Digitale Rechenanlagen

MARIÁN VAJTERŠIC

Fachbereich Computerwissenschaften
Universität Salzburg
marian@cosy.sbg.ac.at
Tel.: 8044-6344

28. September 2017

Vorlesungszeiten

Montag 11:00–12:30, HS T01
Sprechstunden: jeweils nach der Vorlesung

Vorlesungsinhalte

- Grundlagen und Prinzipien
- Informationstheorie
- Zahlendarstellung
- Schaltnetze, Schaltwerke

Prüfung

Schriftlicher Test (2 Stunden) bestehend aus 3 Blöcken:

- Multiple Choice
- Theoriefragen
- Rechenaufgaben

Drei Termine pro Semester (Schwierigkeitsgrad immer gleich).

Literatur

- J. L. Hennessy, D. A. Patterson: Rechnerarchitektur. Vieweg Verlag, 1996
- K. Beuth: Digitaltechnik. Vogel Buchverlag, 2006
- L. Borucki: Digitaltechnik. Teubner Verlag, 1996
- A. S. Tanenbaum, T. Austin: Rechnerarchitektur: Von der digitalen Logik zum Parallelrechner. Pearson, 2014

Folien und Proseminar

- **Vorlesungsfolien:** elektronisch abrufbar unter (PLUSonline)
  https://online.uni-salzburg.at
  VO Digitale Rechenanlagen → LV-Unterlagen
- **Proseminar-Zettel:** elektronisch abrufbar unter (PLUSonline)
  https://online.uni-salzburg.at
  PS Digitale Rechenanlagen → LV-Unterlagen

Kapitel 1: Grundbegriffe der Informationstheorie

Ziel: Definition, Darstellung und Messbarkeit von Information.

- Information: zentraler Begriff der Informatik.
- Informatik: Wissenschaft, die sich mit Information und deren Verarbeitung mit **formalen** und **technischen** Methoden beschäftigt.
  (**Theoretische**, **Angewandte**, **Technische** Informatik)

Wie wird **Information** repräsentiert?

Durch jene Signale, die zwei (oder mehr) Systeme miteinander austauschen.

Beispiele für **Systeme** und von diesen verwendete **Signale**:

System 1		System 2	Signale
Planet	$\rightarrow$	Planet	Schwerkraft
Sonne	$\rightarrow$	Mensch	Wärme, Licht
Pflanze	$\rightarrow$	Mensch	Farbe, Geruch
Mensch	$\rightarrow$	Mensch	Sprache, Schrift

tabelle stellt moegl signale (ie information) dar

Codierung

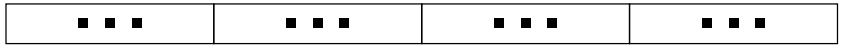
Übersetzung einer speziellen Repräsentation von Information in eine andere Repräsentation.
Beispiel: Schrift [Repräsentation von Sprache] $\rightarrow$ Morsezeichen

Buchstabe	A	E	K	T	SOS
Morsezeichen	$\cdot -$	$\cdot$	$- \cdot -$	$-$	$\cdot\cdot\cdot---\cdot\cdot\cdot$

- Wozu Codierung?
  Einerseits ermöglicht Codierung, **komplexe** Nachrichten (Signale) zu senden, sowie andererseits die Anpassung des Nachrichten-Typs an das **verwendete Medium**.

Wahl der Codierung

- **Anpassung an Information** (Musik, Video, Programme, ...): Nicht mit selber Codierung übertragen. Z. B.: Sprache eignet sich schlecht zur Übertragung von Videos.
- **Beeinflussung der Redundanz** (Informationsüberfluss): Wenn die Datenmenge redundant ist, gelingt es leichter, die Datenmenge zu komprimieren. Z. B.: Die Redundanz der deutschen Sprache ist höher als die der chinesischen Sprache.
- **Variable (unterschiedliche) oder feste Länge**
  - Code **variabler** Länge:
    z. B.: Morsecode: variable Länge (A: 2 Zeichen; E: 1 Zeichen), deshalb ist eine **Pause notwendig**.
  - Code **fester** Länge:
    z. B.: ASCII: feste Länge [A: (0)1000001, Z: (0)1011010], die Trennung der Zeichen ist klar.



Binärcodierung

Unsere Vorlesung beschäftigt sich ausschließlich mit Binärcodierung.

Binärcodierung: Abbildung der Information auf eine Kette aus den Zeichen 0 und 1, die in digitalen Rechenanlagen als Spannung/Strom AUS und Spannung/Strom EIN repräsentiert und verarbeitet wird.

Bemerkung: Morsecode ist kein Binärcode, sondern ein Ternärcode:
besteht aus 3 Zeichen ('·', '–', ' ') .

Binärcode technisch – 2 Signale:
1: **kleine** Spannung [5V] vorhanden.
0: **keine** Spannung [0V] vorhanden.

Bemerkung zur Codierung:

Das menschliche Genom (entschlüsselt 2002) kann durch quaternäre Codierung mit vier Buchstaben (A, T, C, G) repräsentiert werden (Aufbau auf 4 Basen).

$$\text{KLAR: Hier ist Binärcodierung möglich} \longrightarrow \left\{ \begin{array}{l} \text{mit } 2^2 \text{ Zeichen } \{00, 01, 10, 11\} \\ \text{ist eine Binärcodierung möglich,} \\ \text{die Codierungsketten werden} \\ \text{jedoch wesentlich länger} \\ \text{(doppelt so lang)} \end{array} \right.$$

Codes und Codierung (Formalisierung)

[Zeichenvorrat, Alphabet]

Definition

[Zeichenvorrat, Alphabet]: Eine (endliche) Menge A von Zeichen heißt *Zeichenvorrat*. Ein **linear geordneter** Zeichenvorrat heißt *Alphabet.*

Beispiel:

Ein elementarer Zeichenvorrat ist die Menge $B = \{0, 1\}$ (oder auch $B = \{L, H\}$).
Ein Element der Menge B nennen wir Binärzeichen oder Bit (Binary Digit).
Falls die lineare Ordnung durch $0 < 1$ definiert wird, ist B ein Alphabet
(0 steht (ist geordnet) vor 1).

[Wörtermenge über einem Zeichenvorrat]

Definition

[Wörtermenge über einem Zeichenvorrat] : Die Menge A^* der **endlichen** Zeichenfolgen über einem Zeichenvorrat A nennen wir **die Menge der Wörter** über A.

Beispiel:

Die Menge $B^* = \{0, 1\}^*$ bezeichnen wir als die **Menge der Binärwörter**.
Die Elemente der Menge B^n nennen wir **n-Bit-Wörter**
[die selbst wiederum einen Zeichenvorrat bilden können].

[Wörtermenge über einem Zeichenvorrat]

- Ist die Menge $ZV = \{*, \Delta\}$ mit $\Delta < *$ ein Alphabet?
  Nein, weil linear geordnet impliziert, dass mit dem kleineren begonnen werden muss.
  $\longrightarrow A = \{\Delta, *\}$ mit $\Delta < *$ wäre ein Alphabet.
- Wenn $Q = \{A, C, D, B, X, ...\}$ [also Buchstaben vermischt], dann ist Q kein Alphabet [weil z. B. B vor C stehen sollte].

Beispiel:

B^4:	4-Bit-Binärwörter	Nibbles
B^8:	8-Bit-Binärwörter	Bytes
B^{16}:	16-Bit-Binärwörter	Half Words (früher Words)
B^{32}:	32-Bit-Binärwörter	Words
B^{64}:	64-Bit-Binärwörter	Double Words

[Codes und Codierung]

Definition

[Code oder Codierung]: Eine **Abbildung** zwischen zwei Zeichenvorräten A und X, $c : A \to X$ bzw. eine Abbildung von Wörtern über diesen Zeichenvorräten, $c : A^* \to X^*$, nennen wir *Code* oder *Codierung*.

Bemerkung: Für **spezielle Codelängen** (Anzahl der Zeichen) $m, n \in \mathbb{N}$ und $X \equiv B^n$ sieht die Abbildung folgendermaßen aus: $c : A^m \to B^n$

A^m: k^m Wörter der Länge m
k: # Zeichen in A
B^n: 2^n Binärwörter der Länge n

[Codes und Codierung]

Beispiel:

Chiffrierung: Spezielle Codierung, wobei $\underline{m = 1}$ und $\underline{n}$ beliebig.

$A = \{0, 1, 2, \ldots 7, 8\} \equiv A^{\underline{m}} = A^{\underline{1}}$
$B^{\underline{4}} = \{0000, ..., 1111\} \equiv B^{\underline{n}}$

$$c: \quad \left.\begin{array}{c} 0 \to 0000 \\ 1 \to 0001 \\ \ldots \\ \ldots \\ \ldots \\ 7 \to 0111 \\ 8 \to 1000 \end{array}\right\} \text{Chiffrierung der Zahlen 0–8}$$

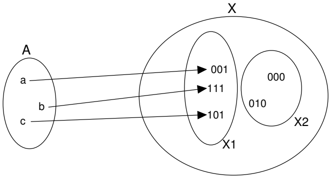
Decodierung

Ein sinnvoller Code sollte auch umkehrbar (decodierbar) sein. Wir müssen von der Codierung $c : A \longrightarrow X$ fordern, dass sie eine *injektive* Abbildung ist. D. h. dass **verschiedene** Zeichen (Wörter) aus A auf **verschiedene** Zeichen (Wörter) aus X abgebildet werden.

Damit ist die **Umkehrabbildung** $d : \{c(a)|a \in A\} \longrightarrow A$ **eindeutig** und es gilt für alle $a \in A : d(c(a)) = a$. Diese **Umkehrabbildung** nennen wir *Decodierung*.

Decodierung

→ Damit der Code umkehrbar ist, muss **jedes durch die Codierung gebildete Element der Bildmenge** genau einem Element der Urbildmenge zugeordnet sein.

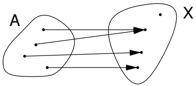


X1: Die Menge der durch die Codierung gebildeten Codewörter.

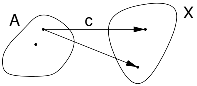
Erklärung: [Abbildung]

Definition

[Abbildung] $c : A \to X$, wobei $\forall a \in A \;\exists\, !\, x \in X$, sodass $c(a) = x$



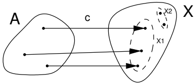
Dieses Beispiel zeigt weder eine injektive noch eine surjektive Abbildung, aber es ist eine Abbildung.

VERBOTEN	
	1. Für das obere Element in A existiert mehr als 1 Bild. 2. Für das untere Element in A existiert kein Bild $c(a)$.

Erklärung: [Injektive Abbildung]

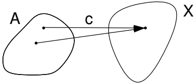
Definition

[Injektive Abbildung] c: $A \to X$, wobei $\forall a_1, a_2 \in A$: $a_1 \neq a_2 \to c(a_1) \neq c(a_2)$



$X = X1 \cup X2$
$X1 = \{x \in X : \exists a : c(a) = x\}$
(Also für jedes $x \in X1$ gibt es **ein Urbild aus A**.)
$X2$: *Kein Urbild.*

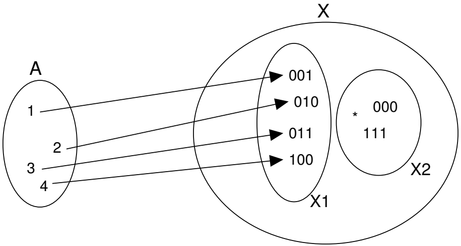
VERBOTEN



3. $a_1 \neq a_2 : \quad c(a_1) = c(a_2)$

Erklärung: [Injektive Abbildung]

Beispiel: injektive Abbildung

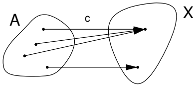


Beispiel: injektive Abbildung

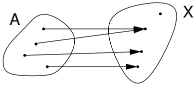
Erklärung: [Surjektive Abbildung]

Definition

[Surjektive Abbildung] $c : A \to X$, wobei $\forall x \in X \, \exists$ (mindestens ein) $a \in A$ sodass $c(a) = x$



VERBOTEN



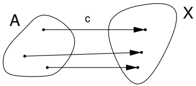
4. ein Element aus X hat kein Urbild in A

Erklärung: [Bijektive Abbildung]

Definition

[Bijektive Abbildung] $c : A \to X$, wobei $\forall x \in X \; \exists \, ! \; a \in A$ sodass $c(a) = x$ (injektiv <u>und</u> surjektiv).

Also: Jedes Element der Bildmenge ist genau einem Element der Urbildmenge zugeordnet.
Also: Für die Decodierung muss die Abbildung injektiv sein [$X2$ kann $\neq \emptyset$ sein] (im Spezialfall bijektiv [$X2 = \emptyset$]).



Also hier: $X2 = \emptyset$

Eigenschaften von Codes

[Eigenschaft von Codes fester Länge]

Prinzipiell unterscheiden wir zwischen Codes **fester** Länge und Codes **variabler** Länge. In digitalen Rechenanlagen finden fast ausschließlich Codes fester Länge Verwendung.

Definition

[Eigenschaft von Codes fester Länge] Der Hammingabstand H_d (Hamming Distance) zweier (binärer) Codewörter r, s der Länge n ist gegeben durch

$$H_d(r,s) = \sum_{i=1}^{n} d_i \quad \text{mit} \quad d_i = \begin{cases} 1 & \text{wenn } r_i \neq s_i \\ 0 & \text{sonst.} \end{cases}$$

[Eigenschaft von Codes fester Länge]

Beispiel [n=4]

$$\begin{bmatrix} r = r_1, r_2, r_3, r_4 \\ s = s_1, s_2, s_3, s_4 \end{bmatrix}$$ Man durchläuft die Folge, vergleicht Bits und summiert von 1 bis 4.

- $$\begin{array}{lcccc} \text{r: } 0101 & r_1 = 0 & r_2 = 1 & r_3 = 0 & r_4 = 1 \\ \text{s: } 1000 & s_1 = 1 & s_2 = 0 & s_3 = 0 & s_4 = 0 \\ \hline \text{d: } 1101 & d_1 = 1 & d_2 = 1 & d_3 = 0 & d_4 = 1 \\ & (ungleich) & (ungleich) & (gleich) & (ungleich) \end{array}$$

  $$\rightarrow \sum_{i=1}^{4} d_i = 3 = H_d(r, s)$$
- $$\begin{array}{ll} \text{r: ABAC} & \\ \text{s: CABA} & \rightarrow H_d(r, s) = 4 \end{array}$$

Bemerkung: Wenn $H_d(r, s) = 0$, dann sind die Wörter r und s gleich.

[Hammingdistanz einer Codierung]

Definition

[Hammingdistanz einer Codierung]

- **Minimale** Hammingdistanz einer Codierungsabbildung $C : A \rightarrow B^n$
  $MIN\ H_d(C) = MIN\{H_d(C(r), C(s)) | r, s \in A \wedge r \neq s\}$
  [Der kleinste Hammingabstand aller codierten Wörter aus A]
- **Maximale** Hammingdistanz einer Codierungsabbildung $C : A \rightarrow B^n$
  $MAX\ H_d(C) = MAX\{H_d(C(r), C(s)) | r, s \in A \wedge r \neq s\}$

Bemerkung: Wenn nur $H_d(C)$ geschrieben steht, wird $MIN\ H_d(C)$ damit gemeint.

[Hammingdistanz einer Codierung]

Beispiel: C1 (n=3)

$\underline{C1}: A \to X \qquad A = \{0, 1, ..., 7\},\ X = \{000, 001, ..., 111\}$

- $0 \to 000$ }=C1(0) $\underline{MIN\ H_d(C1) = 1}$
- $1 \to 001$ (Da der <u>kleinste</u> Abstand zwischen zwei codierten Wörtern 1 ist, z. B. zwischen 000 und 001.)
- $2 \to 010$
- $3 \to 011$
- $4 \to 100$
- $5 \to 101$ $\underline{MAX\ H_d(C1) = 3}$
- $6 \to 110$ (Da der <u>größte</u> Abstand 3=n ist, z. B. zwischen 000 und 111.)
- $7 \to 111$

[Hammingdistanz einer Codierung]

Beispiel: C2 (n=3) und C3 (n=3)

- $\underline{C2}: A \to X \qquad A = \{r_1, r_2, r_3, r_4\}, \ X = \{000, 011, 101, 110\}$

  $r_1 \to 000 \quad \} = C2(r_1) \quad \underline{MIN\ H_d(C2) = 2}$

  $r_2 \to 011$

  $r_3 \to 101 \qquad \underline{MAX\ H_d(C2) = 2}$

  $r_4 \to 110$

- $\underline{C3}: A \to X \qquad A = \{\square, \Delta\}, \ X = \{000, 111\}$

  $\square \to 000 \quad \} = C3(\square)$

  $\Delta \to 111 \quad \} = C3(\Delta) \quad \underline{MIN\ H_d(C3) = 3} = \underline{MAX\ H_d(C3)}$

[Hammingdistanz einer Codierung]

Beispiel: C4 (n=5)

$\underline{C4}$: $A \to X$ $\quad A = \{r_1, r_2, r_3, r_4, r_5\}$,
$X = \{10110, 00001, 00010, 11111, 10000\}$

- $r_1 \to 10110$
  $\left.\begin{array}{l} r_2 \to 00001 \\ r_3 \to 00010 \end{array}\right\} MIN\ H_d(C4) = 2$
  $\left.\begin{array}{l} r_4 \to 11111 \\ r_5 \to 10000 \end{array}\right\} MAX\ H_d(C4) = 4$

Wozu dient die Hammingdistanz?

- zur Beschreibung von Codes
- zur Fehlererkennung
- zur Fehlerkorrektur

Fehlererkennung: [Gültige Wörter einer Abbildung]

Definition

[Gültige Wörter einer Abbildung] Die Menge $G_C = \{C(r) | r \in A\}$ heißt die Menge gültiger Wörter der Codierung C.

Beispiel:

für $\underline{C3}$: $G_{C3} = \{000, 111\}$

Fehlererkennung: [k-Bit-Fehler]

Definition

[k-Bit-Fehler] Wir sagen, dass bei einer Codierung von $a \in A$ ein k-Bit-Fehler entstanden ist [k Bits umgekippt], wenn statt $C(r)$ [gültiges Wort] ein anderes Wort $[C(r)]'$ angekommen ist, für welches $k = H_d([C(r)]', C(r))$ gilt.

Beispiel:

$\underline{C2}$: $A \to X \qquad A = \{r_1, r_2, r_3, r_4\}, \; X = \{000, 011, 101, 110\}$
Wenn durch die Codierung von r_3 $[C2(r_3)]'$=$\underline{0}\ \underline{1}\ \underline{0}$ entsteht (anstelle von $C2(r_3)$=$\underline{1}\ \underline{0}\ \underline{1}$)
$\to$ $\underline{3}$ Bits umgekippt.
$\to$ $\underline{k{=}3}$ (Ein $\underline{\text{3-Bit}}$-Fehler ist entstanden.)

Fehlererkennung: [k-Bit-Fehler]

Es gilt: $1 \leq k \leq n$ (0 ist weggelassen, da in diesem Fall kein Fehler entstanden ist.)

Der k-Bit-Fehler eines fehlerhaften Wortes $[C(r)]'$ kann erkannt werden, wenn dieses kein gültiges Wort der Abbildung C ist. D. h. wenn $[C(r)]' \neq C(s)\ \forall s \in A$.
Wenn hingegen $[C(r)]' \in G_C$, dann kann der k-Bit-Fehler von $[C(r)]'$ nicht erkannt werden.

Fehlererkennung: [k-Bit-Fehler]

Beispiel: Siehe $C4$

$[C4(r_1)]' = \underline{0}\ \underline{1}\ 1\ 1\ 0$ (statt $C4(r_1) = \underline{1}\ \underline{0}\ 1\ 1\ 0$)
Der <u>2-Bit</u>-Fehler kann <u>erkannt</u> werden, weil
$01110 = [C4(r_1)]' \notin G_{C4} = \{10110, 00001, 00010, 11111, 10000\}$.

Falls $[C4(r_1)]' = 1\ \underline{1}\ 1\ 1\ \underline{1}$ (statt $C4(r_1) = 1\ \underline{0}\ 1\ 1\ \underline{0}$),
kann der <u>2-Bit</u>-Fehler <u>nicht erkannt</u> werden, da $[C4(r_1)]' \in G_{C4}$ und somit
$[C4(r_1)]' = C4(r_4)$.

Fehlererkennung: [k-Bit-Fehler]

Das fehlerhafte Wort $[C(r)]'$ kann (als fehlerhaft) erkannt werden, wenn k (seine Hammingdistanz zum gültigen Wort C(r)) **kleiner** als $MIN\ H_d(C)$ oder **größer** als $MAX\ H_d(C)$ des Codes C ist. Also für

$$\left.\begin{array}{ll} \text{i)} & k = 1, 2, ..., MIN\ H_d(C) - 1 \\ \text{ii)} & k = MAX\ H_d(C) + 1, ..., n \end{array}\right\} k\text{-Bit-Fehler erkennbar.}$$

Hingegen kann der k-Bit-Fehler des Wortes $[C(r)]'$ unter Umständen nicht erkannt werden, wenn

$MIN\ H_d(C) \leq k \leq MAX\ H_d(C)$.

In diesem Fall kann iii), muss aber nicht iv), $[C(r)]'$ ein gültiges Wort werden.

Fehlererkennung: [k-Bit-Fehler]

Beweis.

Fehler bei $[C(r)]'$ kann erkannt werden, wenn $k < MIN\ H_d(C)$.

Indirekt [durch Widerspruch]: Angenommen, es gilt $k = H_d([C(r)]', C(r)) < MIN\ H_d(C)$ und der Fehler **kann nicht erkannt werden.**

- $\rightarrow$ $[C(r)]' \in G_C$ ($Also$ $[C(r)]'$ ist ein gültiges Wort).
- $\rightarrow$ $H_d([C(r)]', C(s)) \geq MIN\ H_d(C)\ \forall s \in A$.
- $\rightarrow$ Wenn es für $\forall s \in A$ gilt, muss es auch für $r = s$ gelten.
- $\rightarrow$ $k = H_d([C(r)]', C(r)) \geq MIN\ H_d(C)$ ist, was ein Widerspruch zur Annahme ist.
- $\rightarrow$ Der Fehler ist in diesem Fall erkennbar.

$\square$

(Der Beweis ist analog auch für die übrigen Fälle.)

Fehlererkennung: [k-Bit-Fehler]

Beispiel:

- Die Codierung $C1$ mit $\boldsymbol{MIN\ H_d(C1) = 1}$, $\boldsymbol{MAX\ H_d(C1) = 3}$ lässt nicht zu, $\boldsymbol{k}$-Bit-Fehler für $k = 1, 2, 3$ zu erkennen, weil gilt:

$$MIN\ H_d(C1) \leq k \leq MAX\ H_d(C1)$$

$$1 \leq 1, 2, 3 \leq 3$$

und alle möglichen Änderungen eines gültigen Worts wieder ein gültiges Wort ergeben, da alle möglichen 3-Bit-Wörter gültige Wörter von C1 sind. Wenn z. B. 3 Bits bei $C1(1) = 001$ umfallen, dann ist [C1(1)]' $= 110$ wieder ein gültiges Wort. $\rightarrow$ Der Fehler ist nicht erkennbar.

Fehlererkennung: [k-Bit-Fehler]

Beispiel:

Bei $\underline{C2}$ mit $\boldsymbol{MIN\ H_d(C2) = MAX\ H_d(C2) = 2}$

a) **kann der Fehler erkannt werden, wenn:**

i) $\underline{1}$ $(= MIN\ H_d(C2) - 1)$ Bit umkippt oder wenn
ii) $\underline{3}$ $(= MAX\ H_d(C2) + 1 = n)$ Bits umkippen.

i) Wenn $[C2(r_1)]' = 00\underline{1}$ (statt $C2(r_1)$=$00\underline{0}$), kann der Fehler erkannt werden, weil 001 kein gültiges Wort von C2 ist.
ii) Wenn $[C2(r_4)]' = \underline{001}$ (statt $C2(r_4)$=$\underline{110}$), wird der Fehler erkannt, weil $001 \notin G_{C2}$.

b) **kann der Fehler nicht erkannt werden, wenn:**

iii) $MIN\ H_d(C2) \leq k \leq MAX\ H_d(C2),\ \ 2 \leq k \leq 2.$
$\rightarrow$ Wenn $k = \underline{2}$ Bits umkippen, ist der 2-Bit-Fehler unter Umständen nicht zu erkennen. Im Fall von C2 sind alle 2-Bit-Fehler nicht erkennbar, da jede mögliche Änderung von 2 Bits eines Worts von C2 ein anderes gültiges Wort von C2 ergibt.

iii) Wenn z. B. $[C2(r_3)]' = \underline{01}1$ [statt $C2(r_3)$=$\underline{10}1$] ankommt, ist dieser 2-Bit-Fehler nicht zu erkennen [weil $\underline{011}$ ein gültiges Wort $\underline{C2(r_2)}$ der Codierung $\underline{C2}$ ist].

Fehlererkennung: [k-Bit-Fehler]

Beispiel:

Bei $\underline{C4}$ mit $\boldsymbol{MIN\ H_d(C4) = 2, MAX\ H_d(C4) = 4}$ kann für

iv) $MIN\ H_d(C4) \leq k \leq MAX\ H_d(C4)$

also für $2 \leq k \leq 4$ unter Umständen der Fehler nicht erkannt werden.
$\rightarrow$ Muss nicht in jedem Fall unerkannt bleiben.

iv) Wenn z. B. [C4(r_5)]' $= 10\underline{111}$ [statt C4(r_5) $= 10\underline{000}$] ankommt ($k = 3$):
Da $10111 \notin G_{C4}$, bleibt der Fehler in diesem Fall nicht unerkannt, obwohl $k \in [MIN\ H_d(C4), MAX\ H_d(C4)]$ ist. ([] bedeutet geschlossenes Intervall.)

Fehlererkennung: [k-Bit-Fehler]

Da der k-Bit-Fehler **erkennbar** ist für $1 \leq k < MIN\ H_d(C)$ und $MAX\ H_d(C) < k \leq n$

→ Je größer $\boldsymbol{MIN\ H_d(C)}$ und je kleiner $\boldsymbol{MAX\ H_d(C)}$ wird (also je schmäler das Intervall $[MIN\ H_d(C), MAX\ H_d(C)]$ wird), umso **leichter** ist es, einen Fehler zu **erkennen**.

→ Umgekehrt, je **breiter** das Intervall $[MIN\ H_d(C), MAX\ H_d(C)]$ wird, umso **schwieriger** ist es, einen Fehler zu **erkennen**.

→ Im Extremfall $MIN\ H_d(C) = 1$ und $MAX\ H_d(C) = n$ [wie bei $C1$] ist unter Umständen gar kein k-Bit-Fehler erkennbar.

Spezielle Codes

Codes fester Länge

Paritätsbit

Einfacher Code zur Fehlersicherung: Generierung

Angenommen, wir hätten einen Code

$\underline{C_1} : A \longrightarrow B^n$ *mit* $H_d(C_1) = 1$, *wobei* $H_d(C_1) \equiv MIN\ H_d(C_1)$ *gilt*.

[D. h. die Hammingdistanz von 2 Codewörtern kann $1, 2, \ldots, n$ betragen.]

Aus C_1 wird ein Code C_2 konstruiert: $C_2 : A \longrightarrow B^{n+1}$ mit $H_d(C_2) = 2$,

$C_2(a) = C_1(a) \circ p(C_1(a))$ [also Code der Länge $n + 1$].

$\circ$: Symbol für die Konkatenation (Verknüpfung)

[Parität]

Definition

[Gerade Parität] $p : B^n \longrightarrow B^1$ definiert durch

$$p(C_1(a)) = \begin{cases} 1 & \text{Summe der 1-Bits in } C_1 \text{ ist ungerade} \\ 0 & \text{Summe der 1-Bits in } C_1 \text{ ist gerade} \end{cases}$$

Also mit Paritätsbit ist die Anzahl der 1-Bits im Codewort <u>gerade</u>.

Analog dazu ist die **ungerade Parität** definiert als $1 - p$.

[Gerade Parität]

Beispiel:

7-Bit-ASCII[1]-Code mit geradem Paritätsbit

Buchstabe	7-Bit-ASCII-Code	Paritätsbit
0	0110000	0
A	1000001	0
Z	1011010	0
a	1100001	1

Der 7-Bit-ASCII-Code hat $MIN\ H_d(ASCII) = 1$.
→ 8-Bit-Code $ASCII_p$ [ASCII-Code mit Paritätsbit] hat $MIN\ H_d(ASCII_p) = 2$ (Beweis folgt).
→ Also wenn 1 Bit umkippt, kann der Fehler erkannt werden [bereits bewiesen].

[1] American Standard Code for Information Interchange, eingeführt 1968

[Gerade Parität]

Beweis.

Wenn $MIN\ H_d(ASCII)) = 1 \rightarrow \exists$ solche Buchstaben r, s sodass:

$$H_d(ASCII(r), ASCII(s)) = 1$$

[ASCII(r) und ASCII(s) unterscheiden sich **in einer einzigen** Bitposition]

Wenn $r(1)$: $\#$ 1er in $ASCII(r)$

$s(1)$: $\#$ 1er in $ASCII(s)$

$\rightarrow r(1) = s(1) \pm 1$

$\rightarrow p(ASCII(r)) = \overline{p(ASCII(s))}$

$\rightarrow ASCII_p(r) \equiv ASCII(r) \circ p(ASCII(r))$ unterscheidet sich von $ASCII_p(s) \equiv ASCII(s) \circ p(ASCII(s))$ auch im Paritätsbit

$\rightarrow H_d(ASCII_p(r), ASCII_p(s)) = 2 \rightarrow MIN\ H_d(ASCII_p) = 2$ $\square$

Fehlererkennung mit Paritätsbit

Beispiel:

C_1: 7-Bit-ASCII, C_2: 8-Bit-$ASCII_p$ mit **gerader** Parität
Wenn ein 8-Bit-Wort, mit **ungerader Anzahl von 1-Bits** nach der Codierung ankommt, wird es gleich als **fehlerhaft** erkannt. Warum?

Das 7-Bit-Codewort $C_1(a)$ besitzt entweder eine

1. ungerade Anzahl an 1ern
2. gerade Anzahl an 1ern.

Dann hat $C_2(a) = C_1(a) \circ p(C_1(a))$ bei gerader Parität:

1. ungerade + 1 = **gerade** Anzahl an 1ern
2. gerade + 0 = **gerade** Anzahl an 1ern.

Fehlererkennung mit Paritätsbit

Also darf $C_2(a)$ bei richtiger Codierung von a **keine ungerade Anzahl an 1ern** haben.
$\rightarrow$ Enthält $C_2(a)$ eine ungerade Anzahl an 1ern, wurde a von C_2 falsch codiert.

Frage: Welche Anzahl von Bitfehlern (d. h. welche Anzahl umgekippter Bits) kann bei einem 7-Bit-ASCII-Codewort, das mit einer geraden Parität (even parity) codiert wurde, erkannt werden?

Fehlererkennung mit Paritätsbit

Die Situation:

$$r \text{ (7-Bit-ASCII Wort)} \quad \xrightarrow[]{\substack{\text{Codierung} \\ \text{(even parity)}}} \quad C(r) = r \circ p(r) \text{ (8-Bit-Wort)}$$

$C(r)$ hat **gerade Anzahl an 1ern**

r : ungerade Anzahl an 1ern $\rightarrow p(r) = 1 \rightarrow r \circ p(r)$ **gerade** $\#$ an 1ern
r : gerade Anzahl an 1ern $\rightarrow p(r) = 0 \rightarrow r \circ p(r)$ **gerade** $\#$ an 1ern

$\downarrow$ Übertragung
(Hier kann Umkippen von Bits auftreten)

$$[C(r)]' = [r \circ p(r)]'$$

Der Empfänger hat $[C(r)]'$ zur Verfügung.

Fehlererkennung mit Paritätsbit

- Wenn bei ihm $[C(r)]'$ mit **gerader** Anzahl von 1ern ankommt: Fehler unerkannt
- Wenn bei ihm $[C(r)]'$ mit **ungerader** Anzahl von 1ern ankommt: Fehler erkannt
- Wann kommt $[C(r)]'$ mit einer **geraden** Anzahl von 1ern an? Wenn unterwegs eine gerade Anzahl an Bitpositionen umkippen.

$$\begin{array}{ccc} C(r) = r \circ p(r) & \xrightarrow{2,4,\ldots \text{ Bits kippen}} & [C(r)]' \\ \text{gerade \# 1er} & & \text{gerade \# 1er} \\ & & \\ 1\underline{000000}1 \text{ (2 1er)} & \xrightarrow{\text{6 Bits gekippt}} & 1\underline{111111}1 \text{ (8 1er)} \\ 111010\underline{11} \text{ (6 1er)} & \xrightarrow{\text{2 Bits gekippt}} & 111010\underline{00} \text{ (4 1er)} \end{array}$$

Also: Wenn eine gerade Anzahl an Bits (2,4,6,8) kippen, hat auch das ankommende Wort wieder eine gerade Anzahl an 1er-Bits, weshalb der Fehler nicht erkannt werden kann.

Fehlererkennung mit Paritätsbit

- Wann kommt $[C(r)]'$ mit einer **ungeraden** Anzahl von 1ern an? Wenn unterwegs eine ungerade Anzahl an Bitpositionen umkippen.

$$
\begin{array}{ccc}
C(r) = r \circ p(r) & \xrightarrow{1,3,\ldots \text{ Bits kippen}} & [C(r)]' \\
\text{gerade } \# \text{ 1er} & & \text{ungerade } \# \text{ 1er} \\
1\underline{00}0\underline{000}1 \ (2 \text{ 1er}) & \xrightarrow{5 \text{ Bits gekippt}} & 11101111 \ (7 \text{ 1er}) \\
11101011 \ (6 \text{ 1er}) & \xrightarrow{1 \text{ Bit gekippt}} & 01101011 \ (5 \text{ 1er})
\end{array}
$$

Also: Wenn eine <u>ungerade Anzahl</u> an Bits (1,3,5,7) kippen, hat das ankommende Wort eine ungerade Anzahl an 1er-Bits, weshalb der Fehler <u>erkannt</u> werden kann.

[Gray-Code]

Angenommen: A: Alphabet (also geordnet)

GC: $A \longrightarrow GC(A)$

Definition

[Gray-Code] Die Gray-Codierung (GC) ist so definiert, dass

$$H_d(GC(a_1), GC(a_2)) = 1, \forall a_1, a_2 \in A$$

wobei a_1, a_2 **aufeinanderfolgende** Zeichen in A sind.

[Gray-Code]

Beispiel: 4-Bit-Gray-Code zur Codierung der Dezimalziffern

$A = \{1, 2, 3, \ldots, 9\}$

$GC:$	A	$\longrightarrow$	$GC(A)$
	0		0000
	1		0001
	2		0011
	3		0010
	4		0110
	5		0111
	6		0101
	7		0100
	8		1100
	9		1000

Offensichtlich gilt für alle benachbarten Ziffern in A, dass ihre Gray-Code-Bilder eine Hammingdistanz von 1 aufweisen.

[Gray-Code]

$$\left.\begin{array}{l} a_1 = 6 \to GC(6) = 010\underline{1} \\ a_2 = 7 \to GC(7) = 010\underline{0} \end{array}\right\} \to H_d(GC(6), GC(7)) = 1$$

Besser: GC ist ein zyklischer Gray-Code, da der Code **des ersten** und des **letzten** Zeichens der Ordnung sich ebenfalls nur an einer Stelle unterscheiden.

Analytische Methoden zur Erzeugung von Gray-Codes

[Transitionssequenz](Methode zur Erzeugung von Binary Reflected Gray Codes)

Definition

[Transitionssequenz] $T(n+1) = T(n) \quad n+1 \quad T(n)$
mit $T(1) = 1$.

Diese Sequenz gibt jene Stellen eines binären Codeworts der Länge n+1 an, die <u>hintereinander verändert werden müssen</u>, um einen (zyklischen) Gray-Code zu erhalten.

[Transitionssequenz]

Beispiel:

$n+1=3 \qquad T(2+1) = \underbrace{\mathrm{T}(2)}_{\downarrow} \quad \underbrace{3}_{\downarrow} \quad \underbrace{\mathrm{T}(2)}_{\downarrow}$

$= \underline{1}, \underline{2}, 1 \quad 3 \quad 1, 2, 1$

Binäres Codewort: $a_1 a_2 a_3$ [Länge $n+1$]

Veränderung nach der T-Sequenz:

$\underline{1}$: $a_1 a_2 \bar{a}_3$
$\underline{2}$: $a_1 \bar{a}_2 \bar{a}_3$
1: $a_1 \bar{a}_2 a_3$
3: $\bar{a}_1 \bar{a}_2 a_3$
1: $\bar{a}_1 \bar{a}_2 \bar{a}_3$
2: $\bar{a}_1 a_2 \bar{a}_3$
1: $\bar{a}_1 a_2 a_3$

Offensichtlich: Dieser Gray-Code ist zyklisch, da $H_d(\bar{a}_1 a_2 a_3,\ a_1 a_2 a_3) = 1$.

