00...0$ ist immer ein Codewort.
- Anhand der Codewörter kann das Generatorpolynom abgelesen werden: Codewörter als Binärzahlen interpretieren und nach der kleinsten Zahl ungleich 0 suchen. Die Ziffern dieser Zahl sind die Koeffizienten des Generatorpolynoms
- Verschiebt man die Koeffizienten des Generatorpolynoms um i Positionen nach links, ohne dabei die Wortlänge n zu überschreiten, so erhaltet man wieder ein Codewort. Formal entstehen diese Codewörter durch die Multiplikation von $g(x)$ mit den Polynomen x_i.
- Die ***Zykluslänge*** des Generatorpolynoms ist $2^r - 1$, wobei r der Grad des Polynoms ist und meint die Periode der von ihm erzeugten Sequenz. (Anscheinend nicht ganz korrekt, bei der Prüfung: einfach $2^{r-1} - 1$ rechnen, falls polynom reduzibel)

12.2.3.2. Codierung

Ziel: Gegeben eines empfangenen Codeworts $c(x)$ herausfinden, ob es gültig ist oder nicht.

Ansatz: Polynom $g(x)$ (Generator) wählen, welches alle gültigen Codewörter erzeugt

Beispiel: $g(x) = x^3 + x + 1$ definiert den Körper $GF(2^3)$
- Gültig: $c(x) = m(x) \cdot g(x)$. Für 1-Bit-Fehlerkorrektur wissen wir: $1 + n \leq 2^k$, wobei
  - $k = \deg(g(x))$
  - $n = \deg(c(x)) + 1$
  - $m = \deg(m(x)) + 1$
  - $n = m + k = 2^k - 1$
- Prüfung: $c(x) \bmod g(x) = 0$

Wichtig: Körperstruktur wird vorausgesetzt, da jedes Element ein Inverses haben muss, damit Division garantiert möglich wird. Ebenfalls muss Abgeschlossenheit gilten, die in dem Fall durch $\bmod F(x)$ gegeben ist.

12.2.3.3. Codebedingungen

Mit $g_i = \{0,1\} \wedge g_0 = g_k = 1$ definieren

$$\left.\begin{array}{ll} \text{Generatorpolynom} & g(u) = \sum_{i=0}^{k} g_i \cdot u^i \\ \text{Codewortpolynom} & c(u) = \sum_{i=0}^{n} g_i \cdot u^i \end{array}\right\} \text{Codebedingung}$$

Das Codewortpolynom $c(u)$ ist ohne Rest durch das Generatorpolynom $g(u)$ teilbar (in $\mathbb{Z}_2$) d.h. es gibt ein $g(u)$ aber keinen Rest

$$\frac{c(u)}{g(u)} \equiv q(u) \bmod 2$$
$$c(u) \equiv q(u) \cdot g(u) \bmod 2$$

12.2.3.4. Cyclic Redundancy Check (CRC)

CRC ist ein Fehlererkennungsverfahren, um die Integrität von Daten zu überprüfen. CRC-Codes nutzen ein Generatorpolynom, welches zur Konstruktion eines spezifischen CRC verwendet wird. Die Basis für CRC ist eine Polynomdivision, wobei der Rest der Division das CRC darstellt.

Um Redundanz (Prüfsumme) hinzuzufügen nimmt man ein Generatorpolynom $g(x)$ und erweitert es um $x^1 + x^0$.

***Beispiel** 70:*

$$g(x) = x^4 + x + 1$$
$$\Rightarrow CRC(x) = (x^4 + x + 1) \cdot (x + 1) = x^5 + x^4 + x^2 + 1$$

CRC-Codes haben eine Hammingdistanz von $d_{\min} = 4$. Durch die Multiplikation mit $x + 1$ wird nämlich das Grad des Polynoms um 1 erweitert, also wird die Hammingdistanz des herkömmlichen Hamming-Codes (3) um 1 grösser.

Der Grad des höchsten CRC-Generatorpolynom bestimmt die Anzahl der Kontrollstellen k.

12.2.3.5. Ermittlung der Kontrollstellen durch Mehrfachaddition

$c(u)$ ist durch $g(u) \bmod 2$ teilbar, also muss $c(u)$ durch Addition von $g(u) \bmod 2$ erzeugbar sein!

***Beispiel** 71:*

$m = 4$
$k = 3$
$n = 7$

$(x_1, x_2, x_3, x_4) = (1, 0, 0, 0)$

$g(u) = u^3 + u^1 + u^0$

$(g_1, g_2, g_3, g_4) = (1, 0, 1, 1)$

u^6	u^5	u^4	u^3	u^2	u^1	u^0
1	0	0	0	1	0	1
1	0	1	1			
$\oplus$	0	1	1			
		1	0	1	1	
		$\oplus$	1	1	1	
			1	0	1	1
			$\oplus$	1	0	1

Verfahren: Wort mit Generator-Vektor XOR'en (bzw $\bmod$), bis alle Zeichen auf 0 sind

$\Rightarrow$ Kontrollstellen = 101
Codewort = 1000101

12.2.3.7. Prüfen der Codebedingung

Durch die Codebedingung muss die fortgesetzte Addition ($\mathrm{mod}\,2$) des Generators zum empfangenen CW (entspricht der Division – siehe vorher) das Nullwort ergeben.

