Struktur

Teil 1: Mathematische Grundlagen zu Zahlen und Algebra

1. **Mathematische Grundlagen: Das Stellenwertsystem**
2. **Binärzahlen: Praktisch angeschaut**
3. **Eine (nicht) mathematische Einführung in die Gruppentheorie**
   - Was ist Gruppe, Ring, Körper?
   - Modulo Rechnung
   - Interpretation eines Datenworts als
     - Tupel, Zahl, Vektor, Polynom
   - Was ist eine zyklische Gruppe
   - Bool`sche Alegra
4. **Erste Schritte in den Wahrscheinlichkeitsbegriff und die Kombinatorik**

Teil 2: Codierungs- und Informationstheorie Grundlagen

1. **Einführung in die Informationstheorie**
2. **Kanalmodell**
3. **Kanalcodierung**
   - Blockcodes
   - Faltungscodes
4. **Quellencodierung**
   - Komprimierung
   - Verschlüsselungsverfahren

OST

Lernziele

- Sie haben das Prinzip des Stellenwertsystems verstanden und können es erklären.
- Sie können die Menge der Erlaubten Ziffern und Wertigkeit unterscheiden
- Sie Verstehen das Dualsystem, Oktalsystem, Dezimalsystem Hexadezimalsystem und
- können Zahlen von einem Zahlensystem ins andere transponieren.
- Sie kennen die bekanntesten Zahlensysteme und können die Unterschiede erläutern.
- Sie verstehen, wie man die Subtraktion durch eine Addition realisieren kann
- Sie können das Binärsystem erklären und physikalische Darstellungsformen benennen
- Sie können die einfachen Berechnungen im Binärsystem durchführen

Inhalt

- **Das Stellenwertsystem: ein Überblick**
- **Subtrahieren durch Addieren**
- **Das Dualsystem im Detail**

Das Stellenwertsystem: ein Überblick

- **Wir haben ein scheinbar natürliches Verständnis von Zahlen!**

  So interpretieren wir die Zahlenfolge der Preisangabe 110 CHF
  als einhundert und zehn Franken

- **Dahinter liegt aber eine Vereinbarung zur Darstellung von Zahlen durch geeignete Ziffern und ihres Wertes,**
  wie oben gezeigt, intuitiv das Dezimalsystem

aber..

- **diese Vereinbarung können wir an unsere Bedürfnisse anpassen!**

OST

Das Stellenwertsystem: ein Überblick

- **Das Dezimalsystem:** $R_{10} = 10$ (Basis); $Z_{10} = \{0,1,2,3,4,5,6,7,8,9\}$
- **Dabei gibt R_{10} den Wert zur Basis 10 an der d_{ten} Stelle an**
- **Z_{10} Beschreibt den Zahlenkörper oder die Menge der erlaubten Ziffern des Zahlensystems (hier nur die ganzen positiven Zahlen 1 bis 9 und der 0)**
- **Allgemein:** $N = d_n R^n + \ldots + d_1 R^1 + d_0 R^0$

Ziffer $\in Z_{10}$ **Wertigkeit**

OST

Das Stellenwertsystem: ein Überblick

- **Beispiel Dezimalsystem**
- **Allgemein: $N = d_n R^n + \ldots + d_1 R^1 + d_0 R^0$**

Ziffer $\in Z_{10}$ **Wertigkeit**

  - **Ist es ihnen aufgefallen?**
  - **Eine Zahl im Stellenwertsystem beschreibt ein Polynom!**
- **$N_{10} = 110$**

  $N_{10} = 1 * 10^2 + 1 * 10^1 + 0 * 10^0$

oder

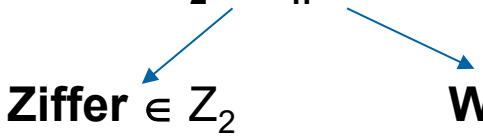
- **$N_{10} = 255$**

  $N_{10} = 2 * 10^2 + 5 * 10^1 + 5 * 10^0$

OST

Das Stellenwertsystem: ein Überblick

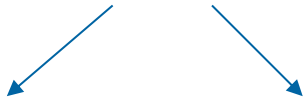
- **Das Dualsystem: $R_2 = 2$ (Basis); $Z_2 = \{0,1\}$**
- **Dabei gibt R_2 den Wert zur Basis 2 an der d_{ten} Stelle an**
- **Z_2 Beschreibt den Zahlenkörper oder die Menge der erlaubten Ziffern des Zahlen Systems (hier nur die Ganzen positiven Zahlen 0 und 1)**
- **Allgemein: $N_2 = d_n R^n + \ldots + d_1 R^1 + d_0 R^0$**



OST

Das Stellenwertsystem: ein Überblick

- **Beispiel Dualsystem**
- **Allgemein: $N = d_n R^n + \ldots + d_1 R^1 + d_0 R^0$**



Ziffer $\in Z_2$ **Wertigkeit**

- **$N_2 = 110$**

  $N_2 = 1 * 2^2 + 1 * 2^1 + 0 * 2^0$ Dezimal: 6

oder

- **$N_2 = 101$**

  $N_2 = 1 * 2^2 + 0 * 2^1 + 1 * 2^0$ Dezimal: 5

Das Stellenwertsystem: ein Überblick

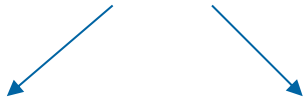
- **Das Oktalsystem: $R_8 = 8$(Basis); $Z_8 = \{0,1,2,3,4,5,6.7\}$**
- **Dabei gibt R_8 den Wert zur Basis 8 an der d_{ten} Stelle an**
- **Z_8 Beschreibt den Zahlenkörper oder die Menge der erlaubten Ziffern des Zahlen Systems (hier nur die Ganzen positiven Zahlen 0 bis 7)**
- **Allgemein: $N_8 = d_n R^n + \ldots + d_1 R^1 + d_0 R^0$**

Ziffer $\in Z_8$ **Wertigkeit**

OST

Das Stellenwertsystem: ein Überblick

- **Beispiel Oktalsystem**
- **Allgemein: $N = d_n R^n + \ldots + d_1 R^1 + d_0 R^0$**



Ziffer $\in Z_8$ **Wertigkeit**

- **$N_8 = 110$**

  $N_8 = 1 * 8^2 + 1 * 8^1 + 0 * 8^0$ Dezimal: 72

oder

- **$N_8 = 101$**

  $N_8 = 1 * 8^2 + 0 * 8^1 + 1 * 8^0$ Dezimal: 65

OST

Das Stellenwertsystem: ein Überblick

- **Das Hexadezimalsystem: R_{16} = 16(Basis);**
  Z_{16} = {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}
- **Dabei stellen A die 10_{16} und B die 11_{16} und F entsprechend die 15_{16} dar**
- **Dabei gibt R_{16} den Wert zur Basis 16 an der d_{ten} Stelle an**

- **Z_{16} Beschreibt den Zahlenkörper oder die Menge der erlaubten Ziffern des Zahlen Systems (hier nur die Ganzen positiven Zahlen 0 bis 9 und die Ziffern A, B, C, D, E, F)**

- **Allgemein: $N_{16} = d_n R^n + \ldots + d_1 R^1 + d_0 R^0$**

Ziffer $\in Z_{16}$ **Wertigkeit**

OST

Das Stellenwertsystem: ein Überblick

- **Beispiel Hexadezimalsystem**
- **Allgemein: $N = d_n R^n + \ldots + d_1 R^1 + d_0 R^0$**

Ziffer **Wertigkeit**

- **$N_{16} = 110$**

  $N_{16} = 1 * 16^2 + 1 * 16^1 + 0 * 16^0$ Dezimal: 272

oder

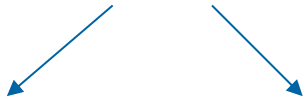
- **$N_{16} = B01$**

  $N_{16} = B * 16^2 + 0 * 16^1 + 1 * 16^0$ Dezimal: 2817

OST

Das Stellenwertsystem: ein Überblick

- **Beispiel Dezimalsystem**
- **Allgemein: $R = d_n R^n + \ldots + d_1 R^1 + d_0 R^0$**



Ziffer $\in Z_{10}$ **Wertigkeit**

- **$R_{10} = 110.13$**

  $N_{10} = 1 * 10^2 + 1 * 10^1 + 0 * 10^0 + 1 * 10^{-1} + 3 * 10^{-2}$

oder

- **$R_{10} = 255.55$**

  $N_{10} = 2 * 10^2 + 5 * 10^1 + 5 * 10^0 + 5 * 10^{-1} + 5 * 10^{-2}$

OST

Die bekanntesten Zahlenmengen

Symbol:	Bedeutung:	So sieht's aus:
$\mathbb{N}$	Menge der natürlichen Zahlen	$N=0$ $N=12$ $N=563$
$\mathbb{Z}$	Menge der ganzen Zahlen	$Z=-20$ $Z=12$
$\mathbb{Q}$	Menge der rationalen Zahlen	$Q=-2,5$ $Q=5$ $Q=14,002$
$\mathbb{R}$	Menge der reellen Zahlen	$R=-5,142\ldots$ $R=12,542\ldots$