Fehlererkennung mit Gray-Code

Fehlererkennung mit Gray-Code
Zwei hintereinander kommende Codewörter müssen $H_d = 1$ haben.
Wenn nicht: Codefehler

k-aus-n-Codes

Definition

[k-aus-n-Code] Code, bei dem ein Codewort der Länge <u>n</u> genau <u>k</u> 1-Bits aufweist.

Beispiel: 1-aus-10-Code für Dezimalziffern

Ziffer	1-aus-10-Code
0	0000000001
1	0000000010
2	0000000100
3	0000001000
4	0000010000
5	0000100000
6	0001000000
7	0010000000
8	0100000000
9	1000000000

$H_d = 2$: immer

0	1
1	0

k-aus-n-Codes

Durch die Erhöhung der Redundanz [also Dehnung der Wortlänge (Dezimalziffern könnte man mit 4-Bit-Code repräsentieren - hier haben wir 10 Bits!)] ist die Fehlererkennung erleichtert.

Man kann viele Fehler erkennen, da $\#1er = 1$.
(Siehe Folie 44: Da $MIN\ H_d = MAX\ H_d = 2$, sind k-Bit-Fehler für $k = 1$ ($k < MIN\ H_d$) und für $k = 3, \ldots, n$ ($k > MAX\ H_d$) erkennbar.)

Aber wenn 2 Fehler auftreten, kann man unter Umständen keinen Fehler erkennen.

z. B.: Statt $\underline{0}00\underline{1}$ kommt $\underline{1}00\underline{0}$ an
$\rightarrow$ Fehler in zwei Bits, trotzdem $\#1 = 1$
$\rightarrow$ Fehler kann nicht erkannt werden.

Code variabler Länge

Möglichkeit, die Codewortlänge an die Häufigkeit des Auftretens eines zu codierenden Zeichens anzupassen.

z. B. Morsecode E: ·

Das Ende und der Anfang jedes Wortes muss getrennt werden.

Beispiel:
– · – ist als K oder als TET zu erkennen [T: – ; E: ·]

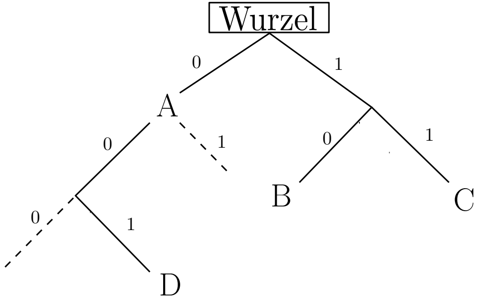
Morse-Codierung benötigt ein drittes Signal [kleine Pause zwischen den Buchstaben].

Codebäume

Codebäume
Geeignet zur Decodierung serieller Codes variabler Länge

Beispiel: ein Code variabler Länge

Zeichen	Code
A	0
B	10
C	11
D	001



Codebäume

Man beginnt an der Wurzel. Pfad im Baum folgen (laut empfangenem Zeichen). Erreicht man einen Knoten (mit dem decodierten Zeichen), notiert man dieses und es geht weiter los von der Wurzel.
[Wir werden D nie decodieren, weil wir dazu nie kommen werden, wir werden A erkennen.]

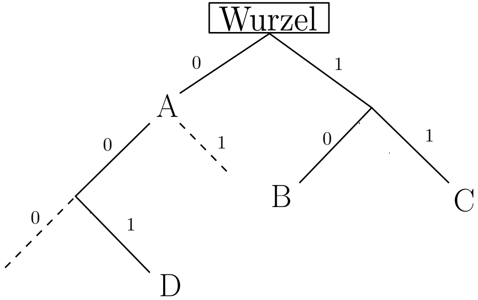
Eindeutigkeit garantiert [ohne Trennzeichen!], wenn decodierte Zeichen nur an den Blättern [Endknoten] des Baums stehen.

→ Fano-Bedingung [andere Formulierung für diese Erkenntnis]

Codebäume

Beispiel:

Zeichen	Code
A	0
B	10
C	11
D	001

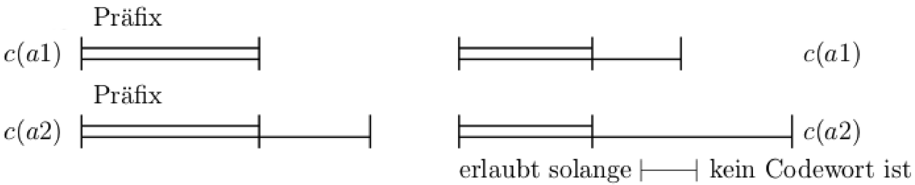


Code (A | C | D | A) = 0 | 11 | 001 | 0

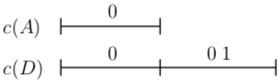
Baum Decode $(0 \underset{W}{\downarrow} 11 \underset{W}{\downarrow} 0 \underset{W}{\downarrow} 0 \underset{W}{\downarrow} 10) = \text{ACAAB} \neq \text{ACDA}$

Fano-Bedingung

Kein Codewort darf Anfang eines anderen Codeworts sein. [Kein Codewort darf Präfix eines anderen Codeworts sein.]
Also verboten für die Eindeutigkeit:



Unser Beispiel:



$\rightarrow$ Verstoß gegen Fano-Bedingung $\rightarrow$ D nicht decodierbar

Fano-Bedingung

Fano-Bedingung ist hinreichend für die Umkehrbarkeit der Codierung (d. h. wenn die Fano-Bedingung erfüllt ist, dann ist der Code eindeutig decodierbar).
Aber nicht notwendig (d. h. wenn der Code eindeutig decodierbar ist, muss die Fano-Bedingung nicht erfüllt sein).

Beispiel: hinreichend – nicht notwendig

hinreichend: $\underbrace{\text{Wenn es regnet,}}_{R}$ ist $\underbrace{\text{die Straße nass.}}_{S}$ $(R \to S)$

aber
nicht notwendig: Wenn die Straße nass ist, muss es nicht geregnet haben (z. B. nass durch die Straßenreinigung).

$\rightarrow$ Es gibt etwas (= Reinigung), das die Straße nass macht.
$\rightarrow$ Wir können einen Code finden, der eindeutig decodierbar ist, aber die Fano-Bedingung verletzt.

Fano-Bedingung (nicht notwendig für die Decodierbarkeit)

Beispiel:

Zeichen	Code
A	10
B	100
C	1000

1 ist als Steuerzeichen (= Beginn) eines Zeichens definiert
$\rightarrow$ Code ist eindeutig decodierbar.

Informationsgehalt

Allgemein

Bislang haben wir uns mit einer **qualitativen** Beschreibung von Zeichen und Wörtern beschäftigt,
d. h. mit der Repräsentation von Informationen.

- Wie kann man Informationen **quantifizieren**, messen?
- Wie groß ist der Informationsgehalt (einer Nachricht)?

Allgemein

Beispiel:

[Nachricht]

a) Die österreichische Nationalmannschaft wird Fußball-Weltmeister.

b) Der Tag hat 24 Stunden.

c) Nach Dienstschluss sind die Büros im Informatik-Gebäude in Itzling dunkel.

Zentraler Begriff bei der Bewertung des Informationsgehalts ist die **Wahrscheinlichkeit**, mit der die Aussage eintritt.

Aussage a) kleine Wahrscheinlichkeit.
b) fast schon Axiom (fast 100% wahrscheinlich)
c) unterschiedlich zu a) b), hat mit <u>Bedingungen</u> zu tun.
Hängt davon ab, <u>wann</u> die Fenster beobachtet werden.

Allgemein

Beispiel:

[Nachricht]

a) Die österreichische Nationalmannschaft wird Fußball-Weltmeister.
b) Der Tag hat 24 Stunden.
c) Nach Dienstschluss sind die Büros im Informatik-Gebäude in Itzling dunkel.

Der Informationsgehalt ist umso höher, je kleiner die Wahrscheinlichkeit ihres Eintretens ist,

d. h. Informationsgehalt von a) hoch
b) fast null.

Formalisierung

Angenommen: [Nachricht, Aussage] A, ihre Wahrscheinlichkeit: p_A, ihr Informationsgehalt: I_A
Wir stellen folgende Bedingungen an den Informationsgehalt:

$$I_A = f(p_A) \tag{1}$$

[Der Informationsgehalt I_A einer Nachricht A ist eine Funktion der Wahrscheinlichkeit von A.]

$$I_A \geq 0 \qquad \text{mit } 0 \leq p_A \leq 1 \tag{2}$$

[Informationsgehalt muss ≥ 0 sein.]

$$\lim_{p_A \to 1} I_A = 0 \tag{3}$$

[Der Grenzwert des Informationsgehalts ist 0, wenn die Wahrscheinlichkeit nach 1 geht.] (Aussage b))

$$I_A > I_B \iff p_A < p_B \tag{4}$$

[Wenn der Informationsgehalt von A größer ist als von B, dann verhalten sich die Wahrscheinlichkeiten umgekehrt.]

Unabhängigkeit von Nachrichten

Viele Funktionen $I_A = f(p_A)$ erfüllen (1)–(4).

Frage: Wie wird der Informationsgehalt von 2 oder mehr Nachrichten A, B, ... gemessen?

Begriff: Unabhängigkeit von Nachrichten (z. B. zwei unabhängige Nachrichten):
A: Heute ist es schön. B: Morgen esse ich Fisch.

Satz c): Abhängige Nachrichten.
[Abhängig vom Tag und von der Uhrzeit der Aussage,
sonst ist die Nachricht nutzlos.]

Wir können den Informationsgehalt der beiden (**ab**hängigen) Nachrichten nicht als Summe der einzelnen Informationsgehalte angeben, aber für die Wahrscheinlichkeit des gleichzeitigen Eintretens zweier **un**abhängiger Nachrichten (Ereignisse) A, B gilt:

$$p(AB) = p_A p_B.$$

Unabhängigkeit von Nachrichten

Beispiel:

(Wahrscheinlichkeit von 2 unabhängigen Ereignissen):

Münze	Kopf/Zahl
	$p_K = 1/2$ $\quad$ $p_Z = 1/2$
$A = K$ $\quad$ $B = K$	(Wenn Kopf dann wieder Kopf)
$p(KK) = p_K \cdot p_K = 1/4$	
[5-mal Kopf: $(1/2)^5$]	

Unabhängigkeit von Nachrichten

Und für 2 unabhängige Nachrichten A, B verlangen wir noch

$$I_{AB} = I_A + I_B \, . \tag{5}$$

Genau eine Funktion $I_A = f(p_A)$ erfüllt alle Forderungen (1)–(5):

$$I_A = \log_b \frac{1}{p_A} = -\log_b p_A \, .$$

Logarithmus zur Basis b

b = 10	(lg a):	Zehner-Logarithmus (dekadischer Logarithmus)
b = e	(ln a) :	natürlicher Logarithmus (Logarithmus naturalis), wobei e die Eulersche Zahl ist
b = 2	(ld a) :	binärer Logarithmus (Logarithmus dualis)

Unsere Wahl: $b = 2$ ($\log_2 \equiv \mathrm{ld} \equiv \log$ (Logarithmus dualis)).

Einheit für den Informationsgehalt

Die Einheit für den Informationsgehalt: **Bit**.

Beispiel:

Warum $\log(\equiv \log_2)$ für Informationsgehalt?

Angenommen, dass beide Zeichen aus dem Zeichenvorrat $B = \{\underline{0},\underline{1}\}$ mit gleicher Wahrscheinlichkeit auftreten,
d. h.

$$p_{\underline{0}} = p_{\underline{1}} = 1/2 \qquad\qquad (p_{\underline{0}} + p_{\underline{1}} = 1)$$

dann ist der Informationsgehalt dieser beiden Zeichen gleich **1 Bit**.

$$I_{\underline{0}} = I_{\underline{1}} = \log \frac{1}{\frac{1}{2}} = \log 2^1 = \mathbf{1}\ [\text{Bit}]$$

Mittlerer Informationsgehalt

Bemerkung: Allgemein:
Wenn für einen Zeichenvorrat $ZV = \{Z1, Z2\}$ $p_{Z1} \neq p_{Z2}$ gilt, dann ist der Informationsgehalt $I_{Z1} \neq I_{Z2} \neq 1$ Bit.

Wir betrachten eine „sinnvolle Quelle“, d. h. eine Nachrichtenquelle, die viele verschiedene Nachrichten (aus **einem** Zeichen/Wörter-Vorrat erzeugt) sendet.

Zur Charakterisierung einer solchen Quelle wird der **mittlere Informationsgehalt** verwendet.

[Stochastische (Shannonsche) Nachrichtenquelle]

Definition

[Stochastische (Shannonsche) Nachrichtenquelle]: Eine Nachrichtenquelle, bei der zu jedem Zeitpunkt die Wahrscheinlichkeit des nächsten zu sendenden Zeichens gleich der mittleren Häufigkeit dieses Zeichens ist.

Mittlere Häufigkeit h_i eines Zeichens z_i: Errechnet sich aus Gesamtzahl N aller gesendeten Zeichen und der Zahl n_i, die angibt, wie oft das Zeichen z_i gesendet wurde:

$$h_i = \frac{n_i}{N} \ .$$

Informationsgehalt einer Nachricht von N Zeichen

Aus der Definition der stochastischen Nachrichtenquelle $\rightarrow$

- Die Zeichen einer stochastischen Nachrichtenquelle sind statistisch unabhängig. Angenommen, dass diese Quelle m verschiedene Zeichen senden kann, dann muss für die Wahrscheinlichkeiten dieser m Zeichen gelten: $\sum_{i=1}^{m} p_i = 1$.
- Der Informationsgehalt einer Nachricht mit insgesamt N Zeichen, die durch eine solche Nachrichtenquelle erzeugt wird (Nachricht Länge N), ist

$$I_m(N) = Np_1 I_1 + Np_2 I_2 + \ldots + Np_{\underline{m}} I_{\underline{m}} = \sum_{i=1}^{\underline{m}} Np_i I_i = N \sum_{i=1}^{\underline{m}} p_i I_i \,.$$

Informationsgehalt einer Nachricht von N Zeichen

Informationsgehalt einer Nachricht von N Zeichen
[die aus m Zeichen $z_1, z_2 \ldots, z_m$ ausgewählt werden]:

$$I_m(N) = n_1 I_1 + n_2 I_2 + \ldots + n_m I_m$$

n_i: # von Zeichen z_i in N, wobei $i = 1, \ldots, m$
$\rightarrow n_1 + n_2 + \ldots + n_m = N$
I_i: Informationsgehalt von Zeichen z_i, wobei $i = 1, \ldots, m$

$$I_m(N) = \frac{N}{N} n_1 I_1 + \frac{N}{N} n_2 I_2 + \ldots + \frac{N}{N} n_m I_m$$

Informationsgehalt einer Nachricht von N Zeichen

- Laut Definition ist $\frac{n_i}{N} = h_i$ (mittlere Häufigkeit des Zeichens z_i) $\rightarrow$

$$I_m(N) = Nh_1 I_1 + Nh_2 I_2 + \ldots + Nh_m I_m$$

- Laut Definition der stochastischen Nachrichtenquelle ist die Wahrscheinlichkeit (p_i) des nächsten zu sendenden Zeichens (z_i) gleich der mittleren Häufigkeit des Zeichens (h_i) $\rightarrow$

$$I_m(N) = Np_1 I_1 + Np_2 I_2 + \ldots + Np_m I_m = N \sum_{i=1}^{m} p_i I_i \,.$$

[Entropie]

Definition

[Entropie]: Der mittlere Informationsgehalt der Quelle heißt Entropie der Quelle und ist definiert als

$$H = \sum_{i=1}^{m} p_i I_i = \sum_{i=1}^{m} p_i log \frac{1}{p_i} = - \sum_{i=1}^{m} p_i log p_i$$

[H: Informationsgehalt der Nachrichtenlänge N=1, also Informationsgehalt pro Zeichen]

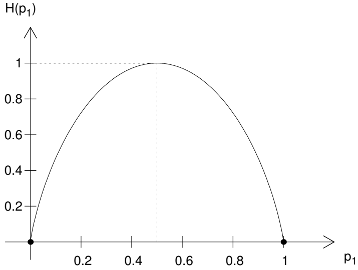
Bemerkung: Für eine Nachricht mit N Zeichen gilt (siehe vorige Folie):
weil $H = \sum_{i=1}^{m} p_i I_i \rightarrow H = \frac{I_m(N)}{N}$.

Entropie einer binären (stochastischen) Nachrichtenquelle

Beispiel:

$$H = p_1 log \frac{1}{p_1} + p_2 log \frac{1}{p_2}$$

Weil $p_1 = 1 - p_2$ gelten muss, ist $H = H(p_1)$ [also als Funktion einer Veränderlichen darstellbar]



Also: Die höchste Entropie erzielt man, wenn beide Zeichen mit gleicher Wahrscheinlichkeit [d. h. $p_1 = p_2 = 1/2$] auftreten.

Entropie einer binären (stochastischen) Nachrichtenquelle

- Maximum von $H = 1$ Bit [$p_1 = 0.5$]
  [$H_{MAX} = \frac{1}{2} \cdot 1$ Bit $+ \frac{1}{2} \cdot 1$ Bit $= 1$ Bit]
- Hat eines der zwei Zeichen eine niedrige Wahrscheinlichkeit [und damit das andere eine hohe], nimmt die Entropie ab.
  [Für $p_1 = 0$ oder $p_1 = 1 \rightarrow H = 0$.]
- Selbst wenn die Zeichen-Probabilität um 0.5 wandert zu $p_1 = 0.3$, ist die Entropie noch ~ 0.88 Bit $\rightarrow$ ein flaches Maximum.

Optimalcodierung

[Mittlere Codewortlänge eines Binärcodes]

Mit Hilfe der Entropie beschreiben wir quantitativ die Effizienz einer Codierung

Definition

[Mittlere Codewortlänge eines Binärcodes]

$$L = \sum_{\underline{i}=1}^{m} p_i \ell_i \,. \qquad \text{[in Bits]}$$

Wobei

m:	die Anzahl der Codewörter (für die Codierung von m Zeichen)
ℓ_i:	die Länge [Anzahl der Bits] des Codeworts $\underline{i}$
p_i:	dessen Wahrscheinlichkeit.

Entropie / Mittlere Codewortlänge

Wenn angenommen wird, dass jedes Bit der Codewörter $i = 1, \ldots, m$ den Informationsgehalt 1 Bit hat [Maximalfall]:

$\rightarrow$ Der Informationsgehalt I_i des Codeworts i der Länge ℓ_i ist ℓ_i und daher

$$I_i = \ell_i \quad (*)$$

$$\rightarrow H = \sum_{i=1}^{m} p_i I_i = \sum_{i=1}^{m} p_i \ell_i = L\,.$$

Die mittlere Codewortlänge $\underline{L}$ entspricht in diesem Fall $\underline{H}$, der Entropie der Quelle.

Der Informationsgehalt I_i des Codeworts für das Zeichen z_i $(i = 1, \ldots, m)$ beträgt $\log \frac{1}{p_i}$ $\rightarrow$ die Länge des entsprechenden Codeworts ist laut $(*)$ $\ell_i = \log \frac{1}{p_i}$.

Also ist in diesem Maximalfall die Wortlänge codierter Wörter umso kürzer, je höher die Auftrittswahrscheinlichkeit des codierten Zeichens ist.

Entropie / Mittlere Codewortlänge

Umgekehrt:
Bei gegebener Entropie der Quelle versucht man, die Codierung so zu wählen, dass die mittlere Wortlänge möglichst nahe an die Entropie der Quelle herankommt.

Dass dies möglich ist, besagt das Shannonsche Codierungstheorem.

Shannonsches Codierungstheorem

Theorem (Shannonsches Codierungstheorem)

Für beliebige binäre Codierungen gilt $H \leq L$. Jede Nachrichtenquelle kann durch binäre Codierung so codiert werden, dass der positive Wert $L - H$ beliebig klein wird.

$\rightarrow$ Dieser Optimalcode ist durch die Bedingung $L = H$ gegeben.

[Redundanz, relative Redundanz, Codeeffizienz]

Definition

[Redundanz, relative Redundanz, Codeeffizienz] Die Redundanz R eines Codes ist

$$R = L - H.$$

Die relative Redundanz r und Codeeffizienz η_c eines Codes ist

$$r = \frac{R}{L} = \frac{L - H}{L} = 1 - \frac{H}{L} = 1 - \eta_c\,.$$

Wenn $H = L$ [Optimalcode] $\rightarrow r = 0$ und Codeeffizienz $\eta_c = 1\,.$

Beispiel: Redundanz der deutschen Sprache

Welche Redundanz weist die deutsche Sprache auf?

Annahme: 26 Zeichen [ohne Umlaute, ß und Zwischenraum], die gleichwahrscheinlich auftreten.

Dann erhält man die maximale Entropie

$$
\begin{aligned}
H_0 &= \sum_{i=1}^{m=26} p_i \log \frac{1}{p_i} = \sum_{i=1}^{m=26} \frac{1}{26} \log \frac{1}{\frac{1}{26}} \\
&= \frac{1}{26} 26 \log 26 = \log 26 \approx 4.7 \text{Bit}.
\end{aligned}
$$

[also Informationsgehalt pro Zeichen]

Beispiel: Redundanz der deutschen Sprache (Fortsetzung)

Unter Berücksichtigung statistischer Abhängigkeiten innerhalb von Silben, Wörtern und Sätzen [Küpfmüller, 1954] gelangt man zu einer (realen) Entropie $H_\infty \approx 1.6$ Bit.
Die relative Redundanz ist dann

$$r = \frac{H_0 - H_\infty}{H_0} = 1 - \frac{H_\infty}{H_0} = 0.6595\,.$$

(Diese Definition ist eine äquivalente Definition zu $r = 1 - \frac{H}{L}$, weil H_0 [maximale Entropie]$= L$ und $H = H_\infty$ [reale Entropie].)

Daher ist in der deutschen Sprache also 2/3 der transportierten Information „überflüssig“.

Redundanz

Redundanz: Informationsüberfluss (überflüssige Information):
Man bräuchte nicht so viel zu übertragen, um die Information zu erhalten.

- Aus dem Codierungstheorem folgt, dass die Redundanz eines Codes beliebig klein werden kann.

  Gibt es Methoden für die Konstruktion eines solchen Codes?
  - Shannon-Fano-Algorithmus
  - Huffman-Codierung [Huffman, 1952]

Huffman-Codierung

Grundidee: Für die optimale Codierung soll gelten: $H = L$

$$\rightarrow \sum_{i=1}^{m} p_i \log \frac{1}{p_i} = \sum_{i=1}^{m} p_i \ell_i \rightarrow \ell_i = \log \frac{1}{p_i}$$

$\rightarrow$ Die Wortlänge eines codierten Wortes sollte indirekt proportional zu der Wahrscheinlichkeit des Auftretens des Wortes sein.

[D. h. für die am häufigsten auftretenden Wörter gibt es die kürzeste Codierung und umgekehrt.]

Huffman-Codierung

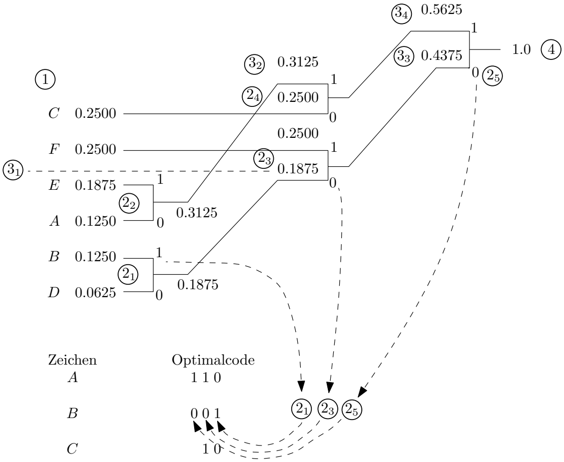
Zur Illustration des Verfahrens: Die Wahrscheinlichkeiten des Auftretens von Zeichen einer stochastischen Nachrichtenquelle mit dem Zeichenvorrat $Z = \{A, B, C, D, E, F\}$

Zeichen	Wahrscheinlichkeit
A	0.1250
B	0.1250
C	0.2500
D	0.0625
E	0.1875
F	0.2500

Das Verfahren [Huffman-Codierung]

1. Man ordnet die Symbole [Zeichen] nach fallenden Wahrscheinlichkeiten.
2. Von unten beginnend ordnet man dem Symbol mit der kleinsten Wahrscheinlichkeiten das Bit 0, dem nächstgrößeren das Bit 1 zu.
3. Man summiert die Wahrscheinlichkeiten der beiden Symbole und ordnet sie quasi als neues Einzelsymbol entsprechend der Summenwahrscheinlichkeit in die vertikale Symbolfolge ein.
4. Wenn die Summenwahrscheinlichkeit $p < 1.0$ ist, dann fährt man mit 2. fort; ansonsten ist der Code fertig konstruiert.
5. Die einzelnen Codewörter liest man vom horizontalen Ende des entstandenen Pfades zu den jeweiligen Symbolen ab, indem man alle dabei auftretenden Bits der Reihe nach notiert.

Das Verfahren [Huffman-Codierung]



Zeichen	Optimalcode
A	1 1 0
B	0 0 1
C	1 0
D	0 0 0
E	1 1 1
F	0 1

Das Verfahren [Huffman-Codierung]

- Die Zeichen mit der höchsten Auftrittswahrscheinlichkeit haben den kürzesten Code [C, F: Wahrscheinlichkeit = 0.2500].
- Die Konstruktion des Huffman-Codes liefert immer einen Code, der die Fano-Bedingung erfüllt.
- Die Konstruktion des Codes ist nicht eindeutig. [Wir hätten einen anderen Huffman-Code erhalten, wenn wir die Summenwahrscheinlichkeit in 3_1, die gleich der Wahrscheinlichkeit von E ist, nicht über E platzieren würden.]
- Wenn wir die Redundanz des entstehenden Codes berechnen, werden wir sehen, dass diese ungleich 0 ist.

Bedingung [Für einen redundanzfreien Code]

Für einen redundanzfreien Code muss für die Auftrittswahrscheinlichkeiten p_i eines Zeichens i gelten, $p_i = \frac{1}{2^{n_i}}$ mit $n_i \in \mathbb{N}$.

Laut Theorem ist eine Verbesserung möglich [da Redundanz $\neq 0$].

$\rightarrow$ Methode der Codeerweiterung

Hierbei werden nicht die einzelnen Zeichen der Quelle codiert, sondern zwei oder mehrere zugleich.
Bei statistischer Unabhängigkeit der Zeichen ergibt sich dann für ein Wort aus zwei Zeichen die Auftrittswahrscheinlichkeit $p_{xy} = p_x \cdot p_y$, was zu einer feineren Aufspaltung der Wahrscheinlichkeiten führt und eine effizientere Huffman-Codierung erlaubt.

[Diese Vorgangsweise ist auch die Basis eines konstruktiven Beweises des Shannonschen Codierungstheorems.]

Methode der Codeerweiterung

Beispiel:

Sechs Zeichen aus vorigem Beispiel erzeugen 6^2 Zeichenpaare
$[AA, AB, \ldots, AF, BA, \ldots, BF, \ldots, FA, \ldots, FF]$

z. B. $p_{AA} = 0.125 \times 0.125 = 0.015625$

$\rightarrow$ feinere Aufspaltung, aber ein größerer Rechenaufwand.

Verbesserung[klein] $\quad r = 0.09$ gegenüber $r = 0.1$.

Anwendung des modifizierten Huffman-Codes

Beispiel: Fax

Warum bei Fax-Übertragung?

Weil das Fax als Bild übertragen wird.
In der Bildverarbeitung wird oft Huffman-Code eingesetzt.
90% der Punkte sind weiß $\rightarrow$ kurzes Codewort
$\rightarrow$ Nur ein paar Bits für ganze weiße Zeile.

Nachrichtenkanal und Leitungscodierung

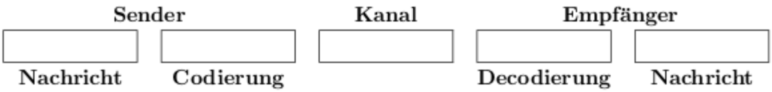
Nachrichtenkanal und Leitungscodierung

Bisher:
Information und deren Codierung „informationstheoretisch“ behandelt.

Ziel:
Einige wesentliche technische Aspekte der Nachrichtenübertragung und deren Codierung zu behandeln.

Die Übertragung von Signalen (Nachrichten)

[Schema des Nachrichtenübertragungsprinzips]



Was ist ein Kanal [im nachrichtentechnischen Sinne]?

z. B. Glasfaserleitung

z. B. Kupferkabel

z. B. Frequenzkanäle für elektromagnetische Wellen.

Qualitätsproblem jedes Nachrichtensystems:

Rauschen (Noise)
Rauschen zerstört die Information [z. B. Lautsprecher].

Allgemein [Rauschen]
Jede Störungsquelle, die ein Nachrichtensignal verändert [und damit Teile der übertragenen Information zerstört]. Es gibt Rauschquellen nicht nur im Kanal, sondern auch in Sender und Empfänger.
Modellhaft wird jegliches Rauschen dem Kanal zugeordnet.

→ Zur genaueren Formulierung des Codierungstheorems.

Fundamentalsatz der Codierung

Theorem (Fundamentalsatz der Codierung)

Für eine Nachrichtenquelle mit der Entropierate H' und einem Übertragungssystem mit der Kanalkapazität C gibt es für den Fall $H' \leq C$ eine Codierung, die die Übertragung der Nachricht über den rauschbehafteten Kanal mit beliebig kleiner Fehlerrate erlaubt.

Dies ist eine andere Formulierung des Shannonschen Codierungstheorems von Folie 91 aus technischer Sicht.

[Entropierate]

Definition

[Entropierate] H': Entropie pro Zeiteinheit [Bit/s]

$\rightarrow$ Kanalkapazität $\underline{C}$ hat die Einheit [Bit/s]

Aus dem Theorem:

Die Kanalkapazität C stellt die über diesen Kanal maximal übertragbare Entropierate dar.

- Kanalkapazität wird durch Rauschen verringert.
  Extrem: kaputte Leitung

$$C = 0 \qquad H \leq C \text{ [Theorem]} \rightarrow H = 0$$

Aus dem Theorem

Wie findet man eine geeignete Codierung?
Im Zusammenhang mit der Übertragung von Signalen über einen Kanal [oder Leitung] spricht man von **Leitungscodierung** der Signale.

Leitungscodierung

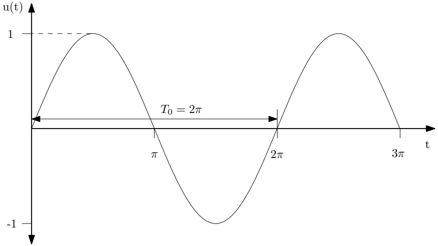
Anpassung des codierten Signals an den Übertragungskanal.

Also: Die (meist schon) codierte Nachricht muss in der Regel einer weiteren Codierung [d. h. Leitungscodierung] unterworfen werden.

Frequenz f	Anzahl von Takten pro Zeiteinheit [Einheit für Frequenz: $Hz = s^{-1}$]
Periodendauer $T_0 = \frac{1}{f}$	Zeit für einen Takt

Beispiel:

- $f = 1kHz \rightarrow T_0 = \frac{1}{10^3 Hz} = 10^{-3}s = 1ms$ [Millisekunde]
- Prozessoren $f = 1GHz = 10^9 Hz$
- Periodendauer [Takt] $T_0 = 10^{-9}s = 1ns$ [Nanosekunde]
- Sinuskurve



$T_0 = 2\pi,\ f = \frac{1}{2\pi}$ Also: Diese Kurve hat nur eine Frequenz.

[Bandbreite (bandwidth)]

Definition

[Bandbreite (bandwidth)] Die **Bandbreite** eines Signals ist jener **Frequenzbereich**, den das Signal benötigt, um (näherungsweise) unverformt beim Empfänger anzukommen (Rauschfreiheit vorausgesetzt).

Es gilt für die Bandbreite:

$$\Delta B \approx \frac{1}{T_0}$$

also je kleiner Periodendauer die T_0 ist, desto höher die Frequenz ($f = \frac{1}{T_0}$)
[und die Übertragungsgeschwindigkeit ist auch höher, dabei weil mehr Bits (= mehr Information) übertragen werden]
und daher desto größer ist die benötigte Bandbreite ($\Delta B = \frac{1}{T_0}$), die der Kanal zur Verfügung stellen muss.

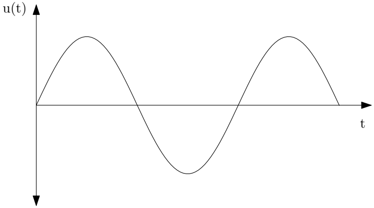
Gleichstromfreiheit

[wichtige] Bedingung an Leitungscode für elektrische Signale

- Folge, die übertragen wird, soll keinen Gleichanteil haben.

Beispiele:

- Der Strom aus der Steckdose ist ein Wechselstrom.

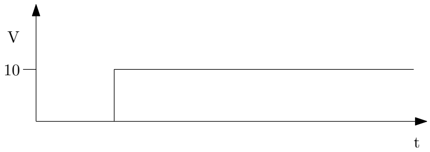


also: Der Mittelwert $\bar{u}$ von u ist $= 0$.
Wechselstrom ist gleichstromfrei [erfüllt also die Bedingung].

Gleichstromfreiheit

Beispiele:

- Hier ist der Mittelwert nicht mehr 0
  [also ist die Gleichstromfreiheit verletzt].



Einfache Codierung

Codierung mit Rechtecksignal:
Codierung eines Binärcodes mit zwei Spannungszuständen, dem Bit

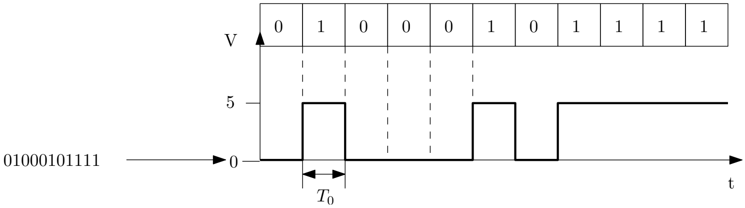
- 0 entspricht die Spannung von 0 Volt
- 1 entspricht die Spannung von 5 Volt.

[ein fester Pegel während eines Bitintervalls]

Einfache Codierung

Beispiel:

Für ein codiertes Wort 01000101111 erhalten wir folgendes Leitungssignal [Rechtecksignal]



- Der Gleichanteil der Spannung ist ungleich 0
  [Verletzung der Gleichstromfreiheit].
- Schwäche bei der Synchronisation
  Wenn z. B. 5 0er und 5 1er hintereinander folgen, ist es schwierig, die Frequenz, mit der die Bits übertragen werden, zu regenerieren.
  $\rightarrow$ Es ist manchmal schwierig, die Länge des codierten Signals festzustellen
  [z. B. kann es sein, dass anstatt 5 1ern 6 1er decodiert werden].
  Wie kann man den Mittelwert von 0 erreichen?
  $\rightarrow$ Bipolare Codierung

Manchester-Code

Die beiden Binärzustände werden auf zwei Spannungspegel abgebildet:
Das Bit **0** wird so codiert, dass in der ersten Hälfte des Bitintervalls [Periode] der negative Spannungspegel und in der zweiten Hälfte der positive Spannungspegel verwendet wird.
[Beim Bit **1** ist es gerade umgekehrt] (dies entspricht technisch einer Phasenverschiebung um 180°)

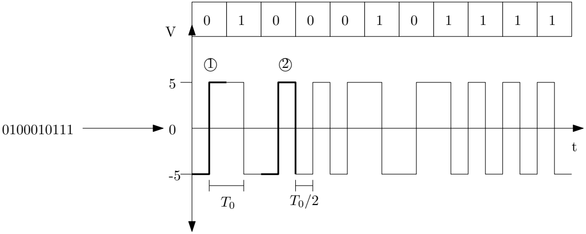
Vorteile

- Der mittlere Wert der Spannung ist gleich 0.
- Die Taktregeneration ist einfach.

Manchester-Code

Beispiel

[Das gleiche Binärwort wie im vorigen Beispiel]



Es gibt mindestens einen Signalwechsel pro Bitintervall und höchstens 2 Signalwechsel.

① **1** Signalwechsel pro Intervall: Frequenz f

② **2** Signalwechsel pro Intervall: Frequenz $2f$

Manchester-Code

$\rightarrow$ Es gibt nur zwei Zeiten zwischen Taktflanken (Signalwechseln): die Periodendauer des Bitintervalls T_0 oder $\frac{T_0}{2}$.

$\rightarrow$ Wenn T_0 die Periodendauer des Bitintervalls bei Frequenz f ist, dann ist $\frac{T_0}{2} = \frac{1}{2f} = \frac{1}{2} T_0$ die Periodendauer des Bitintervalls bei Frequenz $2f$.

$\rightarrow$ Die Taktregeneration ist **einfach**.

Nachteil beim Manchester-Code:
Die benötigte Bandbreite ist <u>doppelt so groß</u> wie bei einem Rechtecksignal konstanter Periodendauer T_0 (einfache Codierung).
(Weil hier Bit-Übertragung mit zwei Periodendauern T_0 und $\frac{T_0}{2}$.)

$\rightarrow \Delta B = \frac{1}{T_0}$ (einfache Codierung)

$\rightarrow \Delta B' = \frac{1}{\frac{T_0}{2}} = 2 \frac{1}{T_0} = \underline{2\,\Delta B}$ (Manchester-Code)