***Beispiel** 72:*

u^6	u^5	u^4	u^3	u^2	u^1	u^0
1	0	0	0	1	0	1
1	0	1	1			
		1	0	1	1	
			1	0	1	1
0	0	0	0	0	0	0

u^6	u^5	u^4	u^3	u^2	u^1	u^0
1	0	0	1	1	0	1
1	0	1	1			
		1	0	1	1	
0	0	0	0	0	1	1

Fehlersyndrom ↑

Verfahren: Codewort mit Generator addieren und mod rechnen.

Falls Resultat 0 ist, ist das Codewort Fehlerfrei.

12.2.3.8. Kontrollmatrix für zyklischen Hamming-Code

***Beispiel** 73:* 1000101

$$\begin{array}{r} 0\,0\,0\,0\,1\,0\,1 \\ \hline 0\,0\,0\,0\,1\,0\,1 \end{array}$$

$$\begin{array}{r} 1\,1\,0\,0\,1\,0\,1 \\ 1\,0\,1\,1 \\ \quad 1\,0\,1\,1 \\ \quad\quad 1\,0\,1\,1 \\ \hline 0\,0\,0\,0\,1\,1\,1 \end{array}$$

$$\begin{array}{r} 1\,0\,1\,0\,1\,0\,1 \\ 1\,0\,1\,1 \\ \quad\quad 1\,0\,1\,1 \\ \hline 0\,0\,0\,0\,1\,1\,0 \end{array}$$

$$\begin{array}{r} 1\,0\,0\,1\,1\,0\,1 \\ 1\,0\,1\,1 \\ \quad\quad 1\,0\,1\,1 \\ \hline 0\,0\,0\,0\,0\,1\,1 \end{array}$$

$$\begin{array}{r} 1\,0\,0\,0\,0\,0\,1 \\ 1\,0\,1\,1 \\ \quad 1\,0\,1\,1 \\ \quad\quad 1\,0\,1\,1 \\ \hline 0\,0\,0\,0\,1\,0\,0 \end{array}$$

$$\begin{array}{r} 1\,0\,0\,0\,1\,1\,1 \\ 1\,0\,1\,1 \\ \quad 1\,0\,1\,1 \\ \quad\quad 1\,0\,1\,1 \\ \hline 0\,0\,0\,0\,0\,1\,0 \end{array}$$

$$\begin{array}{r} 1\,0\,0\,0\,1\,0\,0 \\ 1\,0\,1\,1 \\ \quad 1\,0\,1\,1 \\ \quad\quad 1\,0\,1\,1 \\ \hline 0\,0\,0\,0\,0\,0\,1 \end{array}$$

$$\text{Kontrollmatrix}$$

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

12.2.3.9. Generatormatrix für zyklischen Hamming-Code

Erzeugung der Generatormatrix geschieht durch zyklische Verschiebung von $g(x)$ (entspricht Multiplikation mit x)

***Beispiel** 74:* $g(x) = x^3 + x + 1 \rightarrow (1, 0, 1, 1)$

$$g = 1011000$$

$$G = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

Vorgehen:
- Länge auf $n = 7$ Auffüllen (g)
- $m = 4$ unhabhängige Zeilen durch zyklische Verschiebungen erzeugen

12.2.3.10. Codewort mit Generatormatrix erzeugen

Codewort entsteht durch $c = m \cdot g$

***Beispiel** 75:*

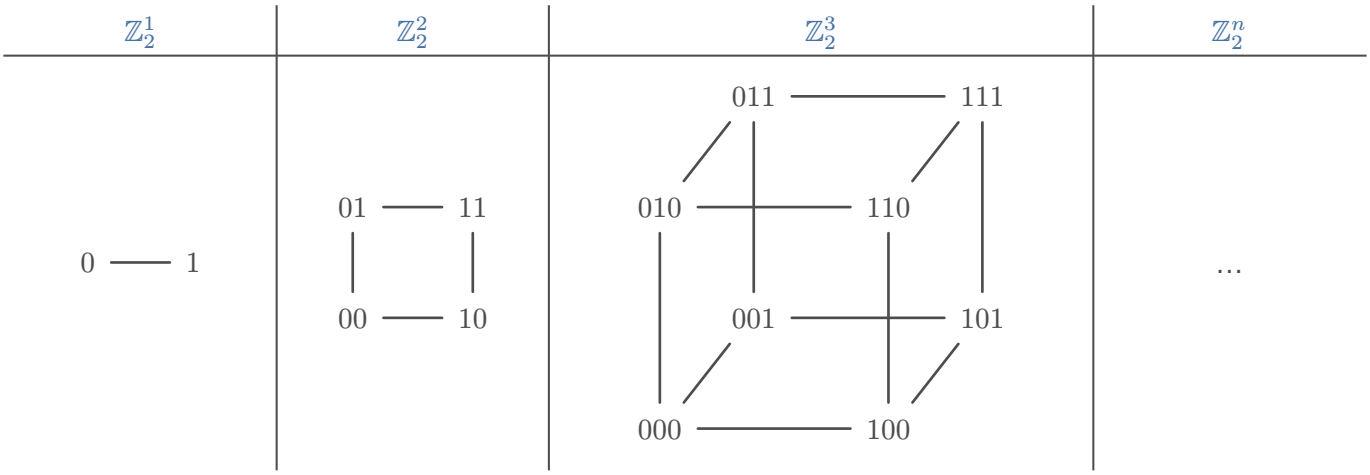
$$m = \left(\underbrace{1}_{Z_1}, 0, \underbrace{1}_{Z_2}, \underbrace{1}_{Z_3} \right)$$

$$G = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

$$\begin{aligned} Z_1 &= 1011000 \\ Z_2 &= 0010110 \\ Z_3 &= 0001011 \\ &\oplus \\ c &= 1000101 \end{aligned}$$