Subtrahieren durch Addieren

- **753 + 247 = 1000**

➢ **gehen wir davon aus, wir haben bei Tausend einen Überlauf.**

Dann gilt:

- **753 + 247 = 0 und daraus folgt 753 = - 247**
- **Dann wäre 620 – 247 =**
- **620 + 753 = 1373**
- **mit Überlauf bei Tausend**
- **= 373**

OST

Subtrahieren durch Addieren

Wie kann man die negative additive Zahl sonst noch berechnen:

- **Gesucht ist die additive Zahl von -247 also 753**
- **Wir bestimmen das Komplement von 247 =**

+ 752 und addieren 1 = 753

999

- **Das geht immer!!!**
- **Und in jedem Zahlensystem!!!**

OST

Subtrahieren durch Addieren

Beispiel 1

- **760 - 233 = ?**
- **Gesucht ist die additive Zahl von - 233**
- **Wir bestimmen das Komplement von 233**

$$\begin{array}{rr} & 233 \\ + & 766 \\ \hline & 999 \end{array}$$

- **und addieren 766 +1 = 767 – d.h. das Zehnerkomplement von 233 ist 767**
- **760 + 767 = ~~1~~527 also ist 760 – 233 = 527 Bingo** ✓

OST

Oder Beispiel 2

- **387 – 26 = ?**
- **Gesucht ist die additive Zahl von - 026 also 026**

+ 973

999

973 + 1 = 974

- **387 + 974 = ~~1~~361 also ist 387 - 26 = 361 Bingo** ✓

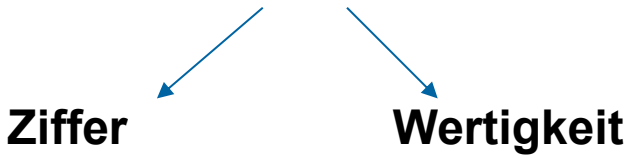
✓ **Das funktioniert auch in anderen Zahlensystemen, probieren Sie es aus!**

✓ **So können wir also die Subtraktion durch eine Addition ersetzen, das werden wir später noch brauchen, warum?**

OST

Das Dualsystem im Detail

- **Allgemein: $N = d_n R^n + \ldots + d_1 R^1 + d_0 R^0 + d_{-1} R^{-1} + d_{-2} R^{-2}$**



- **$N_2 = 110{,}110$**
  - $N_2 = 1 * 2^2 + 1 * 2^1 + 0 * 2^0 + 1 * 2^{-1} + 1 * 2^{-2}$ Dezimal: 6,75
- **oder**
- **$N_2 = 101{,}101$**
  - $N_2 = 1 * 2^2 + 0 * 2^1 + 1 * 2^0 + 1 * 2^{-1} + 0 * 2^{-2} + 1 * 2^{-3}$ Dezimal: 5

OST

Das Dualsystem im Detail

- **Wie kann die Dezimalzahl 25 einfach in eine Dualzahl gewandelt werden?**
- **Erinnerung rechts shift ist eine Division durch 2**
- **Es entsteht nur dann ein Rest, wenn die Stelle $2^0 = 1$ ist.**
- **Daraus folgt:**
  - 25 : 2 = 12 Rest 1
  - 12 : 2 = 6 Rest 0
  - 6 : 2 = 3 Rest 0
  - 3 : 2 = 1 Rest 1
  - 1 : 2 = 0 Rest 1

➢ $11001 = 1*2^4 + 1* 2^3 + 1*2^0 = 16 + 8 + 1 = 25$

OST

Das Dualsystem im Detail

- **Wie kann die Dezimalzahl 0.75 einfach in eine Dualzahl gewandelt werden?**
- **Erinnerung links shift ist eine Multiplikation mit 2**
- **Entsteht ein Wert grösser gleich 1, ist die Bitstelle $2^{-1} = 1$.**
- **Daraus folgt:**
  - 0.75 * 2 = 1.5 Stelle 1
  - 0.5 * 2 = 1.0 Stelle 1
  - 0 * 2 = 0

  ➢ $0.11 = 2^{-1} + 1*2^{-2} = 0.5 + 0.25 = 0.75$

  - Oder:

  - 0.33 * 2 = 0.66 Stelle 0
  - 0.66 * 2 = 1.32 Stelle 1
  - 0.32 * 2 = 0.64 Stelle 0
  - 0.64 * 2 = 1.28 Stelle 1
  - …

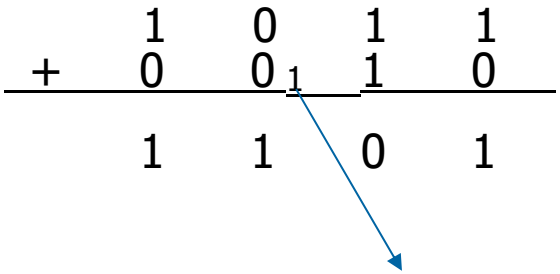
  ➢ $0.0101 = 2^{-2} + 2^{-4} = 0.25 + 0.0625 + .. = 0.3125$

Was schliessen sie hieraus???

OST

- *Die Stelle i ist damit $f_i(x, y) = x_i \oplus y_i \oplus c_i$ (exclusives ODER)*
- *$\Rightarrow$ $f_i(x, y) = 1$ genau dann, wenn eines oder alle drei von x_i, y_i, $c_i = 1$*
- *$\Rightarrow$ $c_{i+1}(x, y) = 1$ genau dann, wenn zwei oder alle drei von x_i, y_i, $c_i = 1$*

- ***Schriftliche Addition ist somit ähnlich zu den Dezimalzahlen***



- Übertrag: erzeugt im Rechner ein carry ripple!
- Überträge der MSBs gehen verloren (*carry bit*)
  Mathematische gesehen liegt eine Abgeschlossenheit vor

OST

Das Dualsystem im Detail: Subtraktion

- ***Geistesblitz: wir führen die Subtraktion auf eine Addition zurück***
- ***Beispiel 5 – 6 = -1***

$$0\,1\,0\,1 \;-\; 0\,1\,1\,0$$

$$-\; 0\,1\,1\,0 \;\Rightarrow 1001 + 1 = 1010$$

$$0\,1\,0\,1 + 1010 = 1\,1\,1\,1$$

-1 ist also 1 1 1 1

- ***Beispiel 5 – 7 = -2***

$$0\,1\,0\,1 \;-\; 0\,1\,1\,1$$

$$-\; 0\,1\,1\,1 \;\Rightarrow 1000 + 1 = 1001$$

$$0\,1\,0\,1 + 1001 = 1\,1\,1\,0$$

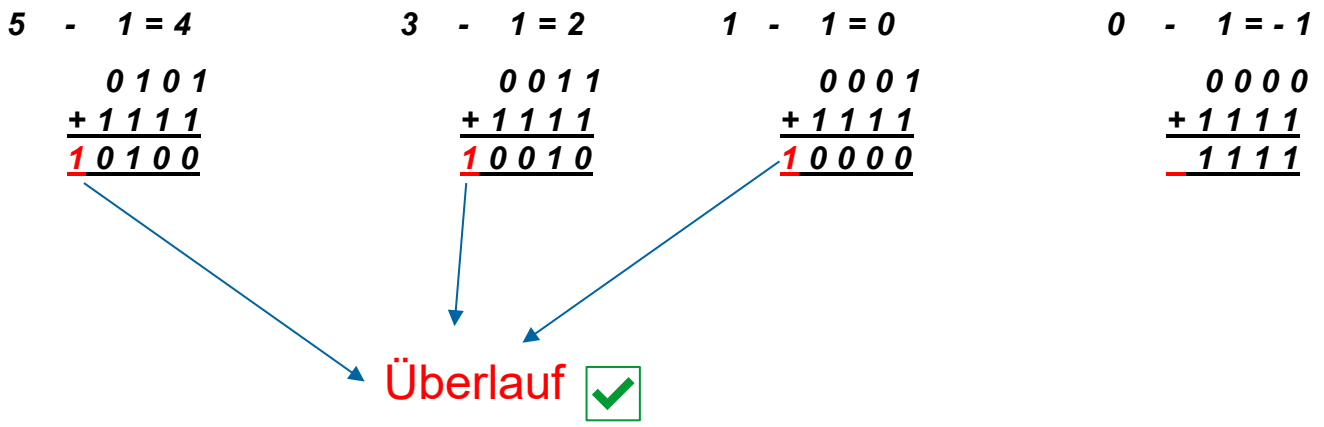
-2 ist also 1 1 1 0

OST

Das Dualsystem im Detail: Subtraktion

- ***Wir haben festgestellt Beispiel 5 – 6 = -1 = 1 1 1 1 ist! Geht das nicht einfacher?***
- ***erinnern wir uns ans Kompliment +1 als negative Zahl***
- ***Dann wird aus 0 0 0 1 Komplement bilden 1 1 1 0 eine 1 addieren und***
- ***Wir haben die negative Zahl -1 = 1 1 1 1***

- ***Beispiele:***



OST

Das Dualsystem im Detail: Negative Zahlen bestimmen

- ***Beispiel***

  1 1 1 1 ist also eine **negative** Zahl

  Wert bestimmen?

  Rückwärts rechnen (umkehren des Zweierkomplements)

  1 1 1 1 – 1 = 1 1 1 0

  Negieren

  1 1 1 0 wird 0 0 0 1 der Wert ist also 1, aber negativ, -1

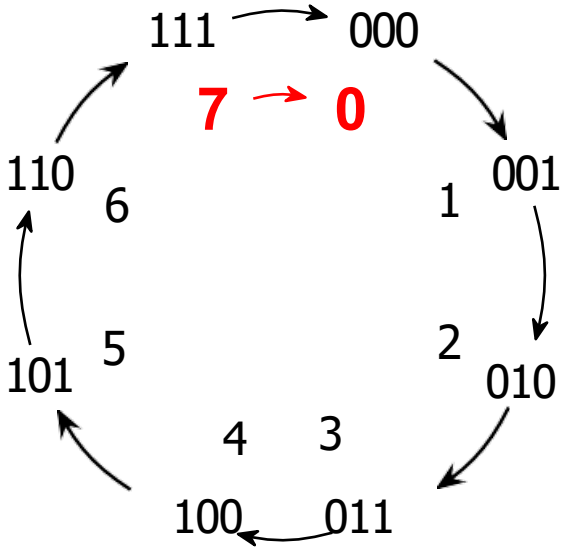
- ***Versuchen sie es selbst!***

  Was repräsentiert die Bitfolge als Zahl 1 1 1 0 ??

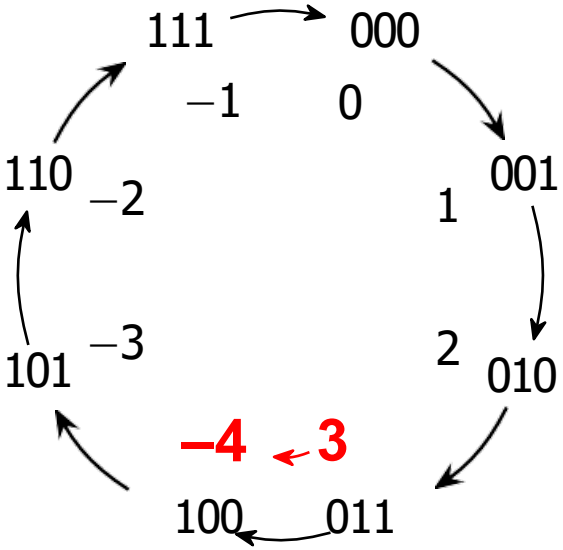
OST

Inkrement und Dekrement sind unabgängig von der Interpretation als signed oder unsigned.

Mathematisch befinden wir uns auf einem Ring, Abgeschlossenheit des Zahlenbereiches



unsigned: Überlauf von 7 auf 0



signed: Überlauf von 3 auf –4

OST

Das Dualsystem im Detail: Wertebereich

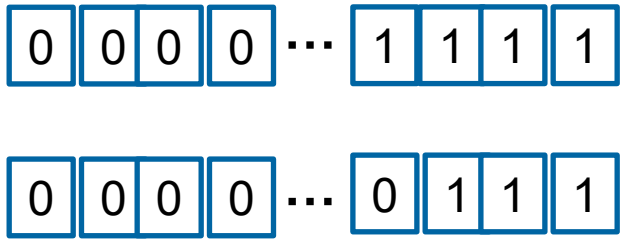
Bei n Bit reichen die Zahlen von $-2^{n-1}, -(2^{n-1}-1), \ldots, -1, 0, 1, \ldots 2^{n-1}-1$:

	signed	unsigned
4 Bit	$-8 \ldots 7$	$0 \ldots 15$
8 Bit	$-128 \ldots 127$	$0 \ldots 255$
16 Bit	$-32K \ldots 32K-1$	$0 \ldots 64K-1$
32 Bit	$-2G \ldots 2G-1$	$0 \ldots 4G-1$

- für -2^{n-1} gibt es also kein positives Äquivalent,
- 0 ist eigentlich weder positives noch negatives,
- wird aber zu positiv dazugezählt, dann gleiche Anzahl positve und negativer Zahlen.

Veranschaulichung

unsigned



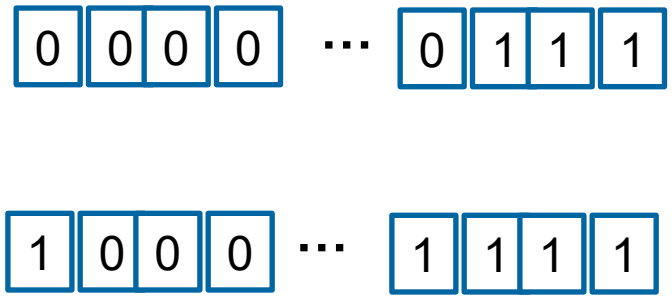
Nur positive Zahlen

Binärzahlen für n bit: [0 … 2^n – 1] also [0 … 15]

mit MSB = 0

n bit: [0 … 2^{n-1} – 1] also [0 … 7]

signed



positive und negative Zahlen

Positive Zahlen, MSB = 0

- n bit: [0 … 2^{n-1} – 1] also [0 … 7]

Negative Zahlen, MSB =1

- n bit: [-(2^{n-1}]) … (-1)] also [-8 … -1]

OST

Das Dualsystem im Detail: Unsigned Multiplikation

- *a b* ist eine Summe von Links-Shifts:
- Sei a =3 und b= 5
- *0011 * 0101*

```
  0 0 0 0
    0 0 1 1
      0 0 0 0
        0 0 1 1
-------------------
0 0 0 0 1 1 1 1 = 15
```

Für jedes $b_i = 1$ müssen wir $a \cdot 2^i$ zum Ergebnis addieren

Beispiel: $13_d \cdot 6_d = 1101 \cdot 0110 = 110100 + 11010$

$$
\begin{array}{r}
110100 \\
+\ \ 11010 \\
\hline
01001110
\end{array}
\quad = 2^6 + 2^3 + 2^1 + 2^1 = 78_d
$$

OST

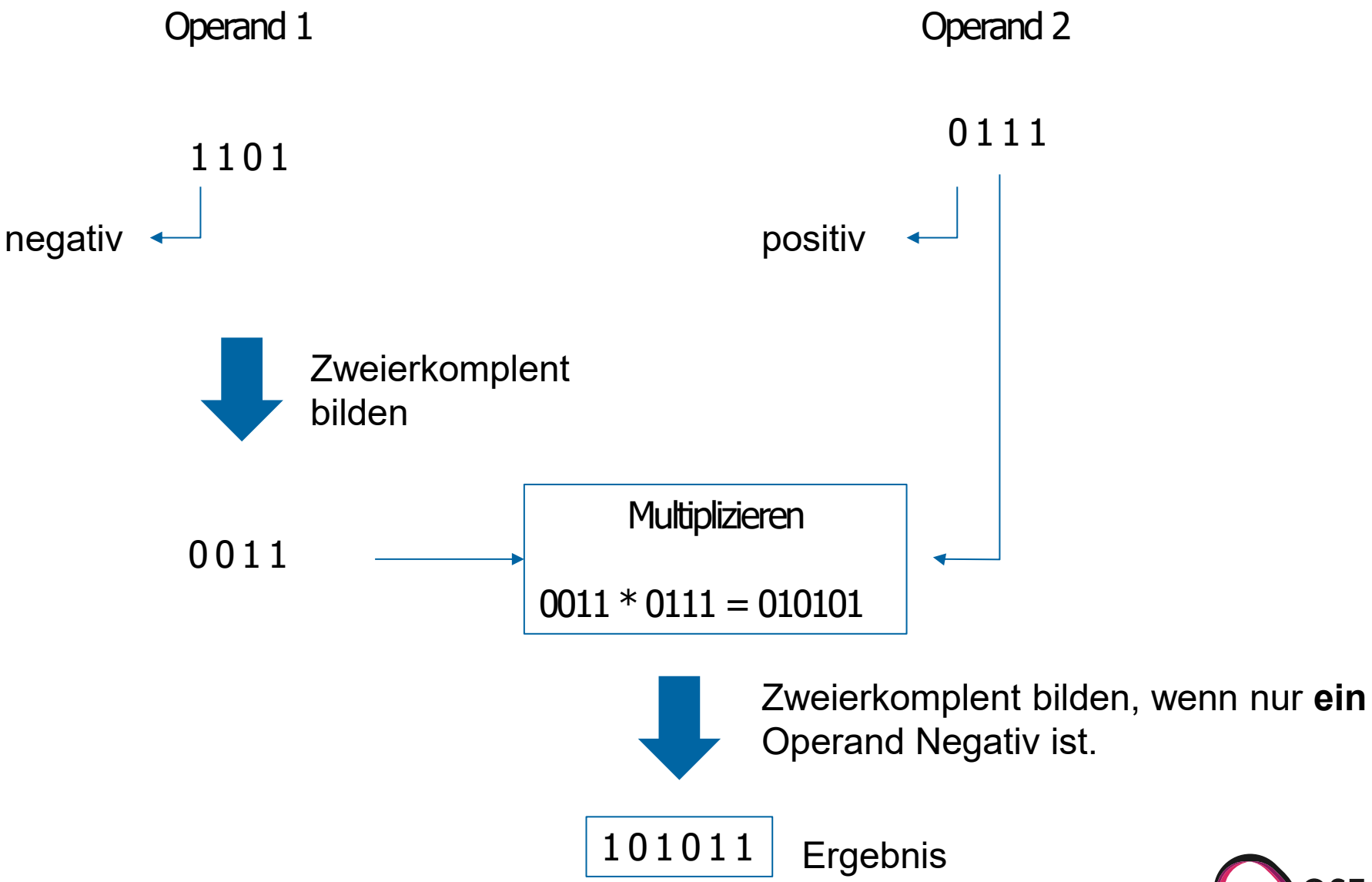
Das Dualsystem im Detail: Multiplikation – Wertebereich

Das grösst-mögliche Produkt zweier *n*-Bit-Zahlen ist das Quadrat der grössten *n*-Bit-Zahl:

$$\begin{aligned}(2^n - 1) \cdot (2^n - 1) &= (2^n - 1)^2 \\ &= (2^n)^2 - 2 \cdot 2^n + 1 \\ &= 2^{2n} - 2^{n+1} + 1\end{aligned}$$

OST

Das Dualsystem im Detail: Signed Multiplikation



Das Dualsystem im Detail: Division

- **Die Division ist eine relativ aufwändige Operation**
- **Es gibt keine gemeinsame Operation für signed und unsigned**
- **Unsigned: Iteratives Verfahren für $i = n - 1$ bis $i = 0$:**
  - Man überprüft ob $b \cdot 2^i$ in die $n - i$ obersten Bits von a «passt»
  - Wenn ja, dann setzt man im Ergebnis Bit i und zieht $b \cdot 2^i$ von a ab

Das Dualsystem im Detail: Division

- **Division sollte gemieden werden, da sehr langsame Operation**
  - Bei 32 Bit: 20mal so lange wie eine Multiplikation
  - Bei 64 Bit: 80mal so lange wie eine Multiplikation
- **kann für Zweierpotenzen durch Rechts-Shift ersetzt werden**
- **kann bei anderen Konstanten oft durch Multiplikation oder Umstellen der Formel ersetzt werden**
- **Moderne Compiler tun dies oft automatisch, aber nicht immer**

Wiederholung

- **Stellenwertsystem**
  - **Eine Zahl im Stellenwertsystem ist ein Polynom! Was bedeutet das?**
  - **Was beschreibt die Wertigkeit und was die Potenz der Wertigkeit**
  - Wie unterscheiden sich Dual-, Oktal-, Dezimal- und Hexadezimalsystem

Wiederholung – Zaubertrick (aus Übung)

1	3	5	7
9	11	13	15
17	19	21	23
25	27	29	31

2	3	6	7
10	11	14	15
18	19	22	23
26	27	30	31

4	5	6	7
12	13	14	15
20	21	22	23
28	29	30	31

8	9	10	11
12	13	14	15
24	25	26	27
28	29	30	31

16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31

OST

Wiederholung Dezimalzahl Dual darstellen

- **Wie kann die Dezimalzahl 25 einfach in eine Dualzahl gewandelt werden?**
- **Erinnerung rechts shift ist eine Division durch 2**
- **Es entsteht nur dann ein Rest, wenn die Stelle 20 = 1 ist.**
- **Daraus folgt:**
  - 25 : 2 = 12 Rest 1
  - 12 : 2 = 6 Rest 0
  - 6 : 2 = 3 Rest 1
  - 3 : 2 = 1 Rest 1
  - 1 : 2 = 0 Rest 1

➢ $11001 = 1*2^4 + 1* 2^3 + 1*2^0 = 16 + 8 + 1 = 25$

OST

Wiederholung Gebrochene Dezimalzahl Dual darstellen

- **Wie kann die Dezimalzahl 0.75 einfach in eine Dualzahl gewandelt werden?**
- **Erinnerung links shift ist eine Multiplikation mit 2**
- **Entsteht ein Wert grösser gleich 1, ist die Bitstelle $2^{-1} = 1$.**
- **Daraus folgt:**
  - 0.75 * 2 = 1.5 Stelle 1
  - 0.5 * 2 = 1.0 Stelle 1
  - 0 * 2 = 0

  ➢ $0.11 = 2^{-1} + 1*2^{-2} = 0.5 + 0.25 = 0.75$

  - Oder:

  - 0.33 * 2 = 0.66 Stelle 0
  - 0.66 * 2 = 1.32 Stelle 1
  - 0.32 * 2 = 0.64 Stelle 0
  - 0.64 * 2 = 1.28 Stelle 1
  - …

  ➢ $0.0101 = 2^{-2} + 2^{-4} = 0.25 + 0.0625 + .. = 0.3125$

Was schliessen sie hieraus???

OST

Wiederholung Subtraktion Multiplikation

- **Subtraktion durch Addition des Zweierkompliments:**

$$
\begin{array}{rl}
5 - 1 = 4 & \\
 & 0101 \\
 & +1111 \\
\hline
 & 10100
\end{array}
$$

- **Multiplikation durch Summe von Links-shifts**

$$
\begin{array}{l}
0011 * 0101 \\
\quad 0011 \\
\quad\quad 0011 \\
\hline
00001111 = 15
\end{array}
$$

- **Oder Polynommultiplikation**
- **$3*3 = (2^1 + 2^0) * (2^1 + 2^0) = 2^3 + 2^0$**

OST

Lernziele

Sie kennen/können:

- aus einer Bitgrösse die maximal darstellbare Zahlenmenge ermitteln
- für eine Zahlenmenge die mindesten Wert von Binärstellen bestimmen
- die Präfixschreibweise und können einfache Berechnungen durchführen
- Verstehen die Grenzen der Zahlendarstellung im Computer

Inhalt

- **Physikalische Darstellung der logischen Werte**
- **Grundbegriffe**
- **Typische Notationen**
- **Binärzahlen und Computertechnik**
- **Präfixe und ihre Bedeutung**
- **Praxis des Hexadezimalsystems**

Frage

- **Warum verwendet man nicht einfach Dezimalzahlen?**

OST

Zwei Zustände sind technisch einfach zu realisieren

Technisch einfach sind zwei Zustände unterscheidbar, z.B.

Lochstreifen	Loch↔kein Loch
Akustisches Morsen	Kurzer Ton↔langer Ton
Optisches Morsen	Kurzer Lichtimpuls↔langer Lichtimpuls
Elektrische Spannung	hoch↔niedrig
CD	Wechsel von tiefer zu flacher Delle↔kein Wechsel
Magnetische Datenträger	Wechsel der Magnetisierungsrichtung↔kein Wechsel

Bit (binary digit)

Definition

Ein **Bit** ist eine Grösse, die zwei Zustände annehmen kann.

Welchem Zustand die logische 0 und welchem die logische 1 entspricht, hängt dabei von Hersteller, Standards und Konventionen, usw. ab.

Begriffe im binären Zahlensystem

Bit = 1 Gesetztes Bit (*set bit*)
Bit = 0 Gelöschtes Bit (*cleared bit*)

LSB *Least Significant Bit*, niederwertigstes Bit, Bit 0.

MSB *Most Significant Bit*, högstwertigstes Bit, Bit n-1 in einer n-stelligen Binärzahl.

Begriffe im Binären Zahlensystem

Nibble	Binärzahl mit 4 Bit. Grössere Binärzahlen werden als Nibbles gruppiert, z.B. 1101 0101.
Oktett	Eine Binärzahl mit 8 Bit. Wird in der Netzwerktechnik bevorzugt
Byte	Für unsere Zwecke und im Bereich von Speichern = Oktett.

Beachte: Führende Nullen werden bei Binärzahlen normalerweise immer mitgeschrieben, um anzuzeigen, wie viele Bits verwendet werden.

Vergleiche z.B. 0101 (4 Bit) und 0000’0101 (8 Bit).

Notation einzelner Bits

Die Bits einer Binärzahl *b* werden häufig durch Indizes einzeln bezeichnet, z.B.:

b =	1	0	1	0
Stelle	b_3	b_2	b_1	b_0

$b_3 = 1$		b[3] = 1
$b_2 = 0$	oder	b[2] = 0
$b_1 = 1$		b[1] = 1
$b_0 = 0$		b[0] = 0

Notation von Unterbereichen von Bits

Zusammenhängende Bereiche von Bits innerhalb einer Binärzahl werden häufig durch Indizes mit .. bezeichnet, z.B.:

$$b = 1010$$

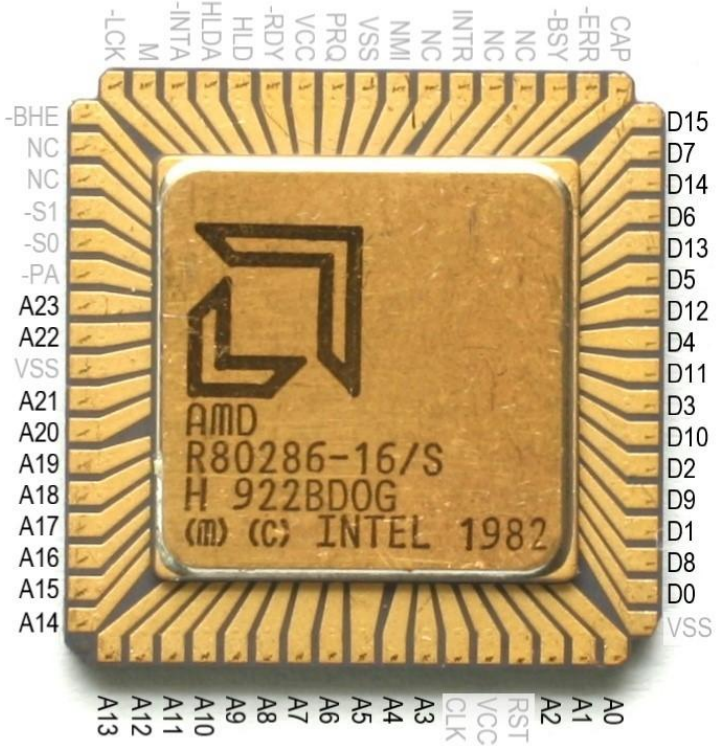
$b_{3...1} = 101$ $\quad$ $b_3 \ldots b_1 = 101$ $\quad$ $b[3..1] = 101$

Zusammenhang Binärzahlen und Computertechnik

- **Jede Menge an Bits in einem Computer kann als**
  - eine Binärzahl
  - ein Tupel
  - eine Menge
  - ein Vektor
  - ein Polynom
  - ein Zustand

  interpretiert werden

Beispiel Binärzahlen und Computertechnik: 80286 Pins



- Adresspins $A_{23}...A_0$ stellen eine 24-Bit Binärzahl dar
- Datenpins $D_{15}...D_0$ stellen eine 16-Bit Binärzahl dar

⇒ Die Pins liegen nicht immer genau Nebeneinander (das hat Design-Gründe wie bspw. Wärmeverteilung, Signalintegrität, Vereinfachung des Herstellungsprozesses usw.)

Neue Prozessoren ähnlich aber komplexer (mehr Kontakte auf kleinerem Raum)

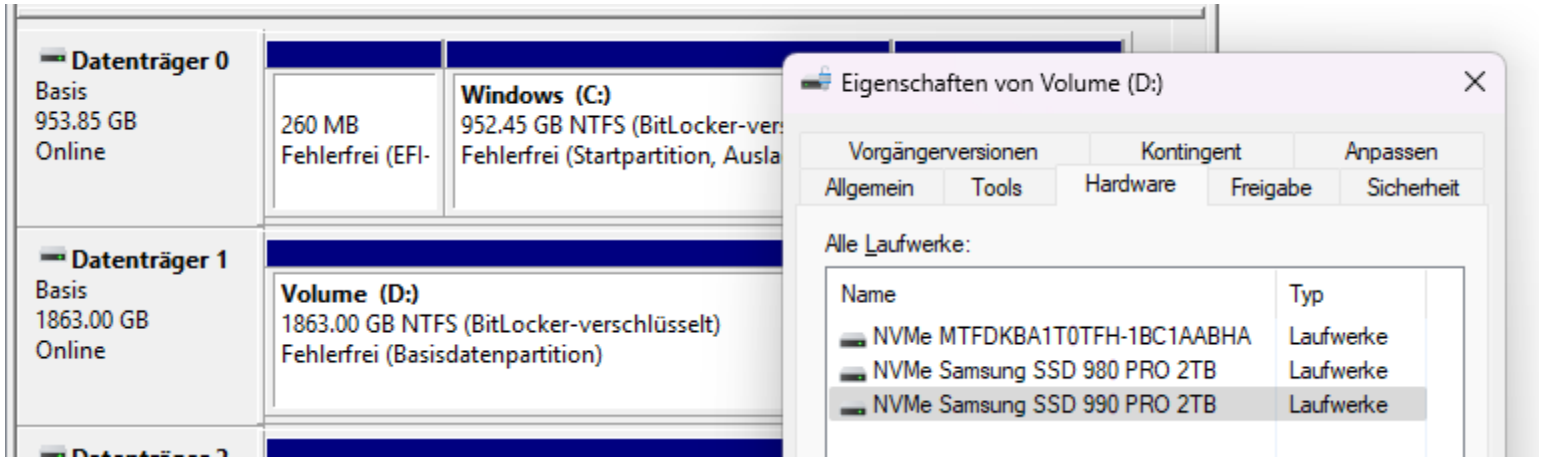
OST

Präfixe und ihre Bedeutung

1× **Samsung** 990 Pro (2000 GB, M.2 2280)

Versendet

Kein Rückgaberecht, Garantie bis 24.1.2028



Gekauft 2000 GB, erhalten 1863GB – hää?

OST

Präfixe und ihre Bedeutung

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

Beachte: Die Dezimalpräfixe sind hier eigentlich falsch bzw. ungenau. Sehr oft verwendet man in der Informatik Dezimalpräfixe, meint aber Binärpräfixe – wir handhaben das hier auch so.

Denn: 1 MiB (Mebibyte) = 1‘024 KiB (Kibibyte) = 1‘048‘576 Byte
(im Gegensatz zu 1 MB = 1‘000 KB = 1‘000‘000 Byte nach SI-Norm)

OST

Multiplikation bei Präfixschreibweise

Um Zahlen in Präfixschreibweise zu multiplizieren:

1. Umformen von Präfixschreibweise in Potenzschreibweise
2. Addition der Exponenten
3. Umformen von Potenzschreibweise in Präfixschreibweise

Beispiel:

$$128\text{K} \cdot 64\text{M} \underset{1.}{=} 2^7 \cdot 2^{10} \cdot 2^6 \cdot 2^{20} = 2^{17+26} \underset{2.}{=} 2^{43} = 2^3 \cdot 2^{40} \underset{3.}{=} 8\text{T}$$

Division bei Präfixschreibweise

Um Zahlen in Präfixschreibweise zu dividieren:

1. Umformen von Präfixschreibweise in Potenzschreibweise
2. Subtraktion der Exponenten
3. Umformen von Potenzschreibweise in Präfixschreibweise

Beispiel:

$$64\text{M}/128\text{K} \underset{1.}{=} 2^6 \cdot 2^{20} \,/\, (2^7 \cdot 2^{10}) = 2^{26-17} \underset{2.}{=} 2^9 \underset{3.}{=} 512$$

Aufgabe

siehe Folien oben

Wieviele Datensätze zu je 512KB passen in einen Speicherbereich der Grösse 1TB?

TB = TiB und KB = KiB

1. Umformen in Potenzschreibweise
   1 TiB = 2 ^ 40
   512 KB = 512 * 2^10 = 2^9 * 2^10
2. Subtraktion der Exponenten
   2^40 / 2^19 = 2^(40-19) = 2^21
3. Umformen von Potenzschreibweise in Präfixschreibweise
   2^21 = 2'097'152
   Lösung = 2'097'152 Mal

Ohne Umwandlung gäbe es 10^12 / 512'000 = 1'953'125 Mal

andere Vorgehensweise (weniger schön)
1 TiB = 2^40= 1'099'511'627'776
512 KB = 512*2^10 = 524'288 Bytes
2. Subtraktion der Exponenten
1'099'511'627'776 Bytes / 524'288 Bytes = 2'097'152 Mal

Decimal System (SI)			Binary System		
Name	Symbol	Decimal Unit	Name	Symbol	Decimal Unit
Kilobyte	KB	10^3	Kibibyte	KiB	2^10
Megabyte	MB	10^6	Mebibyte	MiB	2^20
Gigabyte	GB	10^9	Gigibyte	GiB	2^30
Terabyte	TB	10^12	Tebibyte	TiB	2^40
Petabyte	PB	10^15	Pebibyte	PiB	2^50
Exabyte	EB	10^18	Exbibyte	EiB	2^60
Zettabyte	ZB	10^24	Zebibyte	ZiB	2^70
Yottabyte	YB	10^21	Yobibyte	YiB	2^80

atatus

Andere Aufgabe: wieviele Datensätze zu 128 Byte passen in 4 KB?
1.) 128 Byte = 2^7 und 4KiB = 2^2 * 2^10
2.) 2^(12-7) = 2^5
3.) 32

Das Hexadezimalsystem

Motivation

- Binärzahlen sind gut für Computer, aber unhandlich lang für Menschen
- Die Darstellung von Binärzahlen im Dezimalsystem ist nicht unmittelbar, z.B.
  - Dezimal 10 ist binär 1010
  - Dezimal 100 ist binär 0110'0100

$\Rightarrow$ Hexadezimalzahlen sind so kompakt wie Dezimalzahlen (und kompakter) und lassen sich unmittelbar aus Binärzahlen erzeugen.

Das Hexadezimalsystem

Einführung

- Mit 4 Bit lassen sich die $2^4 = 16$ verschiedenen Zahlen 0 bis 15 darstellen.
- Für die 6 Zahlen 10 bis 15 schreiben wir die Buchstaben A bis F.

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

- Das Hexadezimalsystem ist ein Stellenwertsystem zur Basis 16 mit den Ziffern 0 bis F.

Das Hexadezimalsystem

Grösste darstellbare Zahlen

Die grösste Hexadezimal-Zahl mit *n Stellen* hat an jeder Stelle ein F:

$$\underbrace{\mathrm{F}\cdots\mathrm{F}}_{n} = 16^{n}{}_{\mathrm{d}} - 1 = 2^{4n}{}_{\mathrm{d}} - 1,$$

z.B.:

$$\mathrm{FFFF_h} = 16^{4}{}_{\mathrm{d}} - 1 = 2^{16}{}_{\mathrm{d}} - 1 = 64\mathrm{K} - 1 = 65535_{\mathrm{d}}$$

Das Hexadezimalsystem

Beobachtungen

- Eine hexadezimale Ziffer entspricht genau einem Nibble.

⇒ Binärzahlen werden deshalb in Nibbles gruppiert.

- Zwei hexadezimale Ziffern entsprechen genau einem Byte.

⇒ Binärdaten werden deshalb byteweise als zweistellige Hexadezimalzahlen dargestellt, z.B.:

12 34 CA FE anstatt 0001'0010 0011'0100 1100'1010 1111'1110

Das Hexadezimalsystem

Beobachtungen

- Allgemein benötigt eine n-stellige Hexadezimalzahl $4n$ Bits.

$\Rightarrow$ Wie bei Binärzahlen werden führende Nullen bei Hexadezimalzahlen immer mitgeschrieben.

- Die n-te Potenz von 16 ist hexadezimal die Ziffer 1 mit n Nullen, z.B.:

$$1000_{\mathrm{h}} = 16^{3}{}_{\mathrm{d}} = 2^{12}{}_{\mathrm{d}} = 4\mathrm{K}$$

Notation der Basis einer Zahl

- **Die Basis (des Stellenwertsystems) wird oft durch ein Präfix oder Suffix gekennzeichnet.**
  - In der Literatur und in Programmiersprachen gibt es verschiedene Konventionen
  - Man muss die jeweilige Konvention der Dokumentation entnehmen
- **Zahlen ohne Prä- oder Suffix werden meist dezimal interpretiert (aber nicht immer!)**
  - In Programmiersprachen nahezu immer
  - In der Literatur kann Interpretation vom Kontext abhängen
- **Wir verwenden einen Prä- oder Suffix, wenn die Bedeutung aus dem Kontext unklar ist**

  Bspw.
  - 0x021711 Präfix für Hexadezimalzahlen in C, C++, C#, Java …
  - #x021711 in Lisp
  - 021711h Suffix häufig in Assembler, Ada

Unterscheidung von Zahlen: Beispiele für Konventionen

Konvention	Dezimal	Hexadez.	Binär	Kommentar
Intel Manual	1011	1011H	1011B	
Wikipedia	1011_{10}	1011_{16}	1011_2	Häufig ohne Suffixe, dann kontextabhängig
C, C++, Java, Python, PHP	1011	0x1011	0b1011	Binärzahlen häufig nicht möglich
Pascal, Delphi Ada Visual Basic	1011 1011 1011	$1011 16#1011# &H1011	%1011 2#1011# &B1011	

Zusammenfassung und Fragen

- **Nennen sie typische physikalische Darstellung der logischen Werte in einem Computersystem, in einem Computernetz.**
- **Erklären sie welche Bedeutung Präfixe haben und wo sie eingesetzt werden.**
- **Was verstehen sie unter LSB, MSB, Nibble?**
- **Was ist der Vorteil des Hexadezimalsystems und welche Rolle spielt der Nibble?**
- **Nennen sie Beispiele, wo das Hexadezimalsystem verwendet wird.**

Zwei Zustände sind technisch einfach zu realisieren

Technisch einfach sind zwei Zustände unterscheidbar, z.B.

Lochstreifen	Loch↔kein Loch
Akustisches Morsen	Kurzer Ton↔langer Ton
Optisches Morsen	Kurzer Lichtimpuls↔langer Lichtimpuls
Elektrische Spannung	hoch↔niedrig
CD	Wechsel von tiefer zu flacher Delle↔kein Wechsel
Magnetische Datenträger	Wechsel der Magnetisierungsrichtung↔kein Wechsel

Wiederholung: Grundbegriffe

Zusammenhang Binärzahlen und Computertechnik

- **Jede Menge an Bits in einem Computer kann als**
  - eine Binärzahl
  - ein Tupel
  - eine Menge
  - ein Vektor
  - ein Polynom
  - ein Zustand

interpretiert werden

OST

Wiederholung: Präfixe und ihre Bedeutung

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

Beachte: Sehr oft verwendet man in der Informatik Dezimalpräfixe, meint aber Binärpräfixe.

OST

Das Hexadezimalsystem

Einführung

- Mit 4 Bit lassen sich die $2^4 = 16$ verschiedenen Zahlen 0 bis 15 darstellen.
- Für die 6 Zahlen 10 bis 15 schreiben wir die Buchstaben A bis F.

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

- Das Hexadezimalsystem ist ein Stellenwertsystem zur Basis 16 mit den Ziffern 0 bis F.

Inhalt

- **Die Idee der Codierung**
- **Eine weitere Codierung zur Darstellung von negativen Zahlen der Exzess-Code**
- **Codierung der FIX-Kommazahlen und**
- **der Gleitkommazahlen**
- **Fehler, die sie nicht verhindern können, aber kennen sollten**
- **ASCII**

Lernziele

- **Sie verstehen, das Bitfolgen nur eine Codierung von Inhalten ist und die Semantik der Bitfolge vom Kontext abhängig ist**
- **Sie kennen den Exzess-Code und können einfache Berechnungen durchführen.**
- **Sie kennen verschiedene Codierungen für negative Zahlen**
- **Sie verstehen den Code oder Aufbau von Fixkomma – Zahlen und**
- **Gleitkomma – Zahlen**
- **Sie können einfache Berechnungen zu Fix- und Gleitkommazahlen durchführen**
- **Sie verstehen, wie Rundungsfehler entstehen und können diese bestimmen**
- **Sie kennen den ASCII Code und seine Erweiterungen**

Die Idee der Codierung I

Eine Menge von Nullen oder Einsen bzw. deren physikalischen Ausprägungen

ergeben zunächst keinen Sinn,

erst

wenn wir die Codierung dahinter verstehen

können wir die Daten interpretieren!

Beispiele:

Codierung

- einer Zahl
- eines Textes
- einer PDU
- einer Verschlüsselung
- …

Standardisierung von Encodings

- Encodings sind nicht eindeutig: Es können unterschiedliche bijektive Funktionen E_0, E_1, etc definiert werden, die Definition ist *willkürlich*

$\Rightarrow$ Aus $E(z)$ kann man z nur ableiten, wenn man E bzw. E^{-1} kennt

$\Rightarrow$ Das Encoding muss Sender und Empfänger bekannt sein

$\Rightarrow$ Sender und Empfänger müssen das Encoding miteinander *vereinbaren*

$\Rightarrow$ Bei vielen Sendern und Empfänger bietet es sich an, ein Encoding als *Standard* zu vereinbaren

Darstellung negativer Zahlen

- **Was kennen wir bis jetzt**
  - Betrag und Vorzeichen
  - (b-1)-Komplement (2x Null)
  - b-Komplement
- **Neu**
  - Exzess

Vorzeichen und Betrag (Sign und Magnitude)

- **Ein einfaches binäres System zur Darstellung von positiven und negativen Ganzzahlen.**
- **Struktur:**
  - **Vorzeichenbit (Sign):** Das erste Bit bestimmt das Vorzeichen (0 für positive, 1 für negative Zahlen).
  - **Betrag (Magnitude):** Die restlichen Bits repräsentieren den absoluten Wert der Zahl.
- **Beispiele:**
  - **Positive Zahl (+5):** 0101
  - **Negative Zahl (-5):** 1101

➢ **Was fällt auf?**

Vorzeichen und Betrag (Sign und Magnitude)

- **Anmerkungen:**
  - Einfach und intuitiv, jedoch zwei Darstellungen von Null (+0 und -0).
  - Weniger effizient für mathematische Operationen im Vergleich zu anderen Methoden wie Zweierkomplement

    hier sind zwei Prozessschritte:
    - 1) Vorzeichen und
    - 2) Operation mit Beträgen
  - **Anwendung:** Vorwiegend in Bildungskontexten zur Erklärung der binären Zahlendarstellung.

Das (b-1)-Komplement zur Darstellung negativer Zahlen

- **Das (b-1)-Komplement ist eine Methode zur Darstellung von negativen Zahlen in einem b-basierten Zahlensystem.**
  - Für Binärzahlen (b = 2) wird dies auch als Einerkomplement bezeichnet.
- **Funktionsweise**
  - **Einerkomplement (Binär):** Um das Einerkomplement einer Zahl zu bilden, werden alle Bits der Zahl invertiert (0 wird zu 1, und 1 wird zu 0).
  - **Zahl +5 im Binärsystem:** 0000 0101
  - **Einerkomplement von +5:** 1111 1010 (Invertierung jedes Bits)
  - **Allgemein:**
  - für jede Ziffer z des Stellenwertsystems der Zahl N:
  - wird das Komplement gebildet: (b -1) - z
    - B**eispiel: sei b = 5**
      - **Dann bildet sich das Komplement der Zahl N_5 = 2 3 4 $_5$**
      - **Komplement** (b -1) – z = 4 –z daraus folgt **2 1 0 $_5$**

Das (b-1)-Komplement zur Darstellung negativer Zahlen

- **Anmerkungen**
  - Einfach zu berechnen durch Bitinversion.
  - Zwei Darstellungen von Null: +0 und −0 (z.B. 0000 0000 und 1111 1111 im 8-Bit-System).
  - Addition kann zu einem zusätzlichen Schritt führen: "End-Around Carry".
  - Verwendet in einigen älteren Computersystemen und für Lehrzwecke, um das Konzept der Zahlendarstellung und -umwandlung zu vermitteln.
  - Funktioniert auch in Zahlensystemen mit anderer Basis (→ Übungen)

Das b-Komplement zur Darstellung negativer Zahlen

- **Das b-Komplement ist eine Methode zur Darstellung von negativen Zahlen in einem b-basierten Zahlensystem.**
  - Für Binärzahlen (b = 2) wird dies auch als Zweierkomplement bezeichnet.
- **Funktionsweise**
  - **Zweierkomplement (Binär):** Um das Zweierkomplement einer Zahl zu bilden, invertiert man alle Bits (b -1) – z der Zahl und addiert dann 1.
  - Beispiel:
    - **Zahl +5 im Binärsystem:** 0000 0101 mit n = 8 Bit
    - **Zweierkomplement von -5:**
    - Zuerst invertieren wir +5 zu (1111 1010) und
    - Dann 1 addieren: (1111 1011).
    - Vgl. auch Vorlesung 1 Slides 15 ff.

Das b-Komplement zur Darstellung negativer Zahlen

- **Verallgemeinerung des b-Komplement zur Darstellung von negativen Zahlen**

  - **Allgemein:** $C_{b,\,n}(N) = b^n - N$,wobei
    - n = Anzahl Stellen,
    - b ist die Basis einer Stelle
    - N ist die Zahl von der das Komplent gebildet werden soll

  - Beispiel zur Verdeutlichung
  - n = 2, b = 10, N = 72
    - $C_{10,\,2}\,(72)$
    - $C_{10,\,2}(72) = 10^2 - 72 = 100 - 72 = 28$
      → Überlauf
    - + 1 ist also schon enthalten (in b^n)!

    - Aus voriger Folie: Bilde das Komplement der **Zahl +5 im Binärsystem, bei n=8 Binärstellen**
    - $C_{2,\,8}(5) = 2^8 - 5 = 256 - 5 = 251_{10} = 1111\ 1011_2$

OST

Das b-Komplement zur Darstellung negativer Zahlen

- **Anmerkungen**
  - Die bevorzugte Methode (insbesondere das Zweierkomplement für Binärzahlen) zur Darstellung negativer Zahlen in Computern
  - Ermöglicht einfache Addition und Subtraktion negativer und positiver Zahlen, einschliesslich der 0, ohne separate Logik für Vorzeichen.
  - Eliminiert die doppelte Darstellung der Null, die im (b-1)-Komplement vorhanden ist.
  - Breite Anwendung in der Computerarchitektur und digitalen Logik, insbesondere für arithmetische Operationen in ALUs.

Das b-Komplement zur Darstellung negativer Zahlen

- **Funktioniert auch in Zahlensystemen mit anderer Basis (→ Übungen)**
- **Beispiel: Berechne 12_8 - 6_8 mittels b-Komplement.**
  - Achterkomplement von 6 im Oktalsystem:
  - $C_8(6) = 8^2 - 6 = 58_{10} = 72_8$
  - Addition des Komplements:
    - $12_8 + 72_8 = 104_8 = 4_{10}$
    - Überlauf
    - $10_{10} - 6_{10} = 4$

OST

Das b-Komplement zur Darstellung negativer Zahlen

- **Verallgemeinerung des b-Komplement zur Darstellung von negativen Zahlen**

  - **Allgemein:** $C_{b,\,n}(N) = b^n - N$,wobei
    - n = Anzahl Stellen,
    - b ist die Basis einer Stelle
    - N ist die Zahl von der das Komplent gebildet werden soll

  - Beispiel zur Verdeutlichung
  - n = 2, b = 10, N = 72
    - $C_{10,\,2}(72)$
    - $C_{10,\,2}(72) = 10^2 - 72 = 100 - 72 = 28$
      Überlauf
    - + 1 ist also schon enthalten (in b^n)!

    - Aus voriger Folie: Bilde das Komplement der **Zahl +5 im Binärsystem, bei n=8 Binärstellen**
    - $C_{2,\,8}(5) = 2^8 - 5 = 256 - 5 = 251_{10} = 1111\ 1011_2$

Der **Exzesscode** oder auch **Überschuss-Code** I

Der **Exzesscode** oder auch **Überschuss-Code** (Offset-Binary, Bias-Code)

- ist eine Binärkodierung, mit der sich vorzeichenbehaftete Zahlen binär repräsentieren lassen.
- die Codierung basiert auf einer Wertebereichsverschiebung.
- Üblicherweise werden positive Zahlen im Wertebereich bis 0 bis 2^n-1 als n-stellige Binärzahlen codiert, wobei die 0 die kleinste positive darstellbar Zahl ist.
- Sollen nun auch zusätzlich negative Zahlen dargestellt werden, muss der Nullpunkt verschoben werden.

Der **Exzesscode** oder auch **Überschuss-Code** II

- Einen besonderen Stellenwert hat der Code, der den Zahlenbereich in zwei gleich grosse Hälften von
  - negativen (wenn man die 0 als negative Zahl auffasst) und
  - positiven Zahlen teilt
- Beispiele
  - Bei binär vierstelligen Codes (Dezimal 0 bis 15) würde der Exzess-7- Code also die Zahlen von -7 bis 8 repräsentieren,
    die Bias liegt bei $2^{n-1} -1$
  - Bei fünfstelligen Codes wäre es der Exzess-15-Code
    also die Zahlen -15 bis 16.
  - Bei achtstelligen Codes wäre es der Exzess-127-Code
    also die Zahlen -127 bis 128.

OST

Der **Exzesscode** oder auch **Überschuss-Code** III

- Der **Exzesscode** ist eine Binärkodierung, mit der sich vorzeichenbehaftete Zahlen binär repräsentieren lassen.
- Die Codierung basiert auf einer Wertebereichsverschiebung.

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

- Besondere Bedeutung hat hier (d.h. bei einer 3Bit-Codierung) der Exzess-3-Code, der den Zahlenbereich in zwei gleiche, positive und negative Teile aufteilt, wobei 0 als negativ betrachtet wird.

Eine andere Schreibweise

$$C_{Ex,-q,n}(x)$$

Steht für Exzess code

Steht für den Bias (neg. Überhang zur 0):

- Der niedrigste negative Wert
- Bei gleicher Aufteilung der n Bit
  Bias = $2^{n-1} - 1$
  - Zum Beispiel
  - n = 8
  - Bias = $2^{8-1} - 1 = 127$
  - Entspricht Exzess-127
  - Die kleinste Zahl ist -127 es gibt 129 positive Zahlen (0-128)

Länge der binären Schreibweise

Zu codierende Zahl

Was wäre nun folgende Codierungen (3Bit):

$C_{ex,-4,3}(2)=?$ |2-(-4)| = 6 in 3 bits => 110

$C_{ex,-4,3}(-3)=?$ |-3-(-4)| = 1 : 001

$C_{ex,-4,3}(2)=?$ Cex, -2, 3 (2) = |2-(-2)|= 4= 100

OST

Der **Exzesscode** oder auch **Überschuss-Code** IV

- Um eine Zahl *a* zu codieren, wählt man die kleinste Zahl *b* im Wertebereich und bildet die Differenz: $d = |a - b|$.
  Das Ergebnis wird dann wie üblich codiert.
- Umgekehrt decodiert man eine Exzess-N-codierte Zahl,
  indem man sie zunächst nach der üblichen Codierung in eine Zahl umwandelt und dann die kleinste Zahl des Wertebereichs addiert: $a = d + b$

Berechne die Codierung:

$C_{ex,-4,3}(-1)=?$

-4 => d = |-1-(-4)| = 3 => 011

OST

Beispiel-Rechnung I

- *Die Codelänge sei 8 bit. Codiere die Zahl -79 in der Exzess-127-Codierung oder*
- $\boldsymbol{C_{Ex-127,8}\,(-79)}$

- *D.h. der Nullpunkt liegt bei 0111 1111*

Berechnung der Codierung

- Die zu codierende Zahl ist *x = -79*.
- Die kleinste Zahl im Wertebereich ist $b = -2^{n-1}-1 = -127$
- Die Differenz ist $d = |-79 - (-127)| = 48$
- In der Standardcodierung ist $d = 48_{10} = 00110000_2$
- Damit lautet die ***Lösung***: $\boldsymbol{C_{Ex,-127,8}\,(-79)} = 00110000_{\text{Exzess-127}}$

OST

Beispiel-Rechnung II

- Die zu decodierende Zahl ist *x* = $00110000_{\text{Exzess-127}}$ *oder*
- $\boldsymbol{C_{Ex,-127,8}}$ (00110000)

Berechnung der Codierung

- Der Standardwert von *x* = $00110000_{\text{Exzess-127}}$ ergibt d = 48_{10}
- Die kleinste Zahl im Wertebereich ist $b = -2^{n-1}-1 = -127$
- Damit erfolgt der decodierte Wert zu 48+ (-127) = -79 oder

  $\boldsymbol{C_{Ex-127,8}}$ (00110000) = *-79*

Beispiel-Rechnung III

Codiere die Zahl 3 in der Exzess-3-Codierung. Wortlänge sei 3Bit

- *Mit 3Bit und 0 bei 0 ergibt sich ein Zahlenbereich von 0 bis 7*
- *Exzess-3 bedeutet 0 liegt auf der Zahl 3 also auf 011. Daraus ergibt sich der Zahlenbereich 011 bis 111 also von 0 bis 4 und von -3 bis 0*
- *oder*
- $C_{Ex,-3,3}(3) = 110_{Exzess\text{-}3}$

Berechnung der Codierung

- Die kleinste Zahl im Wertebereich ist $b = -2^{n-1}-1 = -3$
- Die Differenz ist $d = |3 - (-3)| = 6$
- In der Standardcodierung ist $d = 6_{10} = 110_2$
- Damit lautet die ***Lösung***: $C_{Ex,-3,3}(3) = 110_{\text{Exzess-3}}$

OST

Der Exzess-Code zum Vergleich der Lösung

- Der **Exzesscode** ist eine Binärkodierung, mit der sich vorzeichenbehaftete Zahlen binär repräsentieren lassen.
- Die Codierung basiert auf einer Wertebereichsverschiebung.

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

- Besondere Bedeutung hat hier der Exzess-3-Code, der den Zahlenbereich in zwei gleiche, positive und negative Teile aufteilt, wobei 0 als negativ betrachtet wird.

Beispiel-Rechnung IV

Codiere die Zahl 1 in der Exzess-3-Codierung Wortlänge sei 3Bit

- *Mit 3Bit und 0 bei 0 ergibt sich ein Zahlenbereich von 0 bis 7*
- *Exzess-3 bedeutet 0 liegt auf der Zahl 3 also auf 011 daraus ergibt sich der Zahlenbereich 011 bis 111 also von 0 bis 4 und von -3 bis 0*
- *Oder*
- $C_{Ex,-3,3}(1)$=100 Exzess-3

Berechnung der Codierung

- Die kleinste Zahl im Wertebereich ist $b = -2^{n-1}-1 = -3$

- Die Differenz ist $d = |1 - (-3)| = 4$
- In der Standardcodierung ist d = $4_{10} = 100_2$
  - Damit lautet die ***Lösung***: $C_{Ex,-3,3}(1) = 100$ Exzess-3

Übersicht zur Darstellung von Zahlen

Dezimalzahl	Betrag und Vorzeichen	Einer-Komplement	Zweier-Komplement	Exzess-8 Darstellung
-8	-----	------	1000	0000
-7	1111	1000	1001	0001
-6	1110	1001	10	0010
-5	1101	1010	011	0011
-4	1100	1011	11	0100
-3	1011	1100	101	0101
-2	1010	1 1	1110	0110
-1	1001	111	1111	0111
0	1000,0000	111 0000	0000	1000
1		0001	0001	1001
2		0010	0010	1010
3		0011	0011	1011
4	0100	0100	0100	1100
5	0101	0101	0101	1101
6	0110	0110	0110	1110
7	0111	0111	0111	1111

Übung

OST

Fixkomma - Zahlen

- **Wir haben eine weitere Darstellung der negativen Zahlen gefunden, die Exzess-Codierung, mit ihr können wir den Nullpunkt beliebig verschieben.**
- **Jetzt wollen wir noch Vor- und Nachkommastellen unterscheiden**
- **Schreibweise $C_{FK,k,n}(x)$**
- **Ähnlich: wie Exzess-Code aber:**
  - FK steht für Festkomma
  - k = Anzahl der Nach-Kommastellen
  - n = Länge der binären Schreibweise
  - Daraus ergibt sich die Anzahl der Vor-Kommastellen=n – k
  - Ist k < als die wirkliche Anzahl der Nachkomma-Stellen, entstehen Fehler
  - Wie wir die Nachkommastellen berechnen, haben sie in der letzten Vorlesung gelernt
- **Beispiel:**
  - $C_{FK,4,16}(453.1234)$=0001 1100 0101 0001 Was fällt auf?

Fehler bei den Nachkommastellen

- **Absoluter Fehler:**

  Die gerundete Zahl $X_{gerundet}$ wird
  von der nicht gerundete Zahl $X_{korrekt}$ abgezogen
  Daraus ergibt sie der Fehler

$$E_{abs} = |X_{korrekt} - X_{gerundet}|$$

- **Relativer Fehler:**

$$E_{rel} = \frac{|X_{korrekt} - X_{gerundet}|}{X_{korrekt}}$$

Fixkommazahlen - Zusammenfassung

- **Fixkommazahlen sind Binärzahlen**
- **Teilen sich auf in Vor- und Nachkommastellen**
- **Der kleinste Wert ist 2^{-k}**
- **Addition, Subtraktion und Zweierkomplement sind wie bei Ganzzahlen**
  - Addiert man Fixkommazahlen z_0+z_1 mit verschiedenen $k_0 > k_1$ so muss z_1 um k_0-k_1 Bits nach rechts geschoben werden
- **Multiplikation und Division verschieben das Komma**
  - Multipliziert man $z_0 \times z_1 = z_2$ dann ist $k_2=k_0+k_1$
- **→ k muss für jede Zahl berücksichtigt werden**
  - Wird von den meisten Programmiersprachen kaum unterstützt
  - Meist von Hand in Kommentaren
  - Limitiert die Zahlen auf Bereiche, die zur Compilezeit bekannt sind
  - → unflexibel und fehleranfällig, aber performant

Gleitkommazahlen: Motivation Beispiel Computergrafik

- Der Abstand zur Szene bestimmt die Anzahl und die Genauigkeit der dargestellten Objekte
- Beispiel Zoom-In von Erde zu Zelle
  - Am Anfang sieht man die gesamte Sonne: Durchmesser 1 392 700 km
  - Dann zoomt die Animation langsam Richtung Erdoberfläche
  - Am Ende erreicht man die Darstellung einer einzelnen Zelle: Durchmesser 25 µm
- Wenn wir als kleinste Einheit $\mathrm{u} = 100\,\mathrm{nm} = 10^{-7}\,\mathrm{m}$ verwenden, sind
  - der Zelldurchmesser: 250 u
  - der Sonnendurchmesser: $1\,392\,700 \cdot 10^{10}\,\mathrm{u} = 13.927 \cdot 10^{15}\,\mathrm{u}$, benötigt also 50+ Bit

Gleitkommazahlen

- **Bei Gleitkommazahlen wird zusätzlich zum Bitmuster z der eigentlichen Zahl auch noch die Stelle k mitgeführt, an der das Komma steht**
- **z heisst Signifikand, wird aber auch oft als Mantisse (=Nachkommateil) bezeichnet**
- **k heisst Exponent**
- **Der Wert** $C_{GK\ k,\ n}(z) = z * 2^k$

Encodings von Gleitkommazahlen

- Der Standard IEEE 754 definiert seit 1985 verschiedene Encodings
  - Single (Java `float`): 24 Bit Präzision, 8 Bit Exponent
  - Double (Java `double`): 53 Bit Präzision, 11 Bit Exponent
  - Quadruple (seit 2008): 113 Bit Präzision, 15 Bit Exponent, insb. für Zwischenergebnisse
  - Octuple (seit 2008): quasi nicht implementiert
  - (Single-Extended mit 43 Bit)
  - (Double-Extended mit 79 Bit)
  - (Encodings zur Basis 10 mit 32, 64, 128 Bit, seit 2008)

Encoding Grundlagen nach IEEE-754-Standard

- **Achtung: Gleitkommazahlen sind Näherungen**
- **Vorzeichenbit (Sign Bit):**
  - Das linkste Bit (MSB) in der Gleitkommadarstellung ist das Vorzeichenbit.
  - 0 repräsentiert positive Zahlen, und 1 repräsentiert negative Zahlen.
- **Exponent:**
  - Der Exponent repräsentiert den Exponenten der Zahl und ermöglicht die Darstellung von sehr grossen oder sehr kleinen Zahlen.
  - Der Exponent ist als Binärzahl dargestellt und verwendet einen vordefinierten Bias (Verschiebung), um sowohl positive als auch negative Exponenten darzustellen.
- **Mantisse (Signifikand oder Fraction):**
  - Die Mantisse repräsentiert den eigentlichen Wert der Zahl.
  - Sie ist als Binärzahl dargestellt und befindet sich rechts vom Binärpunkt.
  - Der Binärpunkt ist nicht explizit gespeichert, da er immer links vom ersten Bit der Mantisse liegt.
  - Wird nicht im Zweierkomplement dargestellt – Unterscheidung ausschliesslich durch Vorzeichenbit.

Gleitkommazahl Codierung

Das IEEE-Format (Institute of Electrical and Electronics Engineers) interpretiert eine bestimmte Anzahl von Bits als Dezimalzahl. Es gibt hier eine kleinere Variante mit 32 Bit und eine größere Variante mit 64 Bit.

Hier die kleinere Variante:

- **Vorzeichen:** 1 Bit
  - 1 Bit, Wert 1 bedeutet “-” ; Wert 0 bedeutet “+”
- **Mantisse, der eigentliche Wert:** 23 Bit
  - Normiