

Digitale Codierungen | DigCod

Zusammenfassung

1. MATHEMATISCHE GRUNDLAGEN

1.1. ZAHLENSYSTEME

Jede Stelle der Zahl hat den Wert der entsprechenden Potenz, zum Beispiel 16 (Hexadezimal). Die rechte Ziffer entspricht $16^0=1$, die zweite von rechts $16^1=16$ usw. Nimm jede Ziffer bzw. ihren Zahlenwert (A=10, B=11, ...) mal mit der entsprechenden Potenz und summiere.

1.1.1. Bit

Ein Bit ist eine Grösse, die **zwei Zustände** annehmen kann. Welchem Zustand 0 und welchem 1 entspricht, hängt dabei von Hersteller, Standards, Konventionen usw. ab.

- Drei oder mehr Zustände sind technisch aufwändiger zu realisieren und zu verarbeiten
- Direkter Zusammenhant mit **klassischer (boolescher) Logik** (wahr/falsch)
- **Einfachste Entscheidungen** im Programmfluss sind binär (ja/nein)

Das Binärsystem bildet die **Basis aller gängigen Computersysteme**.

1.1.2. Binäres Zahlensystem

Im binären Zahlensystem ist die Basis $a = 2$ und die Ziffern sind $0 \leq z_i < 2$

256 2^8	128 2^7	64 2^6	32 2^5	16 2^4	8 2^3	4 2^2	2 2^1	1 2^0	1/2 0.5 2^{-1}	1/4 0.25 2^{-2}	1/8 0.125 2^{-3}	1/16 0.0625 2^{-4}

		Näherung	Dezimalpräfix	Binärpräfix
- Bit = 1: Gesetztes Bit (set Bit)				
- Bit = 0: Gelöschtes Bit (cleared bit)	2^{10}	$1.024 \cdot 10^3$	K Kilo	Ki Kibi
- LSB: Least Significant Bit, niedrigstwertiges Bit, Bit 0	2^{20}	$1.049 \cdot 10^6$	M Mega	Mi Mebi
	2^{30}	$1.074 \cdot 10^9$	G Giga	Gi Gibi
- MSB: Most significant Bit, höchstwertigstes Bit, Bit $n - 1$ in einer n -stelligen Zahl	2^{40}	$1.100 \cdot 10^{12}$	T Tera	Ti Tebi
	2^{50}	$1.126 \cdot 10^{15}$	P Peta	Pi Pebi
- Nibble: Binärzahl mit 4 Bit. Grössere Binärzahlen werden als Nibble gruppiert, z.B. 0101	2^{60}	$1.153 \cdot 10^{18}$	E Exa	Ei Exbi
- Oktett/Byte: Eine Binärzahl mit 8 Bit oder zwei Nibbles, z.B. 0101 1000				
- Notation: Die Bits einer Binärzahl b werden häufig durch Indizes einzeln berechnet. Zusammenhängende Bereiche von Bits in einer Binärzahl werden «/» bezeichnet. $b = 1010$				

Minimal benötigte Bits zur Adressierung: $2^{n-1} < z < 2^n$

Beispiel: Wieviele Bits benötigt man, um 20MB Speicher adressieren zu können?

$$16M < 20M \leq 32M = 2^{25} \rightarrow 25 \text{ Bit}$$

1.1.3. Hexadezimals Zahlensystem

16^0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16^1	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
16^2	256	512	768	1024	1280	1536	1792	2048	2304	2560	2816	3072	3328	3584	3840
16^3	4096	8192	12288	16384	20480	24576	28672	32768	36864	40960	45056	49152	53248	57344	61440

Sonst einfach Zahl / 16, Rest ergibt den hintersten HEX Wert, dann geteilte Zahl / 16 etc.

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

1.2. AUSSAGENLOGIK

Eine **Aussage** ist ein feststellender Satz, dem eindeutig einer der beiden Wahrheitswerte wahr oder falsch zugeordnet werden kann.»

Auch wenn der Wahrheitswert unbekannt ist, können Behauptungen Aussagen sein, wenn sie im Prinzip beweis- oder widerlegbar sind. Auch Behauptungen über die Zukunft können Aussagen sein. Für Aussagen werden oft Symbole (z.B. A, B, C, ...) verwendet.

Wenn die Behauptung in eine «Wenn <Behauptung>, dann ...» Form gebracht werden kann, handelt es sich um eine Aussage.

1.2.1. Junktoren

- Negation *nicht* ⇒ Implikation *wenn ... dann ...*
- ∧ Konjunktion *und* ⇔ Äquivalenz *... genau dann, wenn ...*
- ∨ Disjunktion *oder*

1.2.2. Negation / Verneinung / ¬

Die Negation einer Aussage ist genau dann wahr, wenn die Aussage falsch ist.

- Heute regnet es ⇒ Heute regnet es nicht
- $x > 1000 \Rightarrow x \leq 1000$
- A: $x < 7 \Rightarrow \neg A: \neg(x < 7)$ oder $\neg A: x \geq 7$

A	$\neg A$	$\neg(\neg A)$
w	f	w
f	w	f

1.2.3. Konjunktion / und / \wedge

Die Konjunktion zweier Aussagen ist genau dann wahr, wenn beide Aussagen wahr sind.

- A: Heute regnet es. B: Die Lufttemperatur ist unter null. $\Rightarrow A \wedge B$: Heute regnet es und die Lufttemperatur ist unter null.
- A: $x < 3$, B: $y > 5 \Rightarrow A \wedge B$: $x < 3$ und $y > 5$

A	B	$A \wedge B$
w	w	w
w	f	f
f	w	f
f	f	f

Achtung: Mit der Konjunktion wird kein kausaler oder zeitlicher Zusammenhang berücksichtigt.

In der Aussagenlogik gilt $A \wedge B \Leftrightarrow B \wedge A$

1.2.4. Disjunktion / (einschliessliches) ODER / \vee

Die Disjunktion zweier Aussagen ist genau dann wahr, wenn mindestens eine der beiden Aussagen wahr ist.

- A: Heute regnet es. B: Die Lufttemperatur ist unter null. $\rightarrow A \vee B$: Heute regnet oder und die Lufttemperatur ist unter null.
- A: $x < 3$, B: $y > 5 \rightarrow A \vee B$: $x < 3$ oder $y > 5$ oder beides

A	B	$A \vee B$
w	w	w
w	f	w
f	w	w
f	f	f

Eine **Kontradiktion** ist eine logische Aussage, die nie wahr ist. Alle Zeilen der Wahrheitstafel ergeben falsch. Eine **Tautologie** ist eine logische Aussage, die immer wahr ist. Alle Zeilen der Wahrheitstafel ergeben wahr.

1.2.5. Implikation / wenn x, dann y / \Rightarrow

Wenn die Aussage A wahr ist, dann gilt auch die Aussage B. Aus A folgt B.

A ist hinreichend für B. B ist notwendig für A. A ist Voraussetzung / Annahme, B ist die Folgerung.

A	B	$A \Rightarrow B$
w	w	w
w	f	f
f	w	w
f	f	w

Abtrennungsregel: Wenn A wahr ist, und $A \Rightarrow B$ wahr ist, dann ist auch B wahr. $(A \wedge (A \Rightarrow B)) \Rightarrow B$

1.2.6. Äquivalenz / y genau dann, wenn x / \Leftrightarrow

Aussage A gilt genau dann, wenn B gilt. Es gilt $A \Rightarrow B$ und $B \Rightarrow A$

A	B	$A \Leftrightarrow B$
w	w	w
w	f	f
f	w	f
f	f	w

Hinweis: ein senkrechter Strich $x \mid y$ bedeutet x ist ein Teiler von y

1.2.7. Bindungsstärke

Die Bindungsstärke legt fest, welche Junktoren in aussagenlogischen Formeln zuerst ausgeführt werden. Das **Nicht** hat die grösste Bindungsstärke, gefolgt von den beiden Junktoren **Und** und **Oder**. Die kleinste Bindungsstärke besitzen die **Implikationen** und die **Äquivalenz**

1.3. AUSSAGENLOGISCHE FORMELN

A und B seien Aussagen. Die verknüpften Aussagen $\neg A$, $A \wedge B$, $A \vee B$, $A \Rightarrow B$ und $A \Leftrightarrow B$ sind **aussagenlogische Formeln**. Die verknüpften Aussagen sind auch aussagenlogische Formeln, wenn A oder B bereits selbst aussagenlogische Formeln sind. Eine aussagenlogische Formel ist selbst wieder eine Aussage. Die logische Konstante W bedeutet «wahr» und F bedeutet «falsch».

SATZ 1: A, B, und C seien aussagenlogische Formeln. Dann gilt

1. Kommutativität: $A \wedge B \Leftrightarrow B \wedge A$, $A \vee B \Leftrightarrow B \vee A$

Wenn A und B zutreffen, gilt auch B und A. Wenn A oder B zutreffen, gilt auch B oder A (umgekehrt)

2. Assoziativität: $A \wedge (B \wedge C) \Leftrightarrow (A \wedge B) \wedge C$, $A \vee (B \vee C) \Leftrightarrow (A \vee B) \vee C$

Wenn A und (B und C) zutreffen, können die Klammern beliebig verschoben oder weggelassen werden (gleich bei oder)

3. Distributivität: $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$, $A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$

A und (B oder C) ist das gleiche wie (A und B) oder (A und C), A oder (B und C) ist das gleiche wie (A oder B) und (A oder C)

4. Absorption: $A \vee (A \wedge B) \Leftrightarrow A$, $A \wedge (A \vee B) \Leftrightarrow A$

A oder (A und B) ist das Gleiche wie A. A und (A oder B) ist das Gleiche wie A

5. Idempotenz: $A \vee A = A$, $A \wedge A = A$

A und A = A, A oder A = A

6. Doppelte Negation: $\neg(\neg A) \Leftrightarrow \neg\neg A \Leftrightarrow A$

Nicht nicht A ist gleich A

7. Konstanten: W = wahr, F = falsch. $A \vee W \Leftrightarrow W$, $A \wedge W \Leftrightarrow A$, $A \vee \neg A \Leftrightarrow W$

A oder wahr = wahr, A und wahr = A, A oder nicht A = wahr

Implikation ersetzen: $A \Rightarrow B$ kann man durch $\neg A \vee B$ ersetzen

Äquivalenz ersetzen: $A \Leftrightarrow B$ kann man durch $(A \wedge B) \vee (\neg A \wedge \neg B)$ ersetzen

SATZ 2: A, B und C seien aussagenlogische Formeln. Dann bedeutet $A \Rightarrow B \Rightarrow C$, dass beide Implikationen, sowohl $A \Rightarrow B$ als auch $B \Rightarrow C$ gelten. Formal kann man also schreiben: $(A \Rightarrow B \Rightarrow C) \Leftrightarrow (A \Rightarrow B) \wedge (B \Rightarrow C)$

SATZ 3 (DE MORGAN): Für beliebige Aussagen A und B gilt:

a) $\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B$ Nicht (A und B) ist das gleiche wie nicht A oder nicht B

b) $\neg(A \vee B) \Leftrightarrow \neg A \wedge \neg B$ Nicht (A oder B) ist das gleiche wie nicht A und nicht B

1.4. NORMALFORMEN

Dienen der Übersichtlichkeit – standardisierte Form

1.4.1. Negationsnormalform

Eine aussagenlogische Formel steht in Negationsnormalform, wenn die Negation (\neg) nur direkt vor Aussagen oder vor Konstanten steht.

Beispiel: $\neg(A \vee B)$ ist keine Negationsnormalform, $\neg A \wedge \neg B$ hingegen schon

1.4.2. Verallgemeinerte Konjunktion

Eine verallgemeinerte Konjunktion ist...

- Eine einzelne Aussage oder seine Negation (also A oder $\neg A$) oder
- Einer der logischen Konstante T=wahr oder F=falsch oder
- Eine Konjunktion $A \wedge B$, falls A und B selbst verallgemeinerte Konjunktionen sind

Eine Verbindung von Aussagen, Negationen und logischen Konstanten mit «und».

Liegt immer in **Negationsnormalform** vor.

1.4.3. Disjunktive Normalform

Eine Aussage liegt in disjunktiver Normalform vor, wenn sie eine **verallgemeinerte Konjunktion** ist, oder wenn sie eine Disjunktion von verallgemeinerten Konjunktionen ist.

Eine Verbindung von **verallgemeinerten Konjunktionen** mit «oder».

1.4.4. Verallgemeinerte Disjunktion

Eine verallgemeinerte Disjunktion ist...

- Eine einzelne Aussage oder seine Negation (also A oder $\neg A$) oder
- Einer der logischen Konstante T=wahr oder F=falsch oder
- Eine Disjunktion $A \vee B$, falls A und B selbst verallgemeinerte Disjunktionen sind

1.4.5. Konjunktive Normalform

Eine Aussage liegt in konjunktiver Normalform vor, wenn sie eine **verallgemeinerte Disjunktion** ist, oder wenn sie eine Konjunktion von verallgemeinerten Disjunktionen ist.

Beispiel: Aussage A: $(X \wedge \neg Y) \vee (\neg X \wedge (Y \vee \neg Z))$

X	Y	Z	A
w	w	w	f
w	w	f	f
w	f	w	w
w	f	f	w
f	w	w	w
f	w	f	w
f	f	w	f
f	f	f	w

Um A in **disjunktiver Normalform** zu schreiben, werden die Zeilen der Tabelle ausgewertet, in denen A richtig ist. Um A in **konjunktiver Normalform** zu schreiben, werden die Zeilen der Tabelle ausgewertet, in denen A falsch ist.

Das heisst, die **konjunktive Normalform** wäre wie folgt:

$$A \Leftrightarrow \neg ((X \wedge Y \wedge Z) \vee (X \wedge Y \wedge \neg Z) \vee (\neg X \wedge \neg Y \wedge Z))$$

Daraus lässt sich mit der Regel von de Morgan eine **Negationsnormalform** erstellen:

$$A \Leftrightarrow (\neg X \vee \neg Y \vee \neg Z) \wedge (\neg X \vee \neg Y \vee Z) \wedge (X \vee Y \vee \neg Z)$$

1.5. BINÄRE FUNKTIONEN

x	y	0	xy	x \bar{y}	x	$\bar{x}y$	y	$\bar{y} \vee \bar{x}y$	$\bar{y} \vee xy$	$\bar{y} \vee \bar{x}y$	$\bar{y} \vee xy$	\bar{y}	$\bar{y} \vee \bar{x}$	$\bar{y} \vee y$	$\bar{x}y$	1
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

x	y	0	AND	x \bar{y}	x	$\bar{x}y$	y	XOR	OR	NOR	EQUIN	\bar{y}	$\bar{y} \uparrow x$	\bar{x}	$x \uparrow y$	NAND	1
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

- **Implikation:** if (0 \Rightarrow 1) then 0; else 1
- **NAND (not and / nicht alle wahr):** $x|y = \overline{x \wedge y} = \bar{x}\bar{y}$
Nur dann 0, wenn x=1 und y=1. Die Basisoperationen können ausschliesslich aus | dargestellt werden (Sheffer stroke).
NAND-Bausteine sind technisch leicht als Transistoren zu realisieren.
- **NOR ($\bar{x} \vee \bar{y}$):** Nur dann 1, wenn x = 0 und y = 0
- **XOR:** $x \oplus y$ Exklusive Diskunktion/Exklusives oder
- **Basis der Addition:** XOR bildet die Addition zweier Bits ab, AND den Übertrag
- **Literal:** Variable oder Negation einer Variablen, bspw. x_3 (positives Literal) oder \bar{x}_1 (negatives Literal)
- **Konjunktionsterm:** Konjunktion von Literalen, bspw. $\bar{x}_1x_3x_4 = \bar{x}_1 \wedge x_3 \wedge x_4$

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

x	y	$x \wedge y$	$x y$	$\bar{x} \wedge \bar{y}$
0	0	0	1	1
0	1	0	1	0
1	0	0	1	0
1	1	1	0	0

- **Diskunktionsterm:** Disjunktion von Literalen, bspw. $\overline{x_1} \vee x_3 \vee x_4$
- **Minterm:** Konjunktionsterm, der alle Parameter der Funktion enthält, z.B. $x_0 \overline{x_1} \overline{x_2} \overline{x_3} x_4$
- **Maxterm:** Disjunktionsterm, der alle Parameter der Funktion enthält, z.B. $x_0 \vee \overline{x_1} \vee \overline{x_2} \vee x_3 \vee x_4$
- **KDNF (kanonische disjunktive Normalform):** Disjunktion von Mintermen.

$$\overline{x} = \overline{x \wedge x} = x \mid x$$

$$x \wedge y = \overline{\overline{x} \mid \overline{y}} = (x \mid y) \mid (x \mid y)$$

$$x \vee y = \overline{\overline{x} \wedge \overline{y}} = (x \mid x) \mid (y \mid y)$$

1.6. COMPUTERARITHMETIK

1.6.1. Addition
 Überträge können bei signed verloren gehen (carry bit) oder erfordern eine $n + 1$ stellige Binärzahl

```

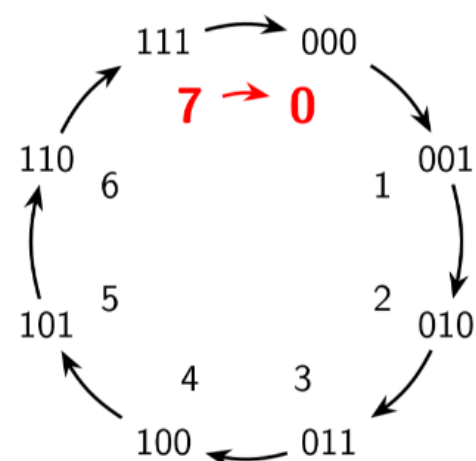
  1 0 1 1
+ 0 0 1 0
-----
  1 1 0 1
  
```

1.6.2. Subtraktion
 Ist der Minuend kleiner als der Subtrahend, erhöht man den Minuenden um 2^n auf eine $n + 1$ stellige Binärzahl:

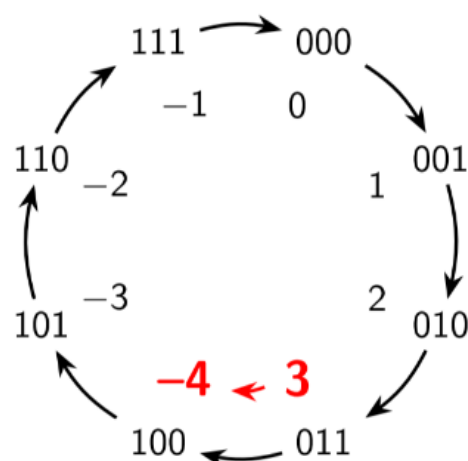
```

(1) 0 0 0 0
- (0) 0 0 0 1
-----
  1 1 1 1
  
```

Zweierkomplement



unsigned: Überlauf von 7 auf 0



signed: Überlauf von 3 auf -4

Von negativ zu positiv

$N(-3) = 3$ darstellen (nur signed)

$x = -3, 1101$

- Minus 1 rechnen, $1101 - 0001 = 1100$
- Zahl invertieren $1100 \rightarrow 0011$
- $0011 \Leftrightarrow 3$

Rechnen von $5 - 3 = 2$

- -3 darstellen, $3 = 0011, -3 = 1101$
- $-3 + 5 = 2, 1101 + 0101 = 1010$
- Im 4bit-System fällt das vorderste Zeichen weg, $5 - 3 = 2 \Rightarrow 0010$

Von positiv zu negativ

$N(3) = -3$ darstellen (nur signed)

$x = 3, 0011$

- Zahl invertieren $0011 \rightarrow 1100$
- Plus 1 rechnen $1100 + 1 = 1101$
- $1101 \Leftrightarrow -3$

Handwritten calculation on grid paper:

$$\begin{array}{r} -8 \mid 4 \mid 2 \mid 1 \\ \hline 1 \mid 1 \mid 0 \mid 1 \\ \hline \Rightarrow -8 + 4 + 1 = -3 \end{array}$$

1.6.3. Multiplikation

$$\begin{array}{r}
 110 \\
 * 101 \\
 \hline
 110 \\
 + 0000 \\
 + 11000 \\
 \hline
 11110
 \end{array}$$

$$\begin{array}{r}
 11011 \\
 * 101 \\
 \hline
 11011 \\
 + 000000 \\
 + 1101100 \\
 \hline
 10000111
 \end{array}$$

$$\begin{array}{r}
 1011 \\
 * 1111 \\
 \hline
 1011 \\
 + 10110 \\
 + 101100 \\
 + 1011000 \\
 \hline
 10100101
 \end{array}$$

1.6.4. Fehler von Kommazahlen

Beschränkte Genauigkeit: Nicht alle Zahlen sind genau darstellbar. Es sind nur ganzzahlige Vielfache von 2^{-10} darstellbar. Alle anderen Zahlen bräuchten eine unendliche periodische Darstellung und werden auf die entsprechende Anzahl Bits abgekürzt. Unflexibel und fehleranfällig, aber performant.

- **Absoluter Fehler:** $|x_{\text{korrekte Zahl}} - x_{\text{falsch gerundet}}|$
- **Relativer Fehler:** $\frac{|x_{\text{korrekte Zahl}} - x_{\text{falsch gerundet}}|}{|x_{\text{falsch gerundet}}|}$

Bei mehreren Additionen/Subtraktionen addieren sich die absoluten Fehler.

1.6.5. Fixkommazahlen

Umrechnung Dezimal zu Binär: Der Vorteil dieser Methode ist, dass man sofort feststellen kann, wenn sich die Binärstellen wie im folgenden Beispiel periodisch wiederholen:

$$0.55 = 0.10011_b$$

$$2 * 0.55 = 1.1 \text{ (die erste Binärstelle nach dem Komma ist 1)}$$

$$2 * 0.1 = 0.2 \text{ (für die zweite Binärstelle werden die Nachkommastellen von 1.1 dupliziert)}$$

$$2 * 0.2 = 0.4$$

$$2 * 0.4 = 0.8$$

$$2 * 0.8 = 1.6$$

$$2 * 0.6 = 1.2$$

$$2 * 0.2 = 0.4 \text{ (ab hier wiederholt sich } 0011_b \text{ bis die Anzahl Stellen erreicht ist)}$$

...

Fest-Komma:

$C_{[FK kn]}(X)$
Länge der binären Schreibweise
Anzahl der Nachkommastellen
n = Gesamtanzahl an Stellen
k = Anzahl der Nachkommastellen
$n-k$ = Menge der Vorkommastellen

-128	64	32	16	8	4	2	1	1/2	1/4	1/8	1/16
0	1	0	0	1	0	0	0	1	1	0	0

$$64 + 8 + 1/2 + 1/4 = 72 \frac{3}{4}$$

Exzess-q

Nullpunkt verschieben um z.B. negative Zahlen ohne Minus Zeichen darzustellen.

Zum Beispiel $n = 3$, Exzess = -4

Kleinste Zahl mit 3 Stellen: 000 -> Das wird die -4. Daraus folgt:

<i>Binär</i>	000	001	010	011	100	101	110	111
<i>Betrag</i>	0	1	2	3	4	5	6	7
<i>Exz-4</i>	-4	-3	-2	-1	0	1	2	3

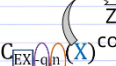
Binärzahl 1001 in Exzeß-q-Schreibweise

Bias: Formel: $2^{l-1} - 1$ $l = 4$

$$= 2^{4-1} - 1$$

$$= 2^3 - 1$$

$$= 8 - 1 = 7$$

Exzeß-q: 

Exzeß Bias Länge der binären Schreibweise

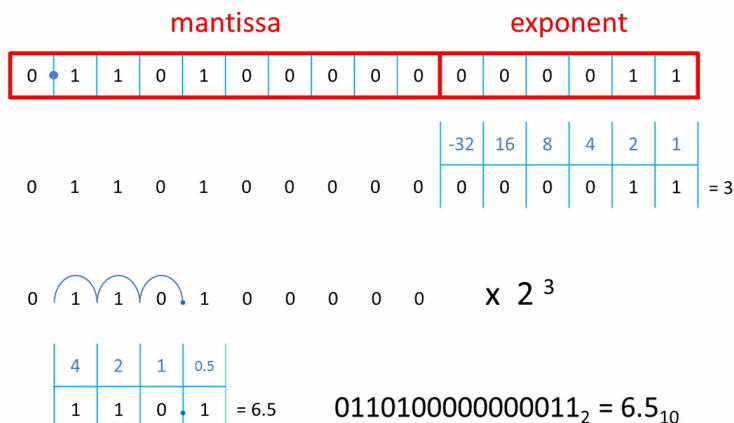
$$(1001)_2 = (9)_{10} \quad X = 9$$

$$C_{EX-7.4}(9)$$

1.6.6. Gleitkommazahlen (32bit)

Bei 32bit float sind 8bit für den Exponenten reserviert.

Wenn die erste Nummer des Exponenten 0 ist, dann ist der Exponent positiv. Das gleiche mit der Mantissa.



1.7. GRUPPENTHEORIE

Ein Ring ist ein Körper, wenn jedes Element des Rings ausser der Null ein multiplikatives Inverses hat, die Multiplikation kommutativ ist ($ab = ba$), das Distributivgesetz gilt ($a(1-b) = a - ab$) und wenn er eine 1 hat (multiplikatives neutrales Element)

1.8. INTERPRETATION EINES DATENWORTS

- Im **endlichen Ganzzahlkörper** gibt es immer eine **grösste** und eine **kleinste** Zahl
- **Begrenzt** wird die Darstellung dieser Zahl durch den zur **Verfügung stehenden Platz** des Speichers und der definierten Wortgrösse
- In der Welt der Zahlen im Rechner oder der Codierung existiert der Begriff **«unendlich» somit nicht** in unserer Vorstellung
- In der Informatik werden alle Informationen in sogenannten Codewörtern abgelegt
- Die Anzahl der darstellbaren Codewörter wird durch die Codewortlänge bestimmt. Ein Byte besteht aus 8 bit, ein Word besteht aus 16 oder 32 bit, ein TCP Paket besteht maximal aus 1024 bit.

1.8.1. Tupel, Zahl, Vektor, Polynom

Diese Darstellungsformen sind äquivalent und beschreiben alle das gleiche Codewort.

- **Tupel:** $(1,0,0,1)$
- **Zahl:** $1001_2 = 9_{10}$. Es gelten die üblichen Operationen der Ganzzahlrechnung.
- **Vektor:** $[1,0,0,1]^T$. Es gelten die üblichen Operationen der Vektorrechnung.
- **Polynom:** $g(u) = u^3 + u^0$ (entspricht $1u^3 \ 0u^2 \ 0u^1 \ 1u^0$). Es gelten die üblichen Operationen der Polynomrechnung.

Alle Berechnungen erfolgen in \mathbb{Z}_2

1.8.2. Multiplikation zweier Polynome mod 2

$$[100101] * [101] \Leftrightarrow$$

$$(1u^5 + 0u^4 + 0u^3 + 1u^2 + 0u^1 + 1u^0) * (1u^2 + 0u^1 + 1u^0) \Leftrightarrow$$

$$(u^5 + u^2 + u^0) * (u^2 + u^0) \bmod 2 = (u^{5+2} + u^{2+2} + u^{0+2} + u^{5+0} + u^{2+0} + u^{0+0}) \bmod 2 =$$

$$(u^7 + u^4 + u^2 + u^5 + u^2 + u^0) \bmod 2 = (u^7 + u^5 + u^4 + 2u^2 + u^0) \bmod 2 = u^7 + u^5 + u^4 + 1$$

ergibt das neue Codewort [1011001]

1.9. ZYKLISCHE GRUPPE

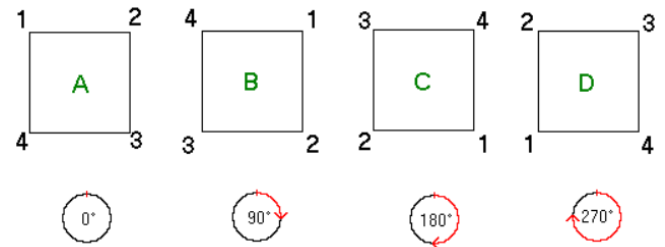
In der Gruppentheorie ist eine zyklische Gruppe eine Gruppe, die von einem einzelnen Element erzeugt wird. Sie besteht nur aus Potenzen des Erzeugers.

$$F(x) = x^3 + x + 1$$

Hat dieses Polynom in $\mathbb{Z}_2 = \{0,1\}$ eine Lösung?

Nach dem Hauptsatz der Algebra hat jedes Polynom so viele Nullstellen, wie durch die höchste Potenz angezeigt wird, hier 3 Nullstellen.

Eine zyklische Gruppe wird von einem einzelnen Element erzeugt und besteht nur aus Potenzen des Erzeugers. Definiert ist:



$$F(a) = a^3 + a + 1 = 0, \text{ das erzeugende Element } a$$

Dann können wir zunächst festhalten:

$$a = a$$

$$a^2 = a^2, \text{ aber:}$$

$$a^3 = a + 1$$

$$a^4 = a(a + 1) = a^2 + a$$

$$a^5 = a(a^2 + a) = a^3 + a^2 = a^2 + a + 1$$

$$a^6 = a(a^2 + a + 1) = a^3 + a^2 + a = a + 1 + a^2 + a = a^2 + 1$$

$$a^7 = a(a^2 + 1) = a^3 + a = a + 1 + a = 1$$

$$a^8 = a: \text{ der Zyklus beginnt von vorne}$$

Damit entsteht aus $\mathbb{Z}_2 = \{0,1\}$ der Erweiterungskörper

$$0, 1, a, a^2, a + 1, a^2 + a, a^2 + a + 1, a^2 + 1$$

Die Elemente können auch codiert/interpretiert werden durch:

$$\{000, 001, 010, 100, 011, 110, 111, 101\}$$

Derartige Zyklen entstehen z.B. beim zyklischen Code, oder der Erzeugung von Zufallszahlen.

1.10. WAHRSCHEINLICHKEIT

Definition: Zufallsvorgang, Zufallsexperiment

Unter einem Zufallsvorgang verstehen wir einen Vorgang, bei dem:

- im Voraus feststeht, welche mögliche Ausgänge dieser theoretisch haben kann (z.B. ein Bit wird gedreht, 0 oder 1, oder ein lesbares Zeichen wird in ein anderes lesbares Zeichen überführt).
- der sich einstellende, tatsächliche Ausgang im Voraus jedoch unbekannt ist (Tritt ein Bitfehler bei der Datenübertragung auf oder nicht? Unsicherheit bei einem Folgezeichen).

Zufallsexperimente, die geplant sind und kontrolliert ablaufen, heissen Zufallsexperimente.

Definition Ergebnismenge

Die Menge aller möglichen Ausgänge (Ergebnisse) eines Zufallsvorgangs heisst Ergebnismenge und wird mit Ω bezeichnet. Ein einzelnes Element $w \in \Omega$ heisst Ergebnis. Wir notieren die Anzahl aller Elemente von Ω , das heisst die Anzahl aller Ergebnisse mit $|\Omega|$.

1.10.1. Eigenschaften von Wahrscheinlichkeiten

- Wahrscheinlichkeit des sicheren Ereignisses: $P(1) = 1$
- Wahrscheinlichkeit des Komplementäreignisses: $P(\bar{A}) = 1 - P(A)$
- Wahrscheinlichkeit des unmöglichen Ereignisses: $P(0) = 0$
- Wertebereich der Wahrscheinlichkeit: $0 \leq P(A) \leq 1$
- Additionssatz für Wahrscheinlichkeiten $P(A \cup B) = P(A) + P(B) - P(A \cap B)$
- Additionssatz für 3 Ereignisse (Wahrscheinlichkeit, dass A, B oder C eintritt):
$$P(A \cup B \cup C) = P(A) + P(B) + P(C) - P(A \cap B) - P(A \cap C) - P(C \cap B) + P(A \cap B \cap C)$$

Wenn ein Experiment eine Anzahl **verschiedener** und **gleich möglicher** Ausgänge hervorbringen kann, dann ist die **Wahrscheinlichkeit** eines gewünschten Ausgangs gleich dem **Verhältnis der Anzahl der gewünschten zur Anzahl der möglichen Ausgänge**:

$$P(A) = \frac{\text{Anzahl der günstigen Ergebnisse } A}{\text{Anzahl aller Ergebnisse } \Omega} = \frac{|A|}{|\Omega|} = \frac{|A|}{n}$$

1.11. KOMBINATORIK

Wahrscheinlichkeit erfordert Berechnung von Anzahlen. Dafür wird die Kombinatorik verwendet. Einige grundsätzliche Fragen der Kombinatorik: Wie viele Möglichkeiten gibt es, bestimmte Objekte anzuordnen? Wie viele Möglichkeiten gibt es, bestimmte Objekte aus einer Menge auszuwählen? Hier betrachten wir nur soweit nötig die geordnete und die ungeordnete Probe.

1.11.1. Geordnete Proben

- Die Anzahl der **k-Tupel** aus einer **n-Menge mit Wiederholung** ist n^k

Beispiel: Bei einem Zifferschloss muss man eine **5-stellige Zahl** einstellen, die aus den Ziffern **0,1,...,9** gebildet wird. Es gilt: Menge (n) = 10, k = 5, ergibt $10^5 = 100'000$ Kombinationen.

- Die Anzahl der **k-Tupel** aus einer **n-Menge ohne Wiederholung** ist

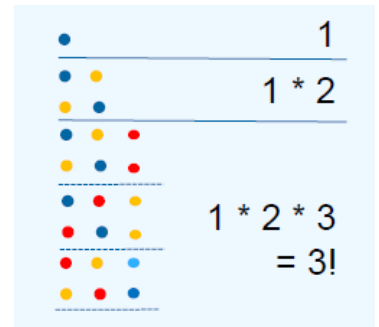
$$\frac{n!}{(n-k)!}$$

Beispiel: Beim Pferde-Toto «3 aus 18» muss man von 18 Pferden 3 gemäss der Reihenfolge ihres Zieleinlaufs tippen.

$$\text{Anzahl} = \frac{18!}{(18-3)!} = \frac{18 \cdot 17 \cdot 16 \cdot \cancel{15!}}{\cancel{15!}} = 18 \cdot 17 \cdot 16 = 4896$$

- Die Zahl der Permutationen einer **n-Menge ohne Wiederholung** für $n = k$ ist $n!$

Beispiel: Zur Festlegung einer Sitzordnung bei einer Feier mit 10 Personen gibt es $10! = 3'628'800$ Möglichkeiten.



1.11.2. Ungeordnete Proben

- Die Anzahl der **k-Tupel** aus einer **n-Menge** ist:

Beispiel: Eine Schulklasse mit 25 Schülern möchte ein Schachturnier austragen, bei dem jeder Schüler einmal gegen jeden anderen Schüler spielt. Wie viele Spiele werden ausgetragen?

$$\binom{25}{2} = \frac{25 \cdot 24 \cdot \cancel{23!}}{2! \cdot (25-2)!} = \frac{25 \cdot 24 \cdot \cancel{23!}}{2! \cdot \cancel{23!}} = \frac{25 \cdot 24}{2!} = 300$$

Die Anzahl aller möglichen Kombinationen k aus n unter Berücksichtigung der Reihenfolgen.

|

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

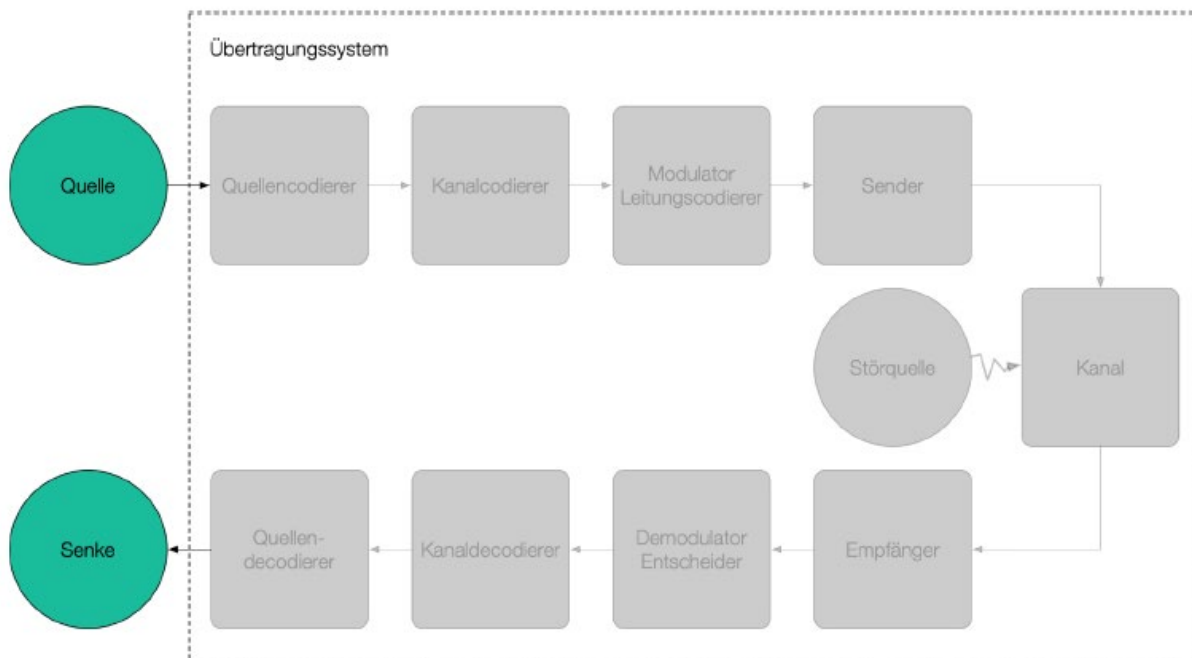
|

Da die Reihenfolge keine Rolle spielt, muss noch durch die Anzahl aller möglichen Kombinationen von k als $k!$ geteilt werden.

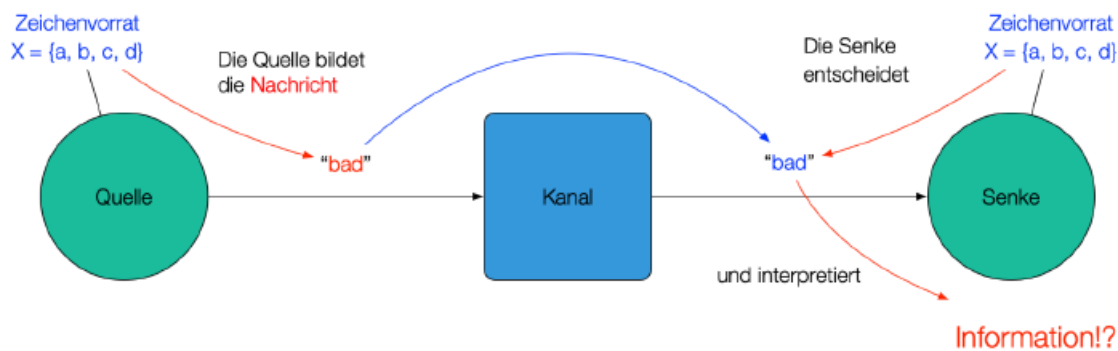
2. INFORMATIONSTHEORIE GRUNDLAGEN

Eine Nachricht wird als «Information» bezeichnet, wenn sie «relevant» und «nicht-redundant» ist.

- **Relevant:** Der Empfänger kann die Nachricht verstehen
- **Nicht-redundant:** Der Empfänger kann die Nachricht nicht voraussagen.



Modell der Informationsverarbeitung



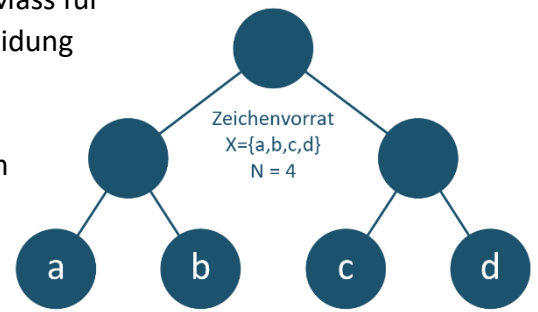
Nachricht	redundant	nicht-redundant
irrelevant	Zeichenvorrat bei Quelle und Senke verschieden	
relevant	Vorhersagbar	Information

2.1. DEFINITIONEN

Definition Entscheidungsgehalt: Der Entscheidungsgehalt ist das Mass für den Aufwand, der zur Bildung einer Nachricht bzw. für die Entscheidung einer Nachricht notwendig ist.

Anders ausgedrückt: «Anzahl Knoten», die auf dem Weg zu diesem Zeichen durchlaufen werden müssen, also die Anzahl Entscheidungen, die man zwischen «links» und «rechts» treffen muss, bis man beim Zeichen angekommen ist.

$$H_0 = \log_2(N) [bit]$$



Definition Entscheidungsfluss: Der Entscheidungsfluss ist definiert als

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

Wobei t die Zeit ist, die zur Übertragung eines Quellzeichens benötigt wird.

Definition Informationsgehalt: Der Informationsgehalt eines Zeichens sagt aus, wie viele Elementarentscheidungen zur Bestimmung dieses Zeichens zu treffen sind.

$$I(X_k) = \log_2 \left(\frac{1}{p(X_k)} \right) [bit]$$

Definition Entropie: Die Entropie bezeichnet den mittleren Informationsgehalt der Quelle. Sie zeigt also auf, wie viele Elementarentscheidungen die Quelle/Senke im Mittel pro Zeichen treffen muss.

$$H(X) = \sum_{k=1}^N p(X_k) * I(X_k) = \sum_{k=1}^N p(X_k) * \log_2 \left(\frac{1}{p(X_k)} \right) = [bit/Zeichen]$$

2.2. CODEWORTLÄNGE $L(X_k)$ VGL. INFORMATIONSGEHALT

Die tatsächliche Bitgrösse (als Integer) der obigen, oftmals gebrochenen Bits bezeichnet man als «Codewortlänge».

$$L(x_k) = \text{Aufgerundet}(I(x_k)) [Bit]$$

2.2.1. Mittlere Codewortlänge vgl. Entropie

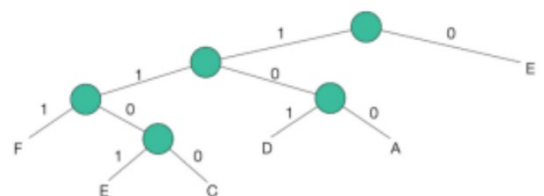
Wir können nun auch die mittlere Codewortlänge von allen Zeichen berechnen (kann eine reelle Zahl sein).

$$L = \sum_{k=1}^N p(x_k) * \left\lceil \log_2 \left(\frac{1}{p(x_k)} \right) \right\rceil \left[\frac{Bit}{Zeichen} \right]$$

Günstig ist, wenn der Wert L möglichst klein ist.

2.3. PRÄFIXEIGENSCHAFT

Eine Gruppe von Codeworten besitzt die Präfixeigenschaft, wenn alle Codes ohne Trennzeichen identifizierbar sind. Die Zeichen befinden sich in den «Blättern» des Baumes. ASCII hat die Präfixeigenschaft, der Morsecode nicht.



⇒ Beispiel eines Codewortbaums mit Präfixeigenschaft

2.4. SHANNON'SCHES CODIERUNGSTHEOREM

Das Codierungstheorem besagt, dass:

- Für jede beliebige Binärcodierung mit Präfixeigenschaft ist die mittlere Codewortlänge nicht kleiner als die Entropie: $H(X) \leq L$
- Für jede beliebige Quelle kann eine Binärcodierung gefunden werden, so dass gilt:
 $H(X) \leq L \leq H(X) + 1$

Der Begriff der Redundanz der Quelle wird erweitert um den Begriff der **Redundanz des Codes**:

Redundanz der Quelle: $R_Q = H_0 - H(X)$ [bit/Zeichen]

Redundanz des Codes: $R_C = L - H(X)$ [bit/Zeichen]

2.5. QUELLEN OHNE GEDÄCHTNIS

Die Auftretswahrscheinlichkeit eines Zeichens ist nicht abhängig von dem zuvor gesendeten Zeichen d.h. die Verbundwahrscheinlichkeit für die beiden Zeichen x und y ist $p(x_i, y_k) = p(x_i) * p(y_k)$

Zeichen	p	Entscheidungsgehalt: $h_z(1.2)$ im TR $H(X) = -\sum p(x_i) \log_2(p(x_i)) =$
a	0.3	$-(2 * 0.3 \log_2(0.3) + 2 * 0.1 \log_2(0.1) + 1 * 0.2 \log_2(0.2)) = 2.16$
b	0.1	$h_0(1.1)$ im TR $H_0 = \log_2(N) = \log_2(5) = 2.32 \text{ bit}$
c	0.1	
d	0.2	Informationsgehalt: $i_z(1.2)$ im TR
e	0.3	$I(x_k) = -\log_2(p(x_k)) \Rightarrow a = 1.74, b = 3.32, \dots, e = 1.74$

Entropie: $\sum^N p(x_k) * I(x_k) = 2.17 \text{ bit/Zeichen}$

Redundanz der Quelle: $R_Q = H_0 - H(X) = 2.32 - 2.16 = 0.16 \text{ bit}$

Zeichen	Codewort	Mittlere Codewortlänge: $L = \sum^N p(x_k) * L(x_k) = 2.4 \text{ bit}$
a	0	Redundanz des Codes: $R_C = L - H(X) = 2.4 - 2.16 = 0.24 \text{ bit}$
b	110	
c	1111	Bessere Codierung: Ziel ist es, für häufige Zeichen möglichst kurze Codeworte zu verwenden. Die könnte erreicht werden, indem folgende Änderung vorgenommen wird: $c = 111, d = 01$.
d	1110	
e	10	

Was sagt das Codierungstheorem von Shannon in diesem Fall?

$$H(x) < L < H(x) + 1$$

$$2.16 < 2.2 / 2.3 < 3.16$$

Wie gross ist die minimale Redundanz eines Codes für diese Quelle (Shannon):

$$H(X) \leq L = R_C + H(X) \rightarrow R_C$$

Code mit weniger Redundanz? Wie sieht dieser aus und wie gross ist die Redundanz?:

Huffman Code anwenden, L = mittlere Codewortlänge, R_C angeben

→ Für die Quelle sind wir mit dieser Codierung nahe an der minimalen mittleren Codewortlänge angelangt.

2.6. QUELLEN MIT GEDÄCHTNIS

Allgemein kann nicht von einer gedächtnislosen Quelle ausgegangen werden: $p(x_i, y_k) = p(x_i) * p(y_k | x_i)$

$$\text{Es galt: } H(X) = \sum_{i=1}^N p(x_i) * \log_2\left(\frac{1}{p(x_i)}\right)$$

Um $H(X, Y)$ zu bestimmen, setzten wir statt $p(x_i)$, $p(x_i, y_k) = p(x_i) * p(y_k | x_i)$ in die obige Gleichung ein, es folgt:

$$H(X, Y) = \sum_{i=1}^N \sum_{k=1}^N p(x_i) * p(y_k | x_i) * \log_2 \left(\frac{1}{p(x_i) * p(y_k | x_i)} \right)$$

$$H(X, Y) = \left[\sum_{i=1}^N \sum_{k=1}^N p(x_i) * p(y_k | x_i) * \log_2 \left(\frac{1}{p(x_i)} \right) \right] + \left[\sum_{i=1}^N \sum_{k=1}^N p(x_i) * p(y_k | x_i) * \log_2 \left(\frac{1}{p(y_k | x_i)} \right) \right]$$

$$H(X, Y) = H(X) + H(Y|X)$$

Es kann gezeigt werden, dass gilt: $H_0 \geq H(Y) \geq H(Y|X)$

2.6.1. Interpretation:

- Die mittlere Entropie einer Quelle ohne Gedächtnis ist stets grösser oder gleich der Entropie einer Quelle mit Gedächtnis. $R_Q = H_0 - H_{OG}(X) \leq H_0 - H_{MG}(X)$
- In der Quellencodierung sind daher nicht Einzelzeichen zu codieren, sondern stets Zeichenketten.
- Beispiel der Redundanz der deutschen Sprache:

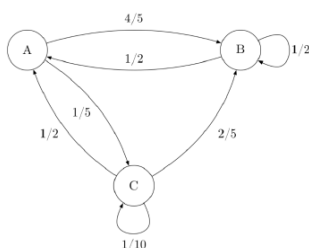
$H_0 = 4.7$ [bit/Zeichen]

Entropie der Einzelzeichen $H = 4.097$ [bit/Zeichen] $\rightarrow R = 0.6$ [bit/Zeichen]

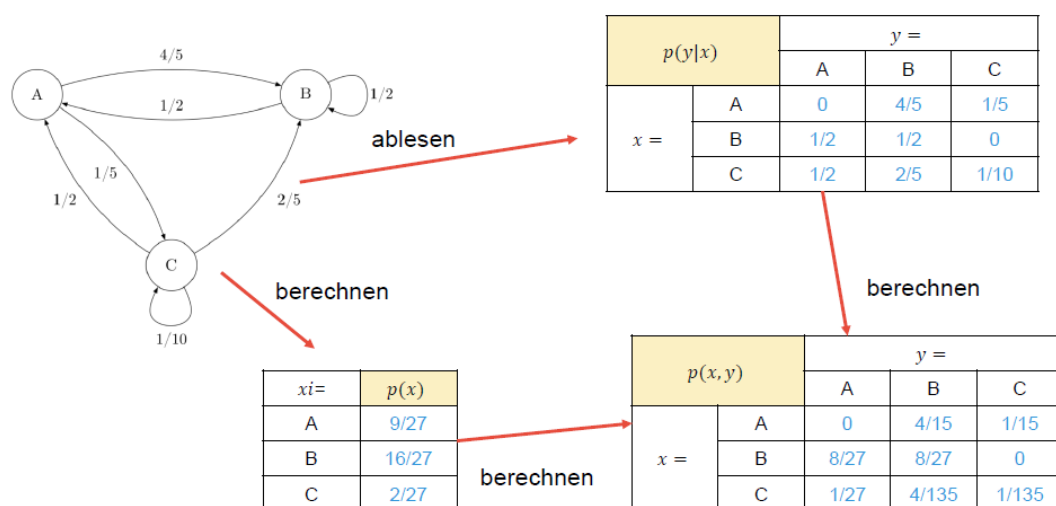
Entropie bei Ausnutzung aller Abhängigkeiten $H = 1.6$ [bit/Zeichen] $\rightarrow R = 3.1$ [bit/Zeichen]

Beispiel: Diskrete Quelle mit Gedächtnis

Gegeben sei eine Quelle mit dem Alphabet A, B, C . Die Abhängigkeiten werden durch ein Markov-Diagramm 1. Ordnung beschrieben.



Zur Berechnung der Entropien H mit $H(X, Y)$ werden die Wahrscheinlichkeiten $p(x)$, $p(y|x)$ und $p(x, y)$ benötigt.



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

3 Unbekannte, $p(A), p(B), p(C)$. Bei drei Unbekannten werden drei Gleichungen gebraucht:

$$p(A) = p(B) * 0.5 + p(C) * 0.5$$

$$p(B) = \dots$$

$$p(C) = \dots$$

Die Gesamtwahrscheinlichkeit muss 1 sein, also gibt es noch eine weitere Gleichung:

$$1 = p(A) + p(B) + p(C)$$

3. QUELLENCODIERUNG & KOMPRIMIERUNG

- **Datenkomprimierung:** verlustfrei oder verlustbehaftet
- **Verschlüsselung:** symmetrisch oder asymmetrisch

3.1. DATENKOMPRIMIERUNG

Komprimierung hat das Ziel, den Aufwand der Datenspeicherung und Datenübertragung zu reduzieren. Es können auch verschiedene Komprimierungsverfahren kombiniert werden.

Anforderungen:

- Hohe Komprimierungsrate für alle Typen von Daten (idealerweise ohne Kenntnis über die Eigenart der Daten)
- Hohe Encode- und Decode-Geschwindigkeit
- Geringe Ansprüche an die Hardware

Verfahren zur Datenkomprimierung

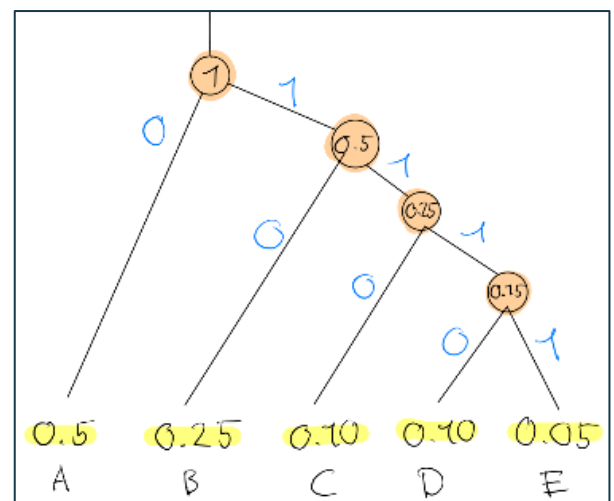
- **Statische Verfahren**, z.B. Huffman-Codierung für die deutsche Sprache
Eigenart der Daten werden berücksichtigt
- **Adaptive Verfahren**, z.B. Huffman-Codierung mit gemessener Häufigkeitsverteilung
Eigenart der Daten werden berücksichtigt
- **Dynamische Verfahren**, z.B. ITU Standard V42.bis basiert auf LZ77 (Lempel, ziv)
Eigenart der Daten werden nicht berücksichtigt

3.2. HUFFMAN-CODIERUNG

Huffman ist ein **rekursives** (Start bei Blättern) Verfahren für die **Bildung eines kommafreen** (Code hat Präfixeigenschaft - Kommafrei) Codes mit **minimaler mittleren Codewortlänge**.

Vorgehen

- Häufigkeit/Auftrittswahrscheinlichkeiten der Zeichen notieren (gelb)
- Häufigkeit in aufsteigender Reihenfolge anordnen
- Baum zeichnen: Knoten mit den geringsten Häufigkeiten verbinden - neu entstehender Knoten mit neuer Wahrscheinlichkeit beschriften (orange)
- Kanten beschriften (links 0, rechts 1)
- Binärcode von Wurzel zu Blättern ablesen (Resultate der Codierung)
A = 0, B = 10, C = 110, D = 1110, E = 1111



Minimale Redundanz: 0, erreichbar bei günstigen Auftretswahrscheinlichkeiten.

3.3. LAUFLÄNGENKOMPRIMIERUNG

Erkennung von Wiederholungen: Bei der Lauflängencodierung werden *Sequenzen von identischen Symbolen* durch deren *Anzahl* und (falls notwendig) das Symbol ersetzt.

RLE (Run Length Encoding) oder RLC (Run Length Coding) genannt.

Beispiel:

Quelltext w : *Agggbbbehffffgggg* $\Rightarrow |w| = 15$

Codiert w_c : *A3g2beh3f4g* $\Rightarrow |w_c| = 11$

Lauflängenkompromierung für Bit-Folgen

Bei der Kodierung von Bitfolgen existieren nur *zwei Möglichkeiten*: Eine Folge von Nullen oder eine Folge von Einsen. Auf *jede Sequenz von Nullen folgt garantiert mindestens eine Eins* – und umgekehrt.

Ausnahme ist, wenn das Ende der Nachricht erreicht ist.

- Der Kodierer einigt sich mit dem Dekodierer, mit *welchem Bit begonnen wird*, 0 oder 1
- Anschliessend werden abwechselnd die *Längen der 0 und 1 Folgen übertragen*
- Der Dekodierer muss zu *jedem empfangenen Wert entsprechend viele 0 oder 1 Bits ausgeben*

Beispiel:

Originalnachricht $w = 1111\ 1110\ 0000\ 1000\ 0001\ 1111 \Rightarrow |w| = 24\ bit$

Start mit einer 1

Übertragener Code: *7 5 1 6 5*

Um jede Stelle von 0 .. 7 zu codieren reichen 3 Bit je Stelle aus, das Codewort wird zu

$w_c = 111\ 101\ 001\ 110\ 101 \Rightarrow |w_c| = 15$

3.4. DATENKOMPRIMIERUNG NACH LEMPEL-ZIV (LZ)

Erkennen von *wiederkehrenden Mustern*. Komprimiert dadurch, dass *zuvor eingelesener Text als Tabelle* genutzt wird. Phrasen aus dem Eingabetext werden durch Zeiger in der Tabelle ersetzt. Dadurch wird die Komprimierung erreicht. Der Grad der Komprimierung hängt von der Länge der Phrasen, Fenstergrösse und Entropie des Ursprungstextes ab. Bei gleichen nah beieinander liegenden Textsequenzen kommt es sehr schnell zur Kompression.

Grundüberlegungen

- Der zu komprimierende Code hat *wiederkehrende Muster oder Phrasen*
- Anstatt den Code vollständig zu übertragen, werden «nur» die codierten Phrasen übertragen
- Dazu müssen die Phrasen zur Laufzeit erfasst und in einem Phrasenspeicher gespeichert und codiert werden. Die Grösse des Wörterbuchs und des *«look ahead buffers»* muss bestimmt werden.

Umsetzung

- Während des Durchlaufens der Daten wird ein *ständig wachsender Baum erzeugt*.
- Der Baum *dient als Wörterbuch* und zeigt Regularitäten auf.
- Die Knoten *dienen als Referenzen*.
- Werden gleiche Subdaten *wiederholt* geparkt, so kann auf den entsprechenden Knoten des Wörterbuches *referenziert* werden.

Die Datenstruktur besteht aus

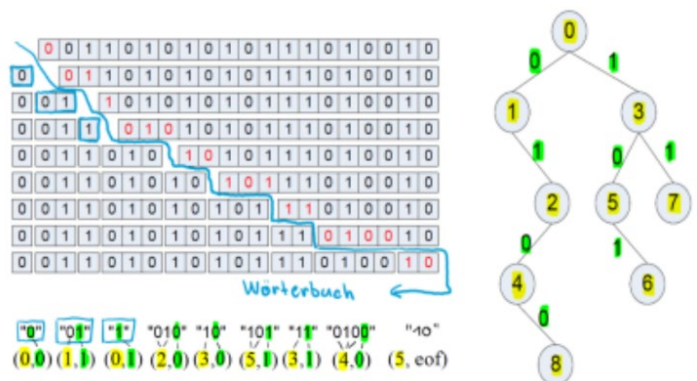
- Einem Textfenster, dem *search buffer*: hier stehen die schon codierten Symbole, wird dynamisch aufgebaut
- Einem nach vorn gerichteten Puffer *look-ahead buffer*, dieser zeigt auf die als nächstes zu codierenden Symbole

- Sollte eine Sequenz von Symbolen in dem *look-ahead buffer* und dem *search buffer* übereinstimmen, so wird ein Code bestehend aus *Position* und *Länge* im search buffer gebildet und abgespeichert.
- Ansonsten wird der Code so gespeichert, wie er vorlag
- Anschliessend schieben sich beide Puffer eine Position nach vorn, deshalb wird diese Methode auch die Methode der gleitenden Fenster genannt.

Beispiel

search buffer/Textfenster	Look-ahead buffer/Puffer	Coding <i>Position</i> und <i>Länge</i>
	sir_sid_eastman	(0,0,"s")
s	ir_sid_eastman	(0,0,"i")
si	r_sid_eastman	(0,0,"r")
sir	_sid_eastman	(0,0,"_")
sir_	sid_eastman	(4,2,"d")
sir_sid	_eastman	

- Suche in der Tabelle eine möglichst lange Zeichenfolge, die mit den nächsten n zu codierenden Zeichen übereinstimmt
- Bilde ein Token und speichere es
- Verschiebe das Fenster um (n+1) Zeichen
- Wiederhole, bis alle Zeichen codiert sind



Lempel-Ziv für Bitfolgen

- Beginne mit einem Binärbaum, der einen Root-Knoten mit dem Index 0 besitzt
- Suche im Binärbaum die längste Zeichenfolge (bzw. «Knotenfolge»), die mit den nächsten n Zeichen übereinstimmt.
- Codiere den Eintrag in der Form (I,N): I = Index vom aktuellen Knoten, N = nächstes Zeichen
- Erstelle beim aktuellen Knoten einen neuen Kindknoten mit dem Index $I_{\max}+1$ und dem Zeichen an der Position n+1 als Kantenbeschriftung.
- Verschiebe das «Fenster» hinter die n+1 Zeichen auf das nächste Zeichen.
- Wiederhole, bis alle Zeichen codiert sind.

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

Buffer	Erkannte Zeichenfolge (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)	

- Quelltext: $w = 123123123123$
- Codiert: $w_c = 1\ 2\ 3\ 10\ 12\ 11\ 13$
- Codegrösse: $\max(w_c) = 13 = 1101_b \Rightarrow 4\text{ Bit}$
- Zeichenanzahl: $|w| = 12, |w_c| = 7$

Kompression: $\frac{4 \cdot |w_c|}{4 \cdot |w|} = \frac{28}{48} = 0.58 = 58\%$

⇒ Hinweis: $4 \cdot |w|$ da immer gilt: $\max(w) = 9 = 1001_b \Rightarrow 4\text{ Bits.}$

4. QUELLENCODIERUNG & VERSCHLÜSSELUNGSVERFAHREN

4.1. SYMMETRISCHE VERFAHREN

Beide Teilnehmer verwenden **denselben Schlüssel**

Anzahl der erforderlichen Schlüssel **wächst stark**, da für jedes Paar, welches Daten austauscht, ein eigener Schlüssel erstellt werden muss.

Anzahl Schlüssel bei 100 Mitgliedern: $N = \binom{100}{2} = \frac{100 \cdot 99}{2} = 4950$

- Caesar Chiffre / Substitutionsverfahren
- Transpositionsverfahren
- Vigenère-Chiffre *Quadratische Anordnung von untereinander stehenden verschobenen Alphabeten*
- DES *Data Encryption Standard*

DES / Data Encryption Standard

Wurde 1977 als offizieller Standard für die US-Regierung bestätigt und wird seither international vielfach eingesetzt. Heute wird er aufgrund der verwendeten Schlüssellänge von nur 56 Bits für viele Anwendungen als nicht ausreichend sicher erachtet.

4.2. ASYMMETRISCHE VERFAHREN

Private und öffentliche Schlüssel

Ein Schlüssel zur Codierung und einen zweiten zur De-Codierung.

4.2.1. RSA-Verfahren

Die RSA-Verschlüsselung ist ein Public-Key-Verfahren. Funktionsprinzip: Sie vergeben einen öffentlichen Schlüssel, mit dem jeder Botschaften an Sie so verschlüsseln kann, dass nur Sie sie entschlüsseln können.

Public & Private Key erhalten

1. **Zwei Primzahlen (p,q) multiplizieren** zum Produkt **n**: $n = p * q = 3 * 11 = 33$

2. Die Eulersche **φ -Funktion von n berechnen**. $\varphi(p * q)$ ergibt dasselbe Resultat und da beide Zahlen Primzahlen sind, kann einfach $(p - 1) * (q - 1)$ gerechnet werden.

$$\varphi(n) = \varphi(p * q) = \varphi(3) * \varphi(11) = (3 - 1) * (11 - 1) = 2 * 10 = 20$$

3. Eine **beliebige Zahl a zwischen 1 und $\varphi(n)$ auswählen**, die mit $\varphi(n)$ teilerfremd ist:

z.B. **a = 3**, Test: $\text{ggT}(20,3)=1$

4. Das **multiplikative Inverse b** von **a** in $\mathbb{Z}_{\varphi(n)}$ berechnen $a * b \equiv 1 \mod \varphi(n)$

$$3 * b \equiv 1 \mod 20$$

Kleinstmögliche Zahl $3 * 7 = 21, \frac{21-1}{20} = 1 \Rightarrow$ durch **20** teilbar, also **b = 7**

5. Nun haben wir den **Public Key des Empfängers (b & n)** und den **Private Key(a)** zur Verschlüsselung:
b = 7, n = 33 und a = 3

Ablauf für Prüfung:

- Beide Primzahlen **p** & **q** und **n** angeben
- **$\varphi(n)$** angeben mit ausrechnen
- Es gilt: **e bzw b * d bzw a mod $\varphi(n) = 1$** – ausrechnen
- Privaten Schlüssel **d bzw a** angeben
- Nachricht entschlüsseln

Text verschlüsseln

1. **Buchstabentabelle** generieren, zum Beispiel

A	B	C	D	E	F	G	H	I	J	K	L	M
1	2	3	4	5	6	7	8	9	10	11	12	13
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
14	15	16	17	18	19	20	21	22	23	24	25	26

2. Text in **Zahlen aus Tabelle umwandeln**: T = 20, E = 5, S = 19, T = 20

3. Zahlen mit **b potenzieren** und **Modulo n** rechnen:

$$20^7 \bmod 33 \equiv 26, 5^7 \bmod 33 \equiv 14, 19^7 \bmod 33 \equiv 13$$

4. Nun kann die **Nachricht** «26, 14, 13, 26» **gesendet** werden.

Text entschlüsseln

1. **Dieselbe Buchstabentabelle** wie beim Verschlüsseln verwenden.

2. Die empfangene Nachricht mit **a potenzieren** und **Modulo n** rechnen:

$$26^3 \bmod 33 \equiv 20, 14^3 \bmod 33 \equiv 5, 13^3 \bmod 33 \equiv 19$$

3. Zahlen in Buchstaben umwandeln: 20 = **T**, 5 = **E**, 19 = **S**, 20 = **T**

4.3. EUKLIDISCHER ALGORITHMUS (MODULO)

Gibt ggT (grösster gemeinsamer Teiler) & kgV (kleinstes gemeinsames Vielfaches) aus.

Wenn der ggT von zwei Zahlen 1 ist, sind die beiden Zahlen **teilerfremd** (sie haben keinen gemeinsamen Teiler). Bei geraden Zahlen sind alle kleineren geraden Zahlen **nicht** teilerfremd. Bei Primzahlen sind alle Zahlen kleiner als die Primzahl teilerfremd. Die Zahl 1 ist bei jeder Zahl teilerfremd.

Teilerfremde Zahlen der Zahlen 1-10

1	2	3	4	5	6	7	8	9	10
1	1	1,2	1,3	1,2,3,4	1,5	1,2,3,4,5,6	1,3,5,7	1,2,4,5,7,8	1,3,7,9

Primfaktorzerlegung

$$12 = 2 * 2 * 3, 18 = 2 * 3 * 3 \Rightarrow 2 * 3 = 6 \Rightarrow \text{ggT}. (12 * 18) / 6 = 36 \Rightarrow \text{kgV}$$

Bei grossen Zahlen sehr rechenaufwendig. Deshalb bestimmen wir den ggT mit dem Euklidischen Algorithmus, und das kgV aus $(a*b) / \text{ggT}(a,b)$

Euklidischer Algorithmus (ggT finden)

Seien $a, b \in \mathbb{N}, a \neq b, a \neq 0, b \neq 0$

Initialisierung: Setze $x := a, y := b$ und $q := x \text{ div } y$ (wie oft passt y in x) und $r := x - q * y$ (d.h. bestimme q und r so, dass $x = q * y + r$ ist).

Wiederhole, bis $r = 0$ ist.

Ablauf	x	y	$q := x \text{ div } y$	$r := x - q * y$
Initialisierung	122	72	1	$122 - 72 = 50$
1. Wiederholung	72	50	1	$72 - 50 = 22$
2. Wiederholung	50	22	2	$50 - 44 = 6$
3. Wiederholung	22	6	3	$22 - 18 = 4$
4. Wiederholung	6	4	1	$6 - 4 = 2$
5. Wiederholung	4	2 = ggT	2	0

Erweiterter Euklidischer Algorithmus (ggT finden und als Linearkombination darstellen)

Seien $a, b \in \mathbb{N}, a \neq b, a \neq 0, b \neq 0$

Initialisierung: Setze $x := a, y := b, q := x \text{ div } y$ (wie oft passt y in x), $r := x - q * y$ (d.h. bestimme q und r so, dass $x = q * y + r$ ist), und $(u, s, v, t) = (1, 0, 0, 1)$

Wiederhole, bis $r = 0$ ist.

$x = y$ aus der vorangegangenen Zeile

$y = r$ aus der vorangegangenen Zeile

$q = x \text{ div } y, r = x \text{ mod } y = x - y * q$

$u = s$ aus der vorangegangenen Zeile

$s = u - q * s$ mit u, q & s aus der vorangegangenen Zeile

$v = t$ aus der vorangegangenen Zeile

$t = v - q * t$ mit v, q & t aus der vorangegangenen Zeile

Ergebnis: In der letzten Zeile gilt $y = \text{ggT}$, $s * a + t * b$. Wenn $\text{ggT} = 1$ ist, dann folgt: $t * b = 1 \text{ mod } a$

Ablauf	x	y	$q := x \text{ div } y$	$r := x - q * y$	u	s	v	t
Initialisierung	99	79	1	$99 - 79 = 20$	1	0	0	1
1. Wiederholung	79	20	3	$79 - 60 = 19$	0	$1 - 1 * 0 = 1$	1	$0 - 1 * 1 = -1$
2. Wiederholung	20	19	1	$20 - 19 = 1$	1	$0 - 1 * 1 = -1$	-1	$1 - 1 * -1 = 2$
3. Wiederholung	19	1	19	$19 - 19 = 0$	-3	$1 - 1 * -3 = 4$	4	$-1 - 1 * 4 = -5$

Bedeutet: $4 * 99 + -5 * 79 = 1$ ($s * x + t * y = \text{kgV}$)

$$\text{kgV} = \frac{x * y}{\text{ggT}(x, y)} = \frac{99 * 79}{1} = 7821$$

Satz von Euler

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

Eulersche φ -Funktion

Sei $n \in \mathbb{N}$ und $\mathbb{Z}_n^* = \{x \in \mathbb{Z}_n \mid x \text{ hat ein multiplikatives Inverses in } \mathbb{Z}_n\}$. Dann berechnet sich die Eulersche φ -Funktion als

$\varphi(n)$ = Anzahl Zahlen $m \in \mathbb{N}$ mit $1 \leq m \leq n$ und $\text{ggT}(n, m) = 1 \Rightarrow$ Anzahl der teilerfremden Zahlen zwischen 0 und n

$\varphi(n)$ = Anzahl Elemente $x \in \mathbb{Z}_n$ mit multiplikativem Inversen

$\varphi(n) = |\mathbb{Z}_n^*|$

Formen zur Berechnung:

1. Sei $n \in \mathbb{N}$ eine Primzahl: Dann ist $\varphi(n) = n - 1$

2. Sei $n \in \mathbb{N}$ eine Primzahl und $p \in \mathbb{N} \setminus \{0\}$: Dann ist $\varphi(n^p) = n^{p-1} * (n - 1)$

3. Seien $m, n \in \mathbb{N} \setminus \{0\}$ und $\text{ggT}(m, n) = 1$: Dann ist $\varphi(n * m) = \varphi(n) * \varphi(m)$

Beispiel 1: $\varphi(11) = 10$

Beispiel 2: $\varphi(5^3) = 5^2 * 4 = 25 * 4 = 100$

Beispiel 3: $\varphi(2 * 47) = \varphi(2) * \varphi(47) = 1 * (47 - 1) = 46$

Multiplikatives Inverses

Seien $q \in \mathbb{N}$, $q \neq 0$ und $m \in \mathbb{N}$, $m \neq 0$. Dann gilt:

Es gibt eine Zahl $n \in \mathbb{N}$ mit $m * n \equiv 1 \pmod{q} \Leftrightarrow \text{ggT}(q, m) = 1$

Beispiel: Multiplikatives Inverse von 7 in \mathbb{Z}_{11} : $7 * x \equiv 1 \pmod{11}$

x kann auf 2 Arten berechnet werden. Entweder durch Durchprobieren oder durch den erweiterten euklidischen Algorithmus.

Durchprobieren:

$7 * 1 - 1 = 6 \Rightarrow$ nicht durch 11 teilbar

$7 * 2 - 1 = 13 \Rightarrow$ nicht durch 11 teilbar

...

$7 * 8 - 1 = 55 \Rightarrow$ durch 11 teilbar, $x = 8$

Erweiterter euklidischer Algorithmus: Siehe unten, danach das letzte Resultat von t (in diesem Fall -3) durch Modulo der Zahl in der Originalrechnung nehmen.

$-3 \pmod{11} = \frac{|-3|}{11} = \text{Rest } 8 =$ multiplikatives Inverses von 7 in \mathbb{Z}_{11}

Liste von Primzahlen

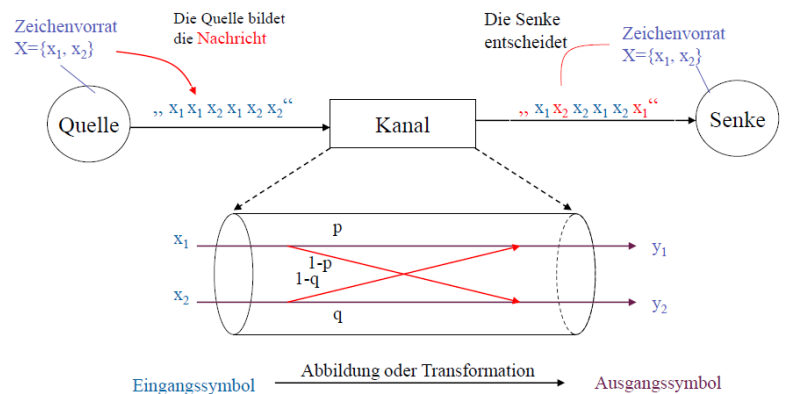
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997, 1009, 1013, 1019, 1021, 1031, 1033, 1039, 1049, 1051, 1061, 1063, 1069, 1087, 1091, 1093, 1097, 1103, 1109, 1117, 1123, 1129, 1151, 1153, 1163, 1171, 1181, 1187, 1193, 1201, 1213, 1217, 1223, 1229, 1231, 1237, 1249, 1259, 1277, 1279, 1283, 1289, 1291, 1297, 1301, 1303, 1307, 1319, 1321, 1327, 1361, 1367, 1373, 1381, 1399, 1409, 1423, 1427, 1429, 1433, 1439, 1447, 1451, 1453, 1459, 1471, 1481, 1483, 1487, 1489, 1493, 1499, 1511, 1523, 1531, 1543, 1549, 1553, 1559, 1567, 1571, 1579, 1583, 1597, 1601, 1607, 1609, 1613, 1619, 1621, 1627, 1637, 1657, 1663, 1667, 1669, 1693, 1697, 1699, 1709, 1721, 1723, 1733, 1741, 1747, 1753, 1759, 1777, 1783, 1787, 1789, 1801, 1811, 1823, 1831, 1847, 1861, 1867, 1871, 1873, 1877, 1879, 1889, 1901, 1907, 1913, 1931, 1933, 1949, 1951, 1973, 1979, 1987, 1993, 1997, 1999

5. KANALMODELL

Abstrakte Abbildung eines Kanals.

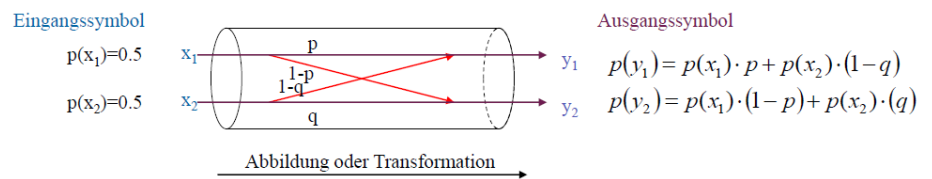
Beschreibt u.a. die **Schwierigkeiten bei der Datenübertragung** im Bezug auf den Kanal, z.B. die **Fehlerwahrscheinlichkeit**.

Aufgrund von «Rauschen» können bei der Übertragung Fehler auftreten. «Rauschen» kann z.B. eine schlechte Verbindung sein. Dieses Phänomen kann in einer Kanalmatrix abgebildet werden.



5.1. KANALMATRIX

Die Kanalmatrix beschreibt die Wahrscheinlichkeit, dass ein Zeichen x_i auf ein korrektes oder inkorrektes Zeichen y_i abgebildet wird.

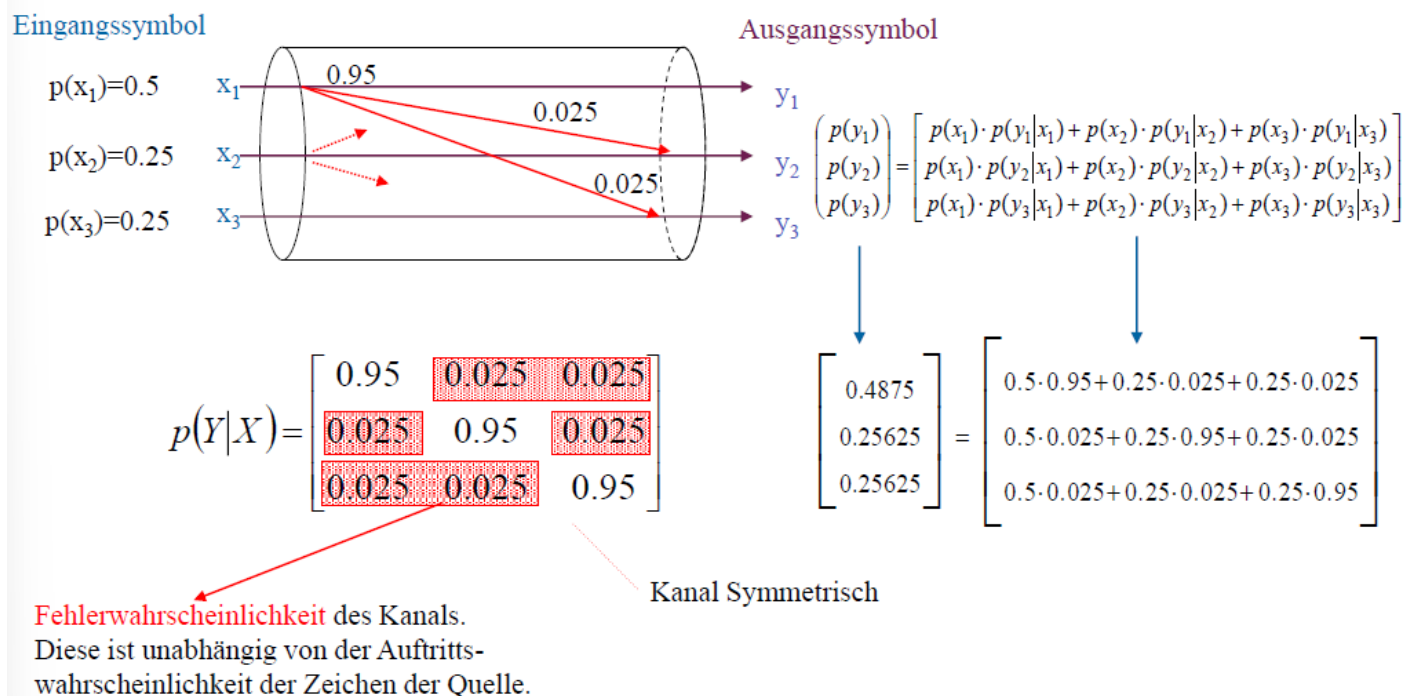


$$P(Y|X) = \begin{bmatrix} p(y_1|x_1) & p(y_2|x_1) \\ p(y_1|x_2) & p(y_2|x_2) \end{bmatrix}$$

$$p(Y|X) = \begin{bmatrix} p & 1-p \\ 1-q & q \end{bmatrix} \mapsto \begin{bmatrix} \sum = 1 \\ \sum = 1 \end{bmatrix}$$

$p(y_1|x_2)$ bedeutet: Die Wahrscheinlichkeit, dass ein y_1 ankommt, wenn ein x_2 gesendet wurde.

Rechenbeispiel



Eigenschaften

- Ist die Wahrscheinlichkeit für eine inkorrekte Zuweisung 0, so ist der Kanal nicht gestört.

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

- Sind alle Zuweisungen gleich wahrscheinlich, so ist der Kanal «vollständig» gestört.

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

- Ist $n = m$, so ist der Kanal symmetrisch

$$P(X|Y) = \begin{bmatrix} 0.95 & 0.025 & 0.025 \\ 0.025 & 0.95 & 0.025 \\ 0.025 & 0.025 & 0.95 \end{bmatrix}$$

Ausgangswahrscheinlichkeit

Wir können nun die Wahrscheinlichkeit für das Auftreten eines Zeichens y_i anhand der Kanalmatrix berechnen. (Die Summe aus den inkorrekten und korrekten Zuweisungen.)

$$p(y_i) = \sum_{k=1}^m p(x_k) * p(y_i|x_k)$$

5.2. MAXIMUM LIKELIHOOD (RESTWAHRSCHEINLICHKEIT)

Ist ein Kanal gestört, so müssen wir anhand des erhaltenen Zeichens y_i entscheiden, welches Zeichen x_i tatsächlich gesendet wurde. Man nehme die höchste Wahrscheinlichkeit in jeder Spalte einer Kanalmatrix und bilde diese auf x ab.

$$P(Y|X) = \begin{bmatrix} 0.6 = y_1 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.6 = y_3 \\ 0.3 & 0.2 = y_2 & 0.5 \end{bmatrix}$$

Aus jeder Spalte (↓) die höchste Wahrscheinlichkeit verwenden -> y-Wert

Jede Zeile (→) ist ein x-Wert, also Zeile 1 bezieht sich auf das x_1 , Zeile zwei auf das x_2 usw.

$$P(Y|X) = \begin{matrix} & y_1 & \dots & y_n \\ \begin{matrix} x_1 \\ \vdots \\ x_m \end{matrix} & \begin{bmatrix} p(y_1|x_1) & \dots & p(y_n|x_1) \\ \vdots & \ddots & \vdots \\ p(y_1|x_m) & \dots & p(y_n|x_m) \end{bmatrix} \end{matrix}$$

$$P = p(x_1) * p(y_1|x_1) + p(x_2) * p(y_3|x_2) + p(x_3) * p(y_2|x_3)$$

5.3. TRANSFORMATION

Wir können feststellen, dass bei der Datenübertragung über einen gestörten Kanal «Informationen» verloren gehen. Das bedeutet, der mittlere Informationsgehalt (die Entropie) nimmt ab.

$H(X)$: Eingangsentropie, $H(Y)$: Ausgangsentropie

- Die Transformation beschreibt den **maximalen, fehlerfreien Informationsfluss** über einen Kanal.
 - Verändert sich die Entropie der Quelle, verändert sich auch die Transformation.
 - Nimmt die **Fehlerwahrscheinlichkeit zu**, so **verringert** sich die **Transformation**.
 - Transformation wird durch die Quelle bestimmt.
 - Sind alle Positionen der Kanalmatrix **gleich besetzt**, so wird die Transformation $T = 0$, d.h. $H(Y) = H(Y|X) = 1$, unabhängig von der Entropie am Kanaleingang. (-> Transformation maximal)
- $$P(Y|X) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, p(x) = [0.5 \ 0.5]$$
- $T = 1$: ungestörter Kanal, $T = 0$: vollständig gestörter Kanal.

Wie gross ist die Transinformation?

$$p(x) = [\text{Auftrittswahrscheinlichkeit}_1 \text{ Auftrittswahrscheinlichkeit}_2]$$

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

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

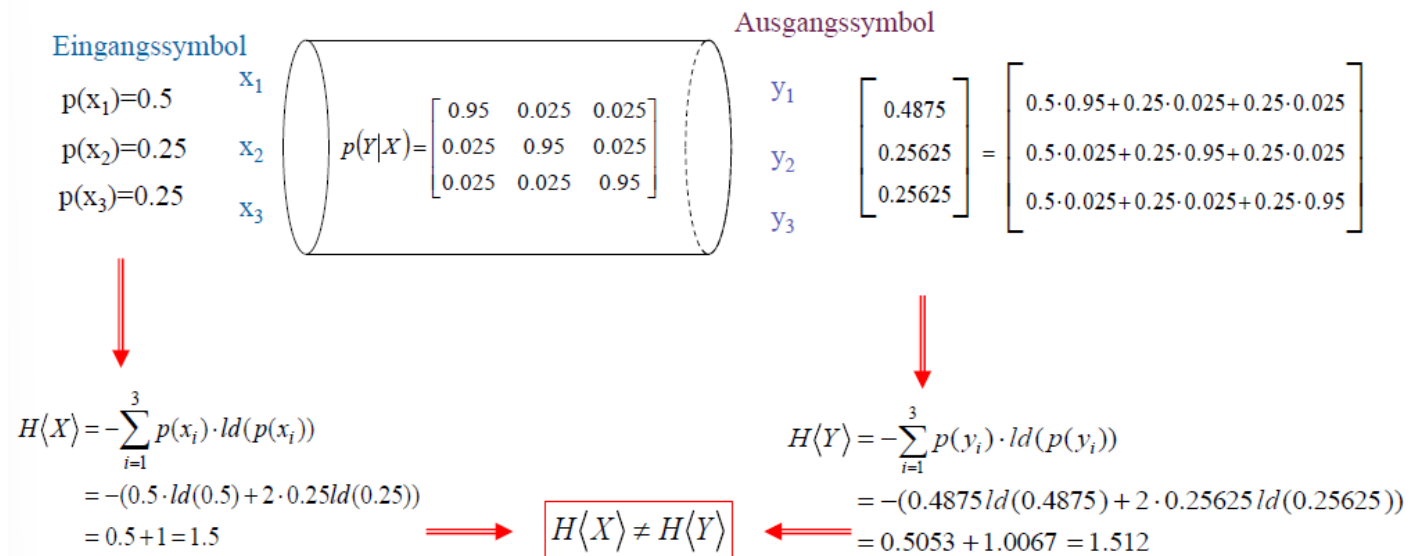
$$py = px * pm = [...] \text{ im TR}$$

Entropie am Kanalausgang: $H(Y) = -\sum_k p(y_k) \log_2 p(y_k)$

Irrelevanz: $H(Y|X) = -\sum_i \sum_k p(x_i, y_k) \log_2 p(y_k|x_i)$

Transinformation $T = H(Y) - H(Y|X)$

Maximale Transinformation: $P(Y|X) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, p(x) = [0.5 \quad 0.5]$



Verbundentropie $H(X, Y)$

Das paarweise Auftreten aller möglichen Kombinationen am Kanalein- und ausgang.

$$H(X, Y) = -\sum_i^n \sum_j^n p(x_i, y_j) * \log_2(p(x_i, y_j))$$

5.4. ÄQUIVOKATION (VERLUST $H(Y|X)$)

Beschreibt die **Ungewissheit über ein gesendetes Zeichen** bei bekannten Empfangszeichen. Ist der Kanal **fehlerfrei**, ist die Äquivokation gleich **0**. Wird auch **Rückschlussentropie** genannt.

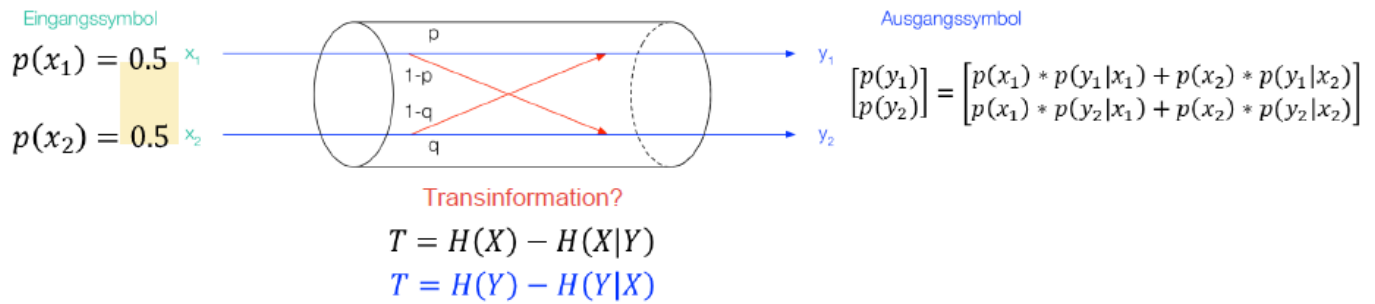
$$H(X|Y) = -\sum_i^n \sum_j^n p(y_i) * p(x_j|y_i) * \log_2(p(x_j|y_i))$$

5.5. IRRELEVANZ (RAUSCHEN)

Beschreibt die Ungewissheit der empfangenen Zeichen bei vorgegebenen Sendezeichen.

$$H(Y|X) = -\sum_i^n \sum_j^n p(x_i) * p(y_j|x_i) * \log_2(p(y_j|x_i))$$

5.6. BEISPIEL



Sei:

$p = q = 1$

$\Rightarrow p(y_1) = p(x_1) = 0.5$

$p(y_2) = p(x_2) = 0.5$

$\Rightarrow H(Y) = H(X) = 1$

$$H(Y|X) = - \sum_i^n \sum_j^n p(x_i, y_j) * \log_2(p(y_j|x_i))$$

$$= -(p(x_1) * p(y_1|x_1) * \log_2(1) + p(x_2) * p(y_2|x_2) * \log_2(1))$$

$$= 0$$

$T = H(Y) - H(Y|X)$

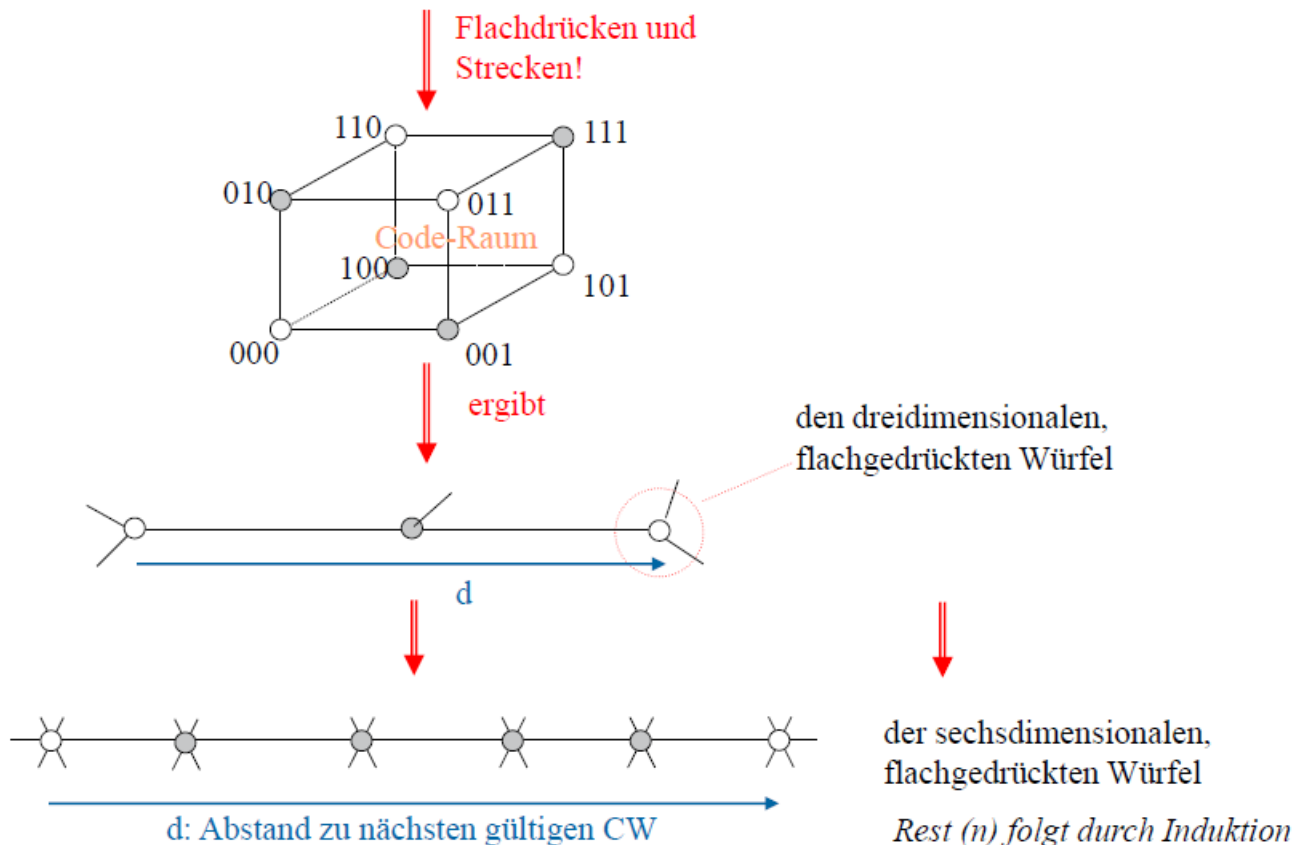
$T = H(Y) = 1$

6. KANALCODIERUNG & BLOCKCODES

Kanalcodierung beinhaltet Blockcodes und Faltungscodes. Die Kanalcodierung hat zum Ziel, bewusst Redundanz in eine Nachricht zu bringen, um Fehlern bei der Übertragung entgegenzuwirken. Der Coderaum wird dafür in gültige und ungültige Codeworte aufgeteilt.

6.1. DER N-DIMENSIONALE CODERAUM

Weisse Punkte: Gültig, graue Punkte: ungültig

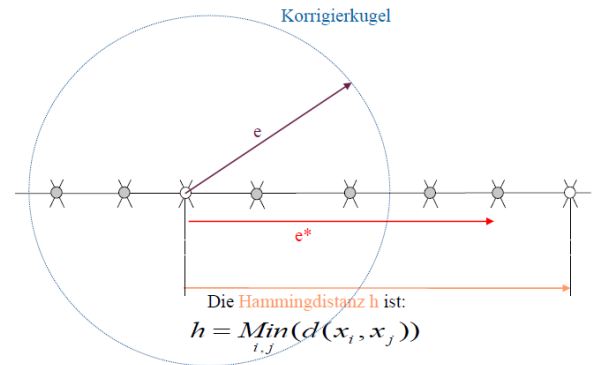


Definition

- Anzahl der sicher **erkennbaren** Fehler: $e^* = h - 1$
- Anzahl der sicher **korrigierbaren** Fehler, wenn h **gerade**: $h = 2e + 2 \Rightarrow e = \frac{h-2}{2}$
- Anzahl der sicher **korrigierbaren** Fehler, wenn h **ungerade**: $h = 2e + 1 \Rightarrow e = \frac{h-1}{2}$

Blockcode «Voci»:

- Anzahl Worte = 2 bei Binärcode
- $n = m + k$ = Anzahl Codestellen ($2^k - 1 = n$, falls $n = m + k$)
- m = Anzahl Nachrichtenstellen
- k = Anzahl Kontrollstellen
- Gültige Codewörter: (Anzahl Worte) m / Binär $2m$
- Mögliche Codewörter: (Anzahl Worte) $m + k$ / Binär $2m + k$
- **Hammingdistanz** h : beschreibt den minimalen Abstand zwischen zwei gültigen Codewörtern im gesamten Coderaum.



$$h = \min_{i,j} (d(x_i, x_j))$$

Treten mehr Fehler auf als korrigierbar sind, so wird entweder falsch korrigiert oder der Fehler wird nicht erkannt.

Der Coderaum ist **Dichtgepackt**, wenn sich alle Codewörter (gültige und ungültige) in einer Korrigierkugel befinden.

Sei:

- n die Dimension des Code (Anzahl aller Codewörter = 2^n)
- m die Dimension der Nachrichten (Anzahl aller gültigen Codewörter = 2^m)
- k die Dimension der Kontrollstellen mit $n = m + k$

So folgt die Codeabschätzung:

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

Anzahl der CW bzw. Korrigierkugeln Anzahl der CW pro Korrigierkugel Anzahl aller CW

Wenn die linke Seite der Gleichung = die rechte Seite, ist der Coderaum **dichtgepackt**.

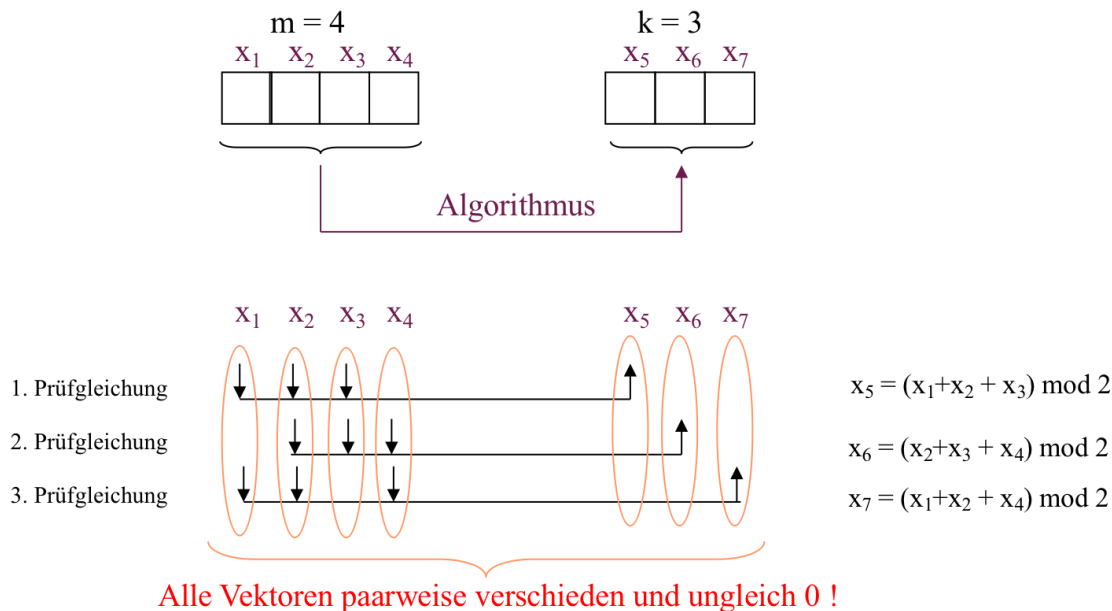
6.2. HAMMING BLOCKCODE

Beim Hamming-Code werden Gleichungen basierend auf den einzelnen Stellen des Codewortes definiert. Ein Codewort ist gültig, wenn es all diese Gleichungen erfüllt.

Das **Generatorpolynom** entspricht der ersten Prüfgleichung. Der Code hat $m + k$ Stellen.

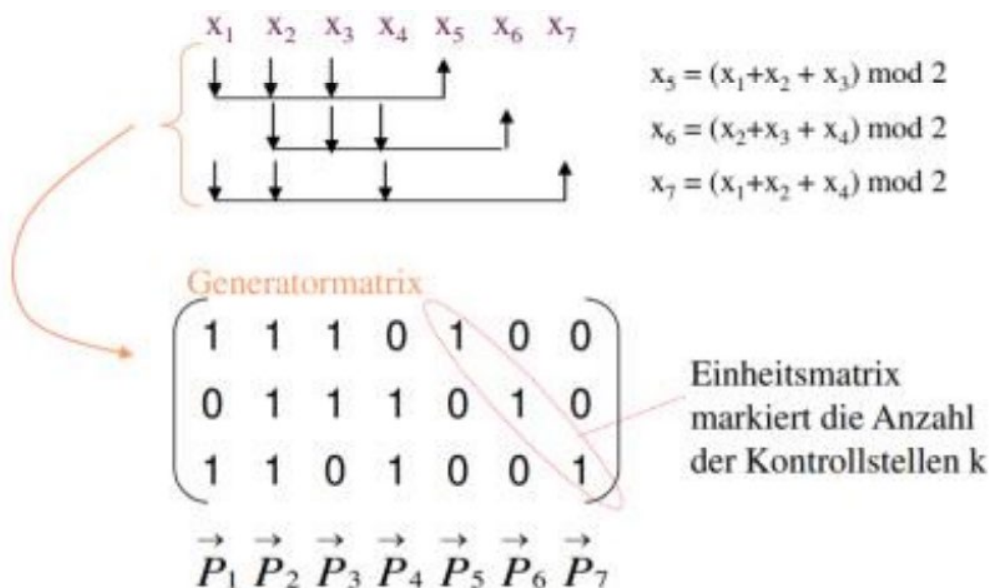
Anzahl Kontrollstellen entspricht Anzahl Prüfgleichungen. Wenn es nur eine Prüfgleichung gibt, gibt es auch nur eine Kontrollstelle.

Anzahl Codeworte: $2^{\text{Anzahl Stellen}}$



Interpretation: wird eine Stelle des CW verletzt, so werden jeweils andere Kombinationen von Prüfgleichungen verletzt, d.h. es müsste ein Fehlersyndrom geben, dass es erlaubt, den Fehlerort zu lokalisieren.

Generatormatrix



Formell können wir nun definieren

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

Fehlersyndrom

Bei einem fehlerhaften Codewort liefert uns die obige Formel keinen Nullvektor, sondern genau die Spalte der Generatormatrix, in der ein Fehler aufgetreten ist. (Funktioniert nicht, wenn mehr als ein Fehler aufgetreten ist)

Beispiel: Generatorpolynom $x^5 + x^3 + x^2 + 1 = 101101$, Fehler bei $x^5 = 100000$

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

Prüfmatrix nach Prüfgleichungen angeben

Bsp: Folgende Gleichungen sind gegeben

$$x_5 = x_1 + x_3 + x_4$$

$$x_6 = x_2 + x_3 + x_4$$

$$x_7 = x_1 + x_2 + x_3$$

Matrix:

x_1	x_2	x_3	x_4	k_1	k_2	k_3
1	0	1	1	1	0	0
0	1	1	1	0	1	0
1	1	1	0	0	0	1

Kontrollstellen: Anzahl der Spalten der Einheitsmatrix, hier 3

Gültige Codeworte: $2^m = 2^{\text{Anzahl Spalten ohne Prüfmatrix}} = 2^4 = 16$

Hamming-Distanz: Spalten müssen $\text{mod } 2 \equiv 0$ ergeben

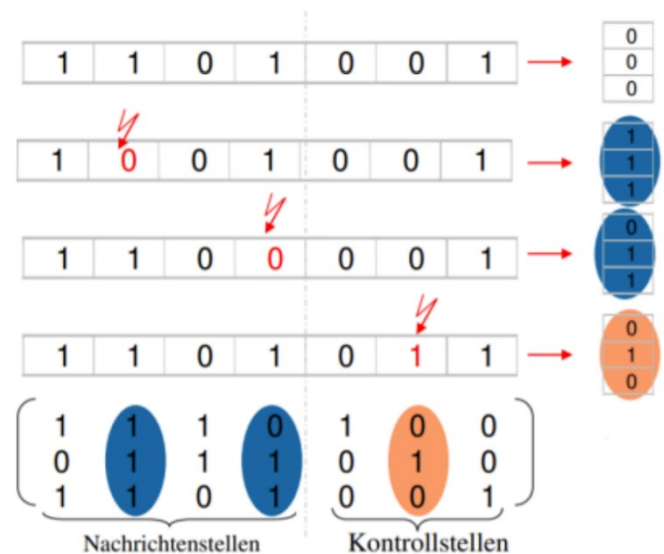
- Spalte mit am meisten 1er nehmen + Spalte mit am zweit-meisten 1er.
- Welche Spalten müssen noch ergänz werden, damit alles $\text{mod } 2 \equiv 0$ ergibt (minimale Anzahl)?

$$\begin{array}{ccc} x_3 & x_4 & \\ 1 & 1 & \\ 1 & 1 & \\ 1 & 0 & \end{array} \begin{array}{l} + ? \\ + ? \\ + ? \end{array} \begin{array}{l} \text{mod } 2 \equiv 0 \\ \text{mod } 2 \equiv 0 \\ \text{mod } 2 \equiv 0 \end{array} \Rightarrow \begin{array}{ccc} x_3 & x_4 & k_3 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{array} \begin{array}{l} \text{mod } 2 \equiv 0 \\ \text{mod } 2 \equiv 0 \\ \text{mod } 2 \equiv 0 \end{array}$$

- Anzahl verwendete Spalten = Hamming-Distanz.
In diesem Fall = 3, das bedeutet, erkennbare Fehler = 3 - 1 = 2, Korrigierbare Fehler = 3 - 2 = 1

Fehlersyndrom, falls x_3 und x_4 gestört sind:

$$\begin{array}{ccc} 1 & 1 & 0 \\ 1 & 1 & = 0 \\ 1 & 0 & 1 \end{array}$$



\Rightarrow Formell können wir auch sagen: $\vec{Z} = \sum_i x_i * \vec{P}_i \text{ mod } 2$

6.3. ZYKLISCHE CODES

Idee: Generatormatrix kann durch ein Generatorpolynom beschrieben werden.

Ziel: Vereinfachte Berechnung der Kontrollstellen durch rückgekoppelte Schieberegister.

Das Generatorpolynom lässt sich in der Polynom- und Binärschreibweise notieren.

- **Polynom:** $G(u) = u^3 + u + 1$
- **Binär / Vektor:** $G(u) = (g_3 \ g_2 \ g_1 \ g_0) = (1 \ 0 \ 1 \ 1) = [1 \ 0 \ 1 \ 1]$

Der höchste Grad des Polynoms bestimmt die Anzahl der Kontrollstellen, hier 3

Ermitteln der Kontrollstellen (Nicht Anzahl, sondern welche)

Die Berechnung der Kontrollstellen einer gegebenen Nachricht funktioniert über die Polynomdivision (=Mehrfachaddition).

- Beginne mit der Nachricht
- Schreibe nun unter die erste 1 das Generatorpolynom aus der Aufgabe hin
- Berechne die Summe mit *mod2*
- Wiederhole mit dem aktuellen Resultat, bis alle Kontrollstellen berechnet wurden
- Sollte das Generatorpolynom länger sein als das Codewort (Raushängt wenn es noch einmal darunter gesetzt wird) ist das Codewort **ungültig**

$$\begin{array}{r}
 \begin{array}{ccccccc}
 u^6 & u^5 & u^4 & u^3 & u^2 & u^1 & u^0 \\
 1 & 0 & 0 & 0 & \cdot & \cdot & \cdot \\
 \% & 1 & 0 & 1 & 1 & & \\
 \hline
 0 & 0 & 1 & 1 & 0 & 0 & \\
 & & 1 & 0 & 1 & 1 & \\
 \hline
 0 & 1 & 1 & 1 & 0 & & \\
 & & 1 & 0 & 1 & 1 & \\
 \hline
 0 & 1 & 0 & 1 & & &
 \end{array}
 \end{array}$$

Nachricht im Bild: 1 0 0 0, Generator $G(u) = (1 \ 0 \ 1 \ 1)$,
Kontrollstellen: 1 0 1

Über die Polynomdivision kann auch bestimmt werden, ob ein Codewort **gültig** ist oder **nicht**. Wenn die **Kontrollstellen** ans Codewort **angehängt** geteilt durch den Generator 0 ergibt, ist das Codewort **gültig**. Bei einem fehlerhaften Codewort liefert uns die Polynomdivision genau die **Spalte**, in der ein **Fehler aufgetreten** ist.

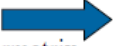
$$\begin{array}{r}
 \begin{array}{ccccccc}
 u^6 & u^5 & u^4 & u^3 & u^2 & u^1 & u^0 \\
 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
 \% & 1 & 0 & 1 & 1 & & \\
 \hline
 0 & 0 & 1 & 1 & 1 & 0 & \\
 & & 1 & 0 & 1 & 1 & \\
 \hline
 0 & 1 & 0 & 1 & 1 & & \\
 & & 1 & 0 & 1 & 1 & \\
 \hline
 0 & 0 & 0 & 0 & & &
 \end{array}
 \end{array}
 \Rightarrow \text{Codewort gültig!}$$

\Rightarrow Codewort: 1000101, Generator: $G(u) = (1 \ 0 \ 1 \ 1)$

Mit einem **gültigen Codewort** kann also die **Generatormatrix hergeleitet** werden, indem jedes Bit einmal invertiert wird.

Gültiges Codewort: 1 0 0 0 1 0 1

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


 Generatormatrix

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

Diese Darstellung der Polynomdivision wird als «Mehrfachaddition» bezeichnet, funktioniert aber identisch. Ist die Anzahl der Kontrollstellen bekannt, so kann man sich die Berechnung sparen (Einheitsmatrix)

6.4. SPEZIELLE CODES

Zyklischer Hamming-Code / Blockcode

Anzahl Kontrollstellen = k

Anzahl Nachrichtenstellen = m

Anzahl Codestellen (Nachrichtenstellen +
Kontrollstellen) = n

Anzahl Codewörter = 2^n

Anzahl gültige Codeworte = 2^m

$n = 2^k - 1 = m + k$, d. h. $m = n - k$

Welches sind die gültigen Codeworte?

Mehrfachaddition mit Vektor des Generatorpolynoms

Zyklischer Abramson-Code (CRC-Code)

Hammingdistanz $h=4$

Diese werden gebildet durch die Multiplikation eines
primitiven Polynoms mit dem Term $(1+x)$

Abramson-Code: $g(x) = p(x) (1+x)$

Bsp.:

$$g(x) = (1+x+x^3) (1+x)$$

$$g(x) = 1+x^2+x^3+x^4$$

Hammingdistanz $h=3$

Diese werden gebildet durch sogenannte
primitive Polynome $p(x) = g(x)$:

$$p(x) = 1+x+x^3$$

$$p(x) = 1+x+x^4$$

$$p(x) = 1+x^2+x^5$$

$$p(x) = 1+x+x^6$$

$$p(x) = 1+x^3+x^7$$

$$p(x) = 1+x^2+x^3+x^4+x^5+x^6+x^7$$

$$p(x) = 1+x^2+x^3+x^4+x^5+x^8$$

$$p(x) = 1+x^4+x^9$$

$$p(x) = 1+x^3+x^{10}$$

$$p(x) = 1+x^2+x^{11}$$

$$p(x) = 1+x+x^4+x^6+x^{12}$$

$$p(x) = 1+x+x^3+x^4+x^{13}$$

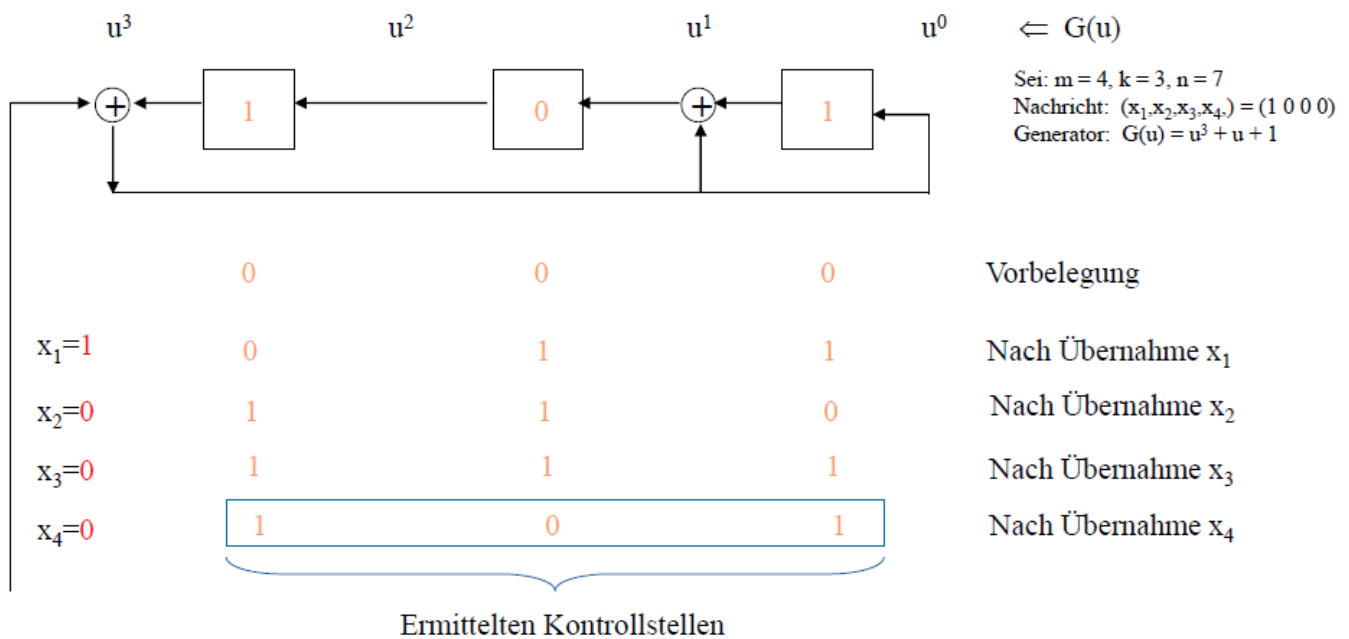
$$p(x) = 1+x^2+x^6+x^{10}+x^{14}$$

$$p(x) = 1+x+x^{15}$$

$$p(x) = 1+x^5+x^{23}$$

$$p(x) = 1+x+x^2+x^4+x^5+x^7+x^8+x^{10}+x^{11}+x^{12}+x^{16}+x^{22}+x^{23}+x^{26}+x^{32}$$

Ermittlung der Kontrollstellen durch rückgekoppeltes Schieberegister



\oplus Modulo 2 Addierer (XOR)

- Gleich viele Quadrate zeichnen wie höchster Grad vom Polynom (Jedes Polynom bildet eine Linie)
- Jeder Grad des Polynoms (jedes Element, welches eine 1 hat im Vektor) bildet ein \oplus . Höchster und tiefster Grad sind ausserhalb der Quadrate ganz links und ganz rechts.

7. FALTUNGSCODES

Faltungscodes erlauben die fortlaufende Codierung eines kontinuierlichen Datenstroms (z.B. Stream), wobei keine Blockbildung oder Synchronisation benötigt wird. Gute Faltungscodes werden durch Rechnersimulation gefunden.

7.1. ENCODERSCHALTUNG

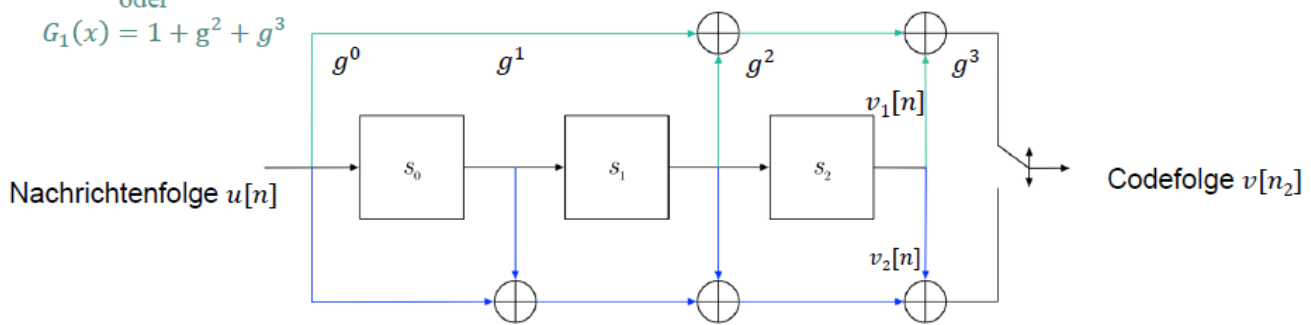
Bei Faltungscodes werden mehrere Generatorpolynome in eine Encoderschaltung abgebildet, wobei gilt:

- Jedes Generatorpolynom bildet eine «Linie»
- Der höchste Grad bestimmt die «Kastenzahl»
- Jeder Grad eines Polynoms bildet ein \oplus / XOR
- Encodergedächtnis = Anzahl Kästchen

$$\{g_1\} = \{1 \ 0 \ 1 \ 1\}$$

oder

$$G_1(x) = 1 + g^2 + g^3$$



$$\{G_2\} = \{1 \ 1 \ 1 \ 1\}$$

oder

$$G_2(x) = 1 + g + g^2 + g^3$$

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

Die Speicherplätze s_0, s_1, s_2 sind mit 0
vorbelegt.

$u[n]$	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	-	-	-

Ausgangszustand erreicht

Encoder Schaltung (2,1,3) / (Output, Input, Gedächtnis)

Tailbits = Encodergedächtnis (m) = 3

Code-Rate für 185 Bits:

$$R = \frac{\text{Input} * \text{Bits}}{\text{Output} * (\text{Bits} + \text{Tailbits})} = \frac{1 * 185}{2 * (185 + 3)} = 0.492$$

Beispiel skizzieren einer Encoder-Schaltung (1,2,2)

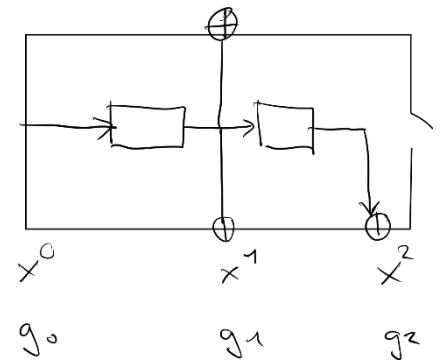
Generatorpolynome:

- $g_1(x) = 1 + x$ oder $\{1,1,0\}$
- $g_2(x) = 1 + x + x^2$ oder $\{1,1,1\}$

Encodergedächtnis = 2

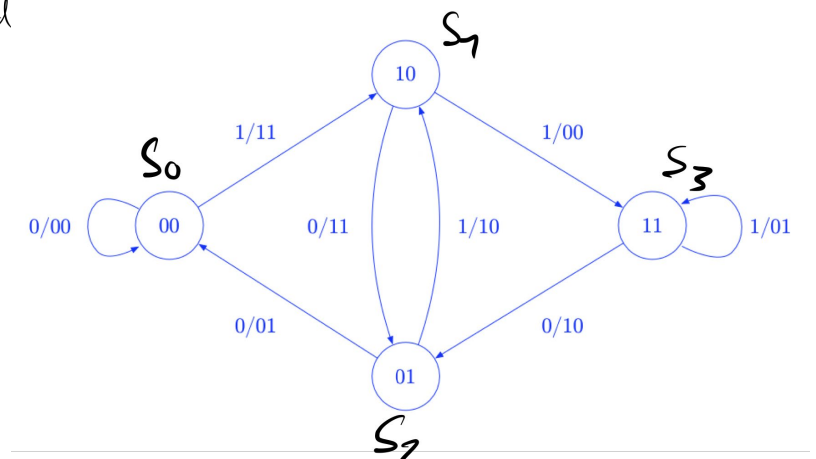
Impulsantwort Encoder: $\{g_1\} = \{1,1,0\}, \{g_2\} = \{1,1,1\}$

Impulsantwort Decoder: encode 1 0 0 $\rightarrow \{v[n]\} = \{11,11,01\}$

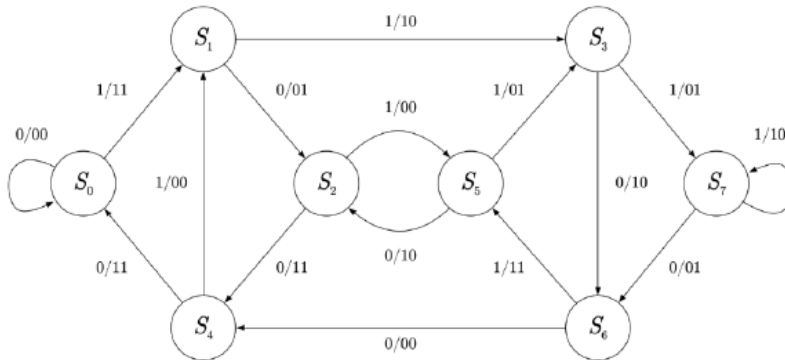
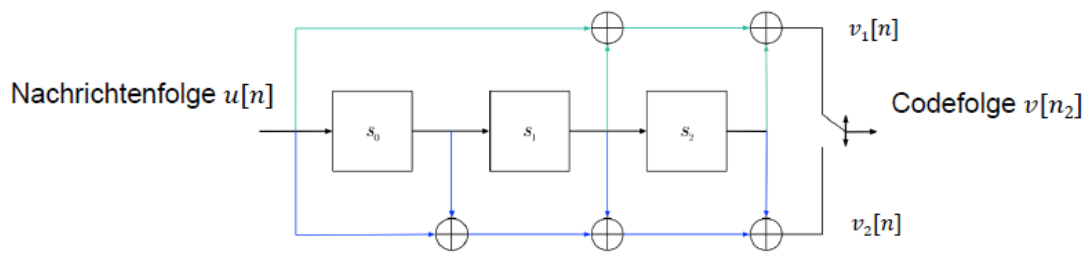


7.2. ZUSTANDSDARSTELLUNG DES (1,2,2) ENCODERS

Input	Register	Zustand	Output	Endzustand
0 0 0		s_0	0 0	s_0
1 0 0		s_0	1 1	s_1
0 1 0		s_1	1 1	s_2
1 1 0		s_1	0 0	s_3
0 0 1		s_2	0 1	s_0
1 0 1		s_2	1 0	s_1
0 1 1		s_3	1 0	s_2
1 1 1		s_3	0 1	s_3



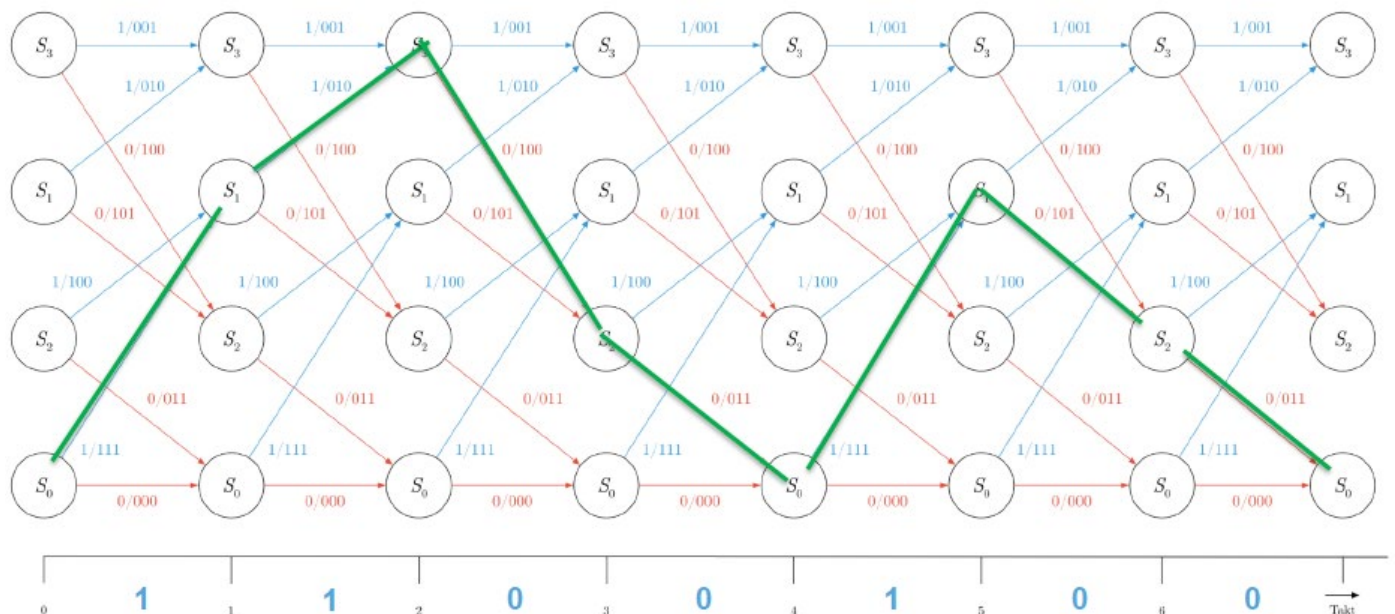
7.3. ZUSTANDSDARSTELLUNG DES (2,1,3) ENCODERS



Zustandsgrößen			Zustand s_i ($i = s_0 \cdot 2^0 + s_1 \cdot 2^1 + s_2 \cdot 2^2$)
s_0	s_1	s_2	
0	0	0	0
1	0	0	1
0	1	0	2
1	1	0	3
0	0	1	4
1	0	1	5
0	1	1	6
1	1	1	7

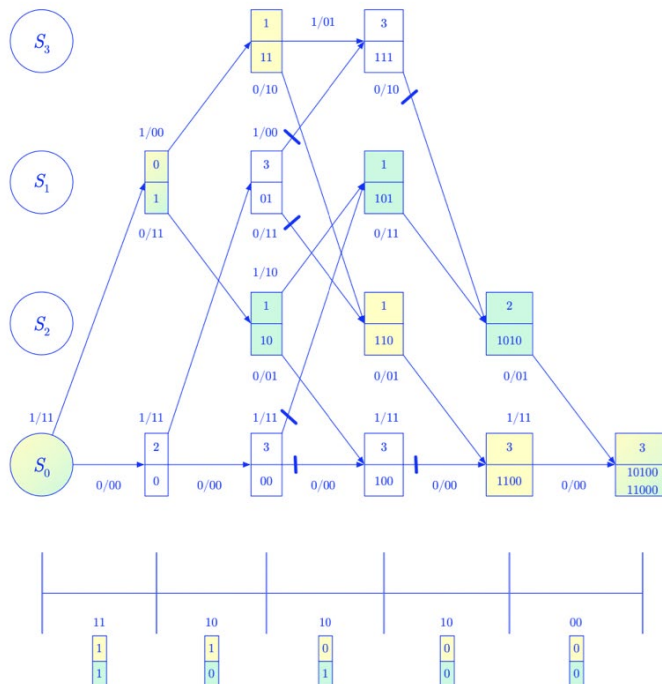
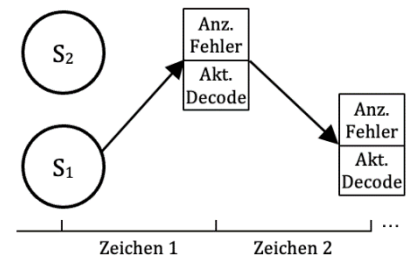
7.4. NETZDIAGRAMM

Das Netzdiagramm bezeichnet ein aufgespanntes Zustandsdiagramm bei einer Folge von Eingabezeichen.



Wir können damit Zeichenketten decodieren. Dekodieren der Zeichenfolge $Z = \{11,10,10,10,00\}$ (anhand vom vorherigen Beispiel).

Es gibt zwei Wege, die zum gleichen Gewicht führen. Man kann deshalb nicht sagen, ob $\{u'[n]\} = \{1, 1, 0\}$ oder $\{u'[n]\} = \{1, 0, 1\}$ war. Merke: Die Tailbits wurden hier bereits entfernt.



7.5. DEFINITIONEN

Anzahl Zustände

Zustände = Entscheidungsmöglichkeiten^{Gedächtnisstellen}

Ausgehend vom oberen Schieberegister (2 Entscheidungsmöglichkeiten, 3 Gedächtnisstellen: $2^3 = 8$ Zustände (kann im Zustandsdiagramm kontrolliert werden)

Anzahl Tailbits

Die Anzahl Tailbits **entspricht der Anzahl Gedächtnisstellen**, damit sicher wieder alle Gedächtnisstellen mit 0 belegt sind.

Gewicht

Das Gewicht des Codes ist die **Anzahl von Bitstellen** eines Codeworts, die von «0» **verschieden** sind.

Fundamentalweg

Ist der (Teil-)**Weg eines Codes**, der im Zustand S_0 beginnt und wieder im Zustand S_0 endet. Die Analyse der Fundamentalwege liefert die Struktur des Faltungscodes.

Metrik:

- **I**: bezeichnet den Zustandsübergang, der durch «1» ausgelöst wird.
- **DI**: bezeichnet die Anzahl der durch den Übergang zur Codefolge hinzukommenden «1» Bitstellen (Gewichtszunahme)
- **J**: Eine Zählvariable, die die Anzahl der Übergänge zählt
- Jede Kante eines Fundamentalweges lässt sich durch das Triplet **(I DI J)** beschreiben (Kantengewicht)

Impulsantwort

Man sendet eine 1 und dann eine Anzahl von Nullen, die dem *Grad des Polynoms entspricht*. Z.B. {1,0,0,0}

7.5.1. Was ist ein «guter» Faltungscode?

Ein Faltungscode ist gut, wenn der Unterschied der Ausgabe bei einem Zustandsübergang immer maximal ist.

$$S_1 = 00: u = 0 \rightarrow v = 00$$

$$u = 1 \rightarrow v = 11$$

⇒ Maximaler Unterschied der Ausgaben (2 Zeichen)

⇒ Siehe «Zustandsdiagramm»-Beispiel für einen guten Code.

7.5.2. Generatorpolynome für optimale Faltungscode

m	$R = \frac{1}{2}$			$R = \frac{1}{3}$				$R = \frac{1}{4}$				
	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

g_i in oktaler
Schreibweise

[Martin Werner, Information und Codierung, vieweg]