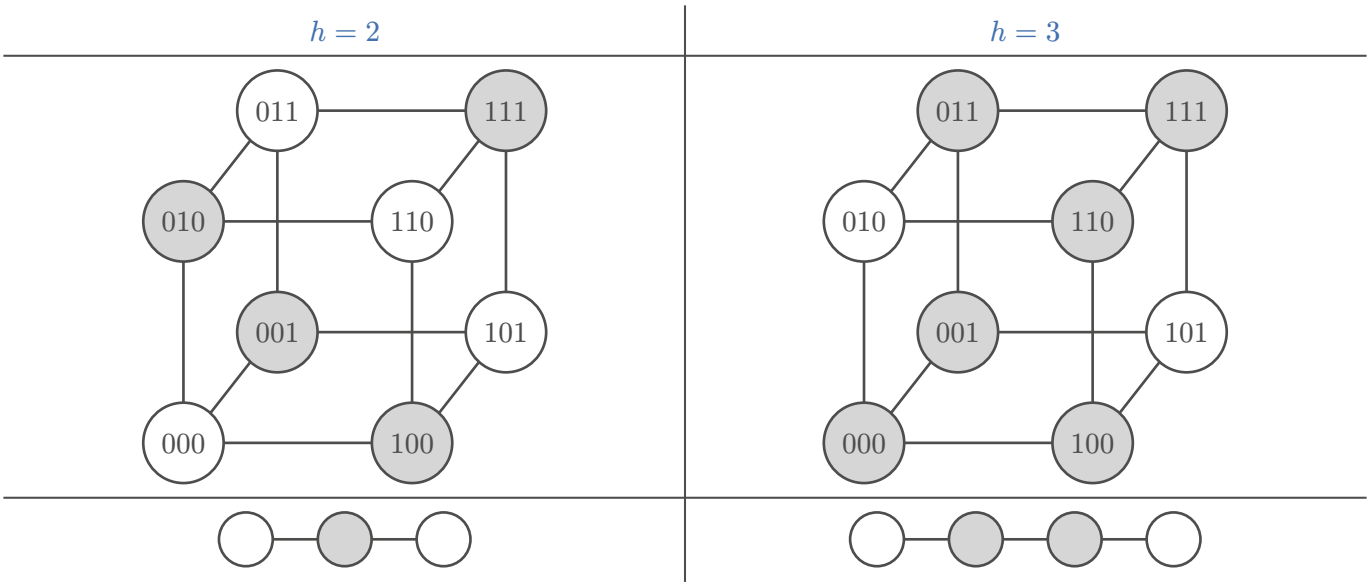
13. FEHLERERKENNUNG UND FEHLERKORREKTUR

13.1. CODERAUM



13.1.1. Gültige Wörter

Weiss = gültig, Grau = ungültig



13.1.2. Bedeutung des Coderaums

– Jeder Punkt entspricht einem möglichen Codewort

– Abstände zwischen Punkten = Hamming-Distanz

⇒ Je grösser der Abstand zwischen gültigen Codewörtern, desto robuster gegenüber Fehlern

13.1.3. Zentrale Idee

– Fehler verschieben ein Codewort im Raum
– Dekodierung = Zuordnung zum nächstgelegenen gültigen CW

13.2. HAMMING-GEWICHT UND HAMMING-DISTANZ

Definition 11: **Hamming-Gewicht**

Das ***Hamming-Gewicht*** $\omega(c)$ eines Codewortes c entspricht der Anzahl Elemente, welche im Codevektor ungleich 0 sind. Bei einem binären Code C entspricht das Hamming-Gewicht somit der Anzahl 1 im Codewort c.

Definition 12: **Hamming-Distanz**

Die ***Hamming-Distanz*** misst die Anzahl unterschiedlicher Positionen zwischen zwei gleich langen Codewörtern. Sie wird genutzt, um die Fähigkeit eines Codes zur Fehlererkennung und -korrektur zu bewerten.

$$h(v_i, v_j) = \Delta(v_i, v_j)$$

Definition 13: **Code-Distanz**

Für einen Code C fester Länge ist die Code-Distanz ΔC die minimale Distanz zwischen zwei Codewörtern.

$$\Delta C = \min\{\Delta(v_1, v_2) \mid v_1, v_2 \in C, v_1 \neq v_2\} = d_{\min}$$

Beispiel 76: **Kleinste Distanz** $h = 2$

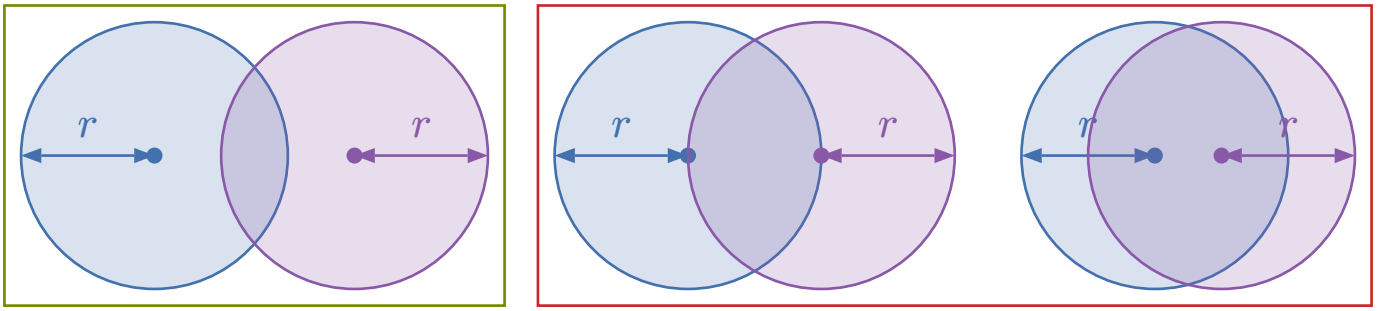
```
00000000
11000000
10001100
01010000
01010101
10000110
11111111
```

13.3. FEHLERERKENNUNG

Anzahl der sicher erkennbaren Fehler: $e^* = d_{\min} - 1$

C ist r-fehlererkennend $\Leftrightarrow d_{\min} > r$

OK NOK



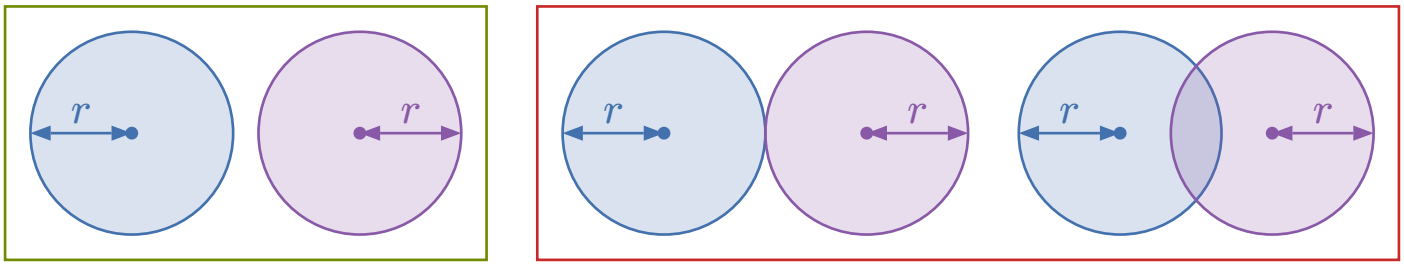
13.4. FEHLERKORREKTUR

Anzahl der sicher korrigierbaren Fehler: $e = \left\lfloor \frac{d_{\min}-1}{2} \right\rfloor$

$d_{\min} \geq 2e + 1$ notwendig für eindeutige Fehlerkorrektur

C ist r-fehlerkorrigierend $\Leftrightarrow d_{\min} > 2r$

OK NOK



13.4.1. Korrigierkugel / Hamming-Kugel

Illustriert die Fehlertoleranz eines Codes, und zeigt visuell, wie viele Fehler innerhalb eines Codewortes korrigiert werden können.

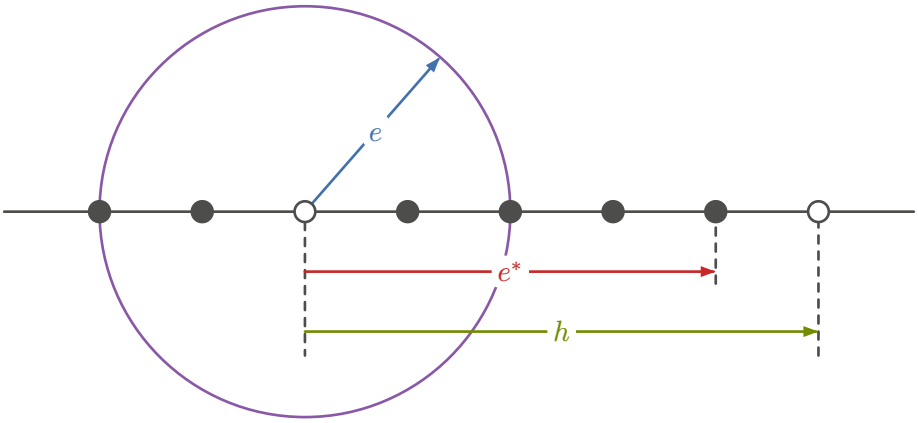
***Definition** 14:* **Hamming-Kugel**

Es seien $C \subset \Pi^n$ ein Code fester Länge, $v \in C$ und $e \in \mathbb{N}$. Die Menge

$$K_e(v) = \{w \in C \mid \Delta(v, w) \leq e\}$$

wird als ***Hamming-Kugel*** von v mit dem Radius e bezeichnet.

- Zentrum der Korrigierkugel: Liegt bei jedem gültigen Codewort.
- Radius der Korrigierkugel e: Maximale Anzahl an Fehlern, die sicher korrigiert werden können.
- Hammingdistanz h: Minimale Anzahl von Stellen, in denen sich zwei beliebige gültige Codewörter unterscheiden. Sie dient als Mass für die Fehlererkennungs- und -korrekturfähigkeit eines Codes.
- Sicher erkennbare Fehler e^*: Maximale Anzahl von Fehlern, die sicher erkannt werden können.
- Ungültige Codeworte: Liegen innerhalb der Korrigierkugel und stellen mögliche fehlerhafte Zustände des Codeworts dar, die zu einem gültigen Codewort korrigiert werden können.



Beispiel *77:*

$$\begin{aligned}
\Pi &= \{0,1\} \\
\Pi^3 &= \{000,001,010,011,100,101,110,111\} \\
K_0(001) &= \{001\} \\
K_1(001) &= \{001,000,011,101\} \\
K_2(001) &= \{001,000,011,101,010,100,111\} \\
K_3(001) &= \{001,000,011,101,010,100,111,110\} \\
K_k(001) &= K_3(001) \qquad k > 3
\end{aligned}$$

13.4.2. Dichtgepacktheit eines Codes

m = Nachrichtenstellen (Anzahl Spalten der Matrix ohne Einheitsmatrix)

k = Kontrollstellen (Anzahl Spalten der Einheitsmatrix)

$n = m + k$ = Dimension des Codes (Anzahl Spalten der Matrix)

2^m = Anzahl gültigen Codewörter

2^n = Anzahl möglicher Codewörter

Definition *15:* **Hamming-Schranke**

Die ***Hamming-Schranke*** besagt:

$$\underbrace{2^m}_{\text{Anzahl gültige CW}} \cdot \underbrace{\sum_{w=0}^{e} \binom{n}{w}}_{\text{Anzahl CW pro Korrigierkugel}} \leq \underbrace{2^n}_{\text{Anzahl aller CW}}$$

Interpretation:
- Lücken → ineffizient
- Exakt gefüllt → optimal

Definition *16:* **Dichtgepackter Code**

Gilt

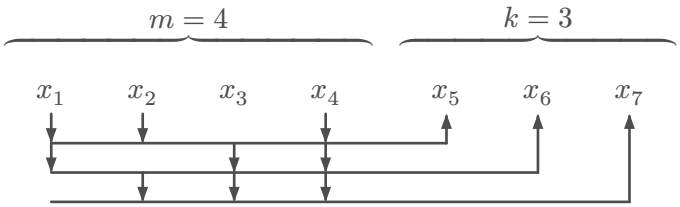
$$2^m \cdot \sum_{w=0}^{e} \binom{n}{w} = 2^n$$

so ist der Code ***dichtgepackt***

Interpretation: Ein code ist dichtgepackt, wenn sich alle gültigen und ungültigen Codeworte in einer Korrigierkugel befinden.

13.5. KONTROLLMATRIX UND CODEBEDINGUNG

Die Kontrollmatrix besitzt die Eigenschaft, dass das Syndrom eines Vektors, in dem ein einziges Bit verfälscht wurde (wo also eine 1 steht), mit der Position des defekten Bits identisch ist.



$$x_5 = (x_1 + x_2 + x_4) \bmod 2$$
$$x_6 = (x_1 + x_3 + x_4) \bmod 2$$
$$x_7 = (x_2 + x_3 + x_4) \bmod 2$$

Kontrollmatrix H:

$$\left(\begin{array}{cccc|ccc} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right)$$

$$\vec{P}_1 \vec{P}_2 \vec{P}_3 \vec{P}_4 \vec{P}_5 \vec{P}_6 \vec{P}_7$$

Codebedingung:

$$\sum_i x_i \cdot \vec{P}_i \equiv \vec{0} \bmod 2$$

***Beispiel** 78:* **Berechnung der Kontrollstellen und Fehlersyndrome**

$$\text{Hamming blockcode} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\text{Codewort} = 10110000010$$

$$\begin{aligned} x_{12} &\equiv (x_1 + x_2 + x_3 + x_4 + x_6 + x_7 + x_8) \bmod 2 \\ &\equiv (1 + 0 + 1 + 1 + 0 + 0 + 0) \equiv 1 \\ x_{13} &\equiv (x_1 + x_2 + x_3 + x_5 + x_6 + x_9 + x_{10}) \bmod 2 \\ &\equiv (1 + 0 + 1 + 0 + 0 + 0 + 1) \equiv 1 \\ x_{14} &\equiv (x_1 + x_2 + x_4 + x_5 + x_7 + x_9 + x_{11}) \bmod 2 \\ &\equiv (1 + 0 + 1 + 0 + 0 + 0 + 0) \equiv 0 \\ x_{15} &\equiv (x_1 + x_3 + x_4 + x_5 + x_8 + x_{10} + x_{11}) \bmod 2 \\ &\equiv (1 + 1 + 1 + 0 + 0 + 1 + 0) \equiv 0 \end{aligned}$$

$$\text{Kontrollstellen} = 1100$$

$$\text{Codewort} = 10110000010\textcolor{violet}{1100}$$

$$\text{Fehlersyndrome für } x_2 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}, x_{11} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}, x_2 \wedge x_{11} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

13.6. FEHLERSYNDROM

Das Fehlersyndrom wird verwendet, um die Position eines Fehlers in einem Codewort zu identifizieren. Es ist das Ergebnis der Multiplikation des empfangenen Vektors mit der transponierten Prüfmatrix. Ein Fehlersyndrom, das ungleich dem Nullvektor ist, deutet auf das Vorhandensein eines oder mehrerer Fehler im Codewort hin.

***Definition** 17:* **Syndrom**

Es sei H die Kontrollmatrix eines linearen Codes. Der Vektor

$$w \cdot H^\top$$

heisst ***Syndrom*** von w.

Für binärcodes ist das Syndrom $Z = w \cdot H^\top \bmod 2 = \sum_i w_i \cdot P_i \bmod 2$

14. FALTUNGSCODES

Ein Merkmal von Blockcodierern ist es, dass sie kein Gedächtnis besitzen. Das bedeutet, dass jeder Block immer nach der gleichen Vorschrift verarbeitet wird. 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.

Für die Klassifikation von Faltungscodierungen greifen wir auf die Tripelschreibweise

$$(k, n, l)$$

zurück, wobei den einzelnen Komponenten die folgende Bedeutung zukommt:
– k ist die Anzahl der parallel entgegengenommenen Eingabebits.
– n ist die Anzahl der parallel produzierten Ausgabebits.
– l ist die Anzahl der Schieberegisterstufen

$$R = \frac{k}{n}$$

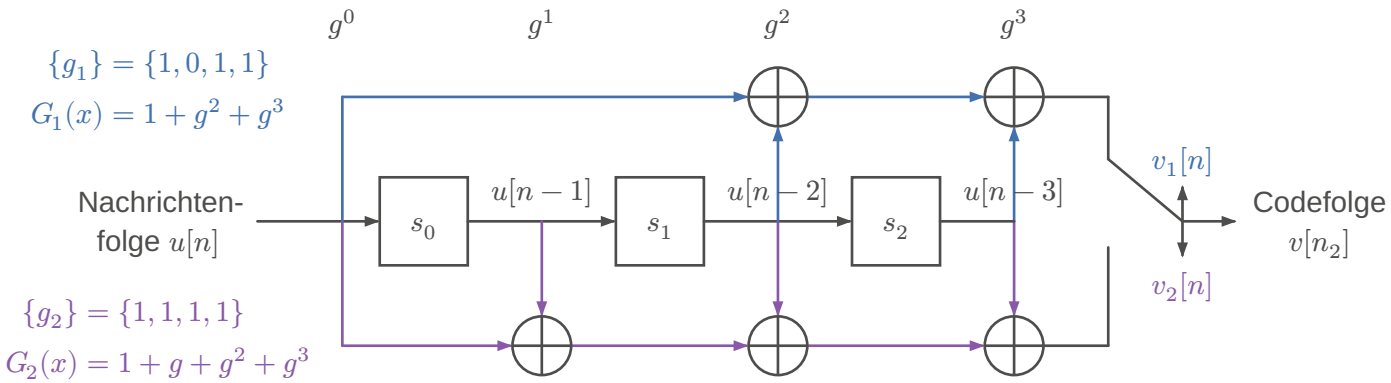
ist die Coderate des Faltungscodes: Sie gibt an, wie viele Nachrichten- bits durch ein einzelnes Codewortbit abgedeckt werden,

14.1. FALTUNG

– Generatorfolge $g[n]$
– Eingangsfolge $\{u[n]\} = \{u_1, u_2, ..., u_n\}$
– Ausgangsfolge $\{v[n]\} = \sum_{m=0}^{M} g^m u[n-m]$
– Gedächtnisfolge

Encoder «mischt» beide Folgen, Ausgabe = gewichtete Kombination der Vergangenheit

Hier zu sehen ist ein $(2, 1, 3)$ Encoder.



Beispiel 79: $\{u_n\} = \{1, 0, 1\}$

$u[n]$	s_0	s_1	s_2	$v_1[n]$	$v_2[n]$	$v[n_2]$
–	0	0	0	–	–	–
1	0	0	0	1	1	11
0	1	0	0	0	1	01
1	0	1	0	0	0	00
0	1	0	1	1	0	10
0	0	1	0	1	1	11
0	0	0	1	1	1	11
–	0	0	0	–	–	–

Speicherplätze s_0, s_1, s_2 sind mit 0 vorbelegt.

Ausgangszustand ist 110100101111

Da $G_1 = g^0 + g^2 + g^3$ ist $v_1[n] = u[n] \oplus s_1 \oplus s_2$

Da $G_2 = g^0 + g^1 + g^2 + g^3$ ist $v_1[n] = u[n] \oplus s_0 \oplus s_1 \oplus s_2$

Die Anzahl der Tailbits ergibt sich aus der Anzahl der Schieberegister (= anzahl speicherstellen, damit sie wieder mit nullen belegt sind).

Die Anzahl der Ausgangsbits entspricht der Anzahl an Schieberegister + das Eingangsbit.

Die Block-Coderate R gibt das Verhältnis der Anzahl codierter Bits zur Gesamtzahl der gesendeten Bits an. Sie ist ein Mass für die Effizienz eines Codierungsverfahrens in Bezug auf die Bandbreitenausnutzung.

$$R = \frac{n}{2 \cdot (k + m)}$$

14.2. IMPULSANTWORT

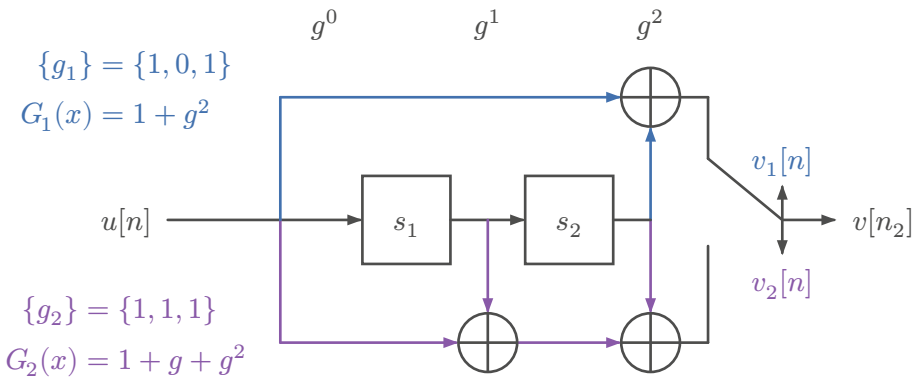
Wir testen den Encoder mit einem einzelnen Bit und beobachten, wie lange dieses Bit im System nachwirkt. Die Folge der erzeugten Ausgangsbits nennt man **Impulsantwort**.

– Da ein einzelnes Bit mehrfach in der Codefolge enthalten ist, bleiben trotz einzelner Fehler genügend Informationen erhalten. Dadurch steigt der Abstand zwischen gültigen Folgen (= freie Distanz) und Fehler können besser erkannt und korrigiert werden.
– Trotz eines Fehlers ähnelt die Folge weiterhin stärker dem ursprünglichen Verlauf als anderen möglichen Codefolgen.
– Durch die zeitliche Verteilung eines Bits entstehen charakteristische Codefolgen mit grösserem Abstand. Dadurch kann der Decoder später den wahrscheinlichsten Pfad bestimmen.

Beispiel *80:*

Eingangsimpuls: 1, 0, 0, 0, …
Impulsantwort: 11, 01, 11, 00, …
⇒ Die einzelne 1 wirkt vier Takte lang nach.

14.3. GENERATORPOLYNOM BESTIMMEN



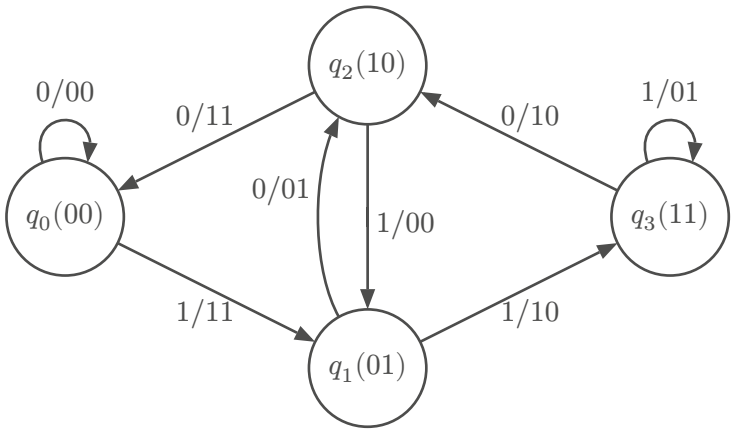
14.3.1. Generatorpolynome für optimale Faltungscodes

g_i in oktaler schreibweise

m	$R = 1/2$			$R = 1/3$				$R = 1/4$				
	g_1	g_2	d_f	g_1	g_2	g_3	d_f	g_1	g_2	g_3	g_4	d_f
2	5	7	5	5	7	7	8	5	7	7	7	10
3	15	17	6	13	15	17	10	13	15	15	17	15
4	23	35	7	25	33	37	12	25	27	33	37	16
5	53	75	8	47	53	75	13	53	67	71	75	18
6	133	171	10	133	145	175	15	135	135	147	163	20
7	247	371	10	225	331	367	16	235	275	313	357	22
8	561	753	12	557	663	711	18	463	535	733	745	24

14.3.2. Zustandsdiagramm

Eingang	s_1	s_2	*Ausgang*
g^0	g^1	g^2	$v_1[n]/v_2[n]$
0	0	0	q_0(0/0)
1	0	0	q_3(1/1)
0	1	0	q_1(0/1)
1	1	0	q_2(1/0)
0	0	1	q_3(1/1)
1	0	1	q_0(0/0)
0	1	1	q_2(1/0)
1	1	1	q_1(0/1)



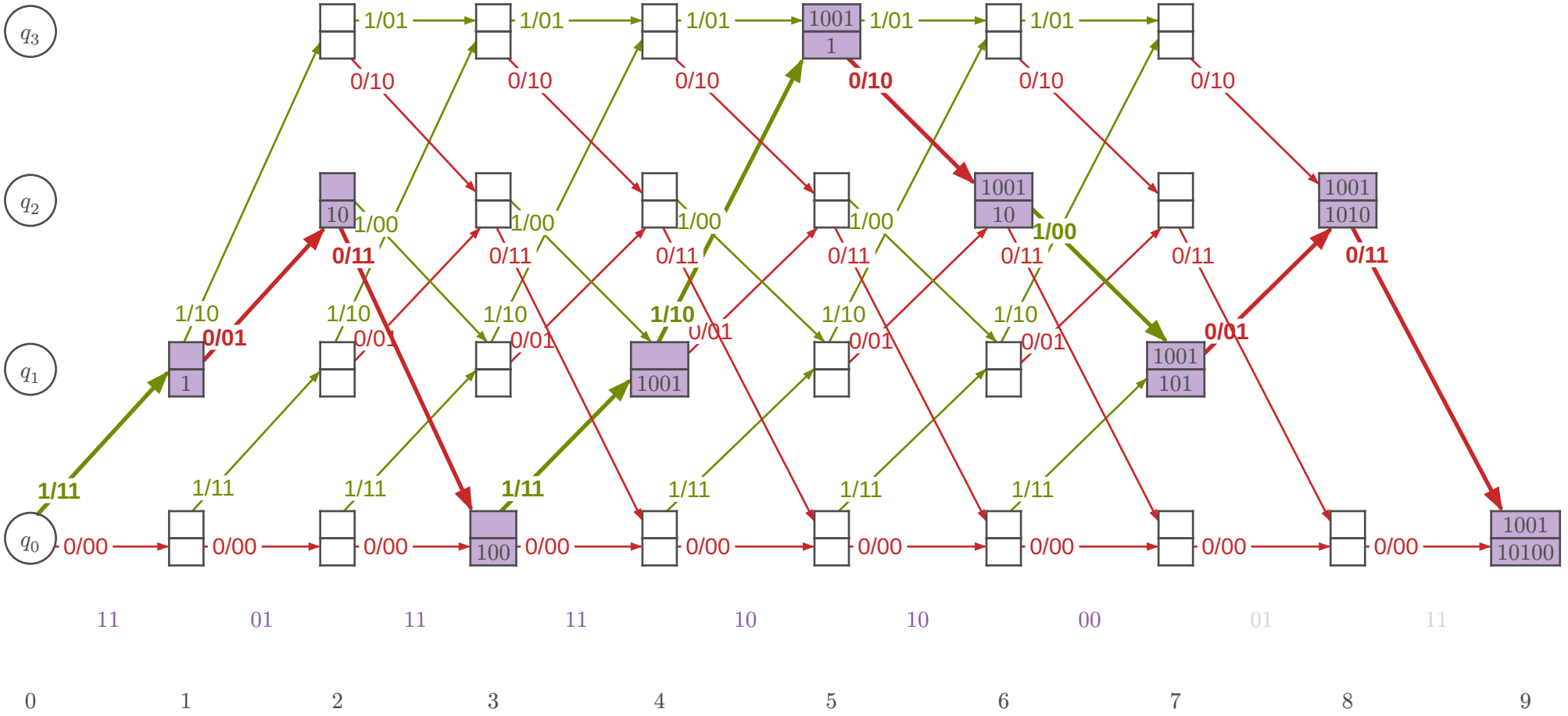
14.3.3. Codieren

$$\{u[n]\} = \{\ 1, \quad 0, \quad 0, \quad 1, \quad 1, \quad 0, \quad 1\}$$
$$q_0 \to q_1 \to q_2 \to q_0 \to q_1 \to q_3 \to q_2 \to q_1 \to q_2 \to q_0$$
$$\{v[n]\} = \{\ 11, \quad 01, \quad 11, \quad 11, \quad 10, \quad 10, \quad 00, \quad 01, \quad 11\}$$

14.3.4. Decodieren (Viterbi-Algorithmus & Trelli-Diagramm)

Input: 1101111111010000111

Output: 1001101



14.4. EIGENSCHAFTEN

- Bei einem (n, k)-Faltungscodierer ergibt sich die Coderate wie bei den Blockcodes zu $R = {}^{k}/_{n}$
- Man bezeichnet m als das Gedächtnis (Memory) des Codes und den Convolutional Code selbst mit $CC(n, k, m)$
- Daraus ergibt sich die Einflusslänge (Constraint Length) zu $\nu = m + 1$
- Für $k > 1$ gibt man diese Parameter oft auch in Bit an: $m_{\text{Bit}} = m \cdot k$ bzw. $\nu_{\text{Bit}} = (m + 1) \cdot k$

Ein binärer Faltungscode mit $m = 0$ (also ohne Gedächtnis) wäre identisch mit einem binären Blockcode.

15. QUBIT

- Superposition
  - Ein Qubit kann gleichzeitig 0 und 1 repräsentieren. Mehrere Qubits können dadurch alle 2^n möglichen Zustände gleichzeitig darstellen.
  - $|\Psi\rangle = \alpha\ |0\rangle + \beta\ |1\rangle$
  - $P(0) = |\alpha|^2, P(1) = |\beta|^2$
- Beispiel
  - $|\Psi\rangle = \sqrt{0.75}\ |0\rangle + \sqrt{0.25}\ |1\rangle$
  - $P(0) = 0.75, P(1) = 0.25$
- Interferenz
  - α, β sind Amplituden. Zwei Qubits können sich gegenseitig beeinflussen, indem die Amplituden mittels Interferenz verstärkt oder ausgelöscht werden
  - Quantenalgorithmen verändern gezielt die Amplituden:
    - richtige Lösungen werden verstärkt
    - falsche Lösungen werden abgeschwächt
- Verschränkung
  - Verschränkung bedeutet, dass zwei Qubits einen gemeinsamen Zustand besitzen, der sich nicht in zwei unabhängige Einzelzustände zerlegen lässt.
  - Misst man eines der Qubits, ist der Zustand des anderen sofort festgelegt

16. CPU

Gesamtübersicht des CPU Zyklus

1) Fetch: Instruktion aus RAM lesen
2) Decode: Opcode interpretieren
3) Operand-Fetch: Speicherwerte laden
4) Execute: ALU rechnet
5) Writeback: Ergebnis ins Register
6) Instruction Pointer (IP) erhöhen

BIBLIOGRAFIE

[1] D. Bühler, «Zusammenfassung DigCod». Zugegriffen: 3. Mai 2026. [Online]. Verfügbar unter: https://studentenportal.ch/dokumente/digcod/2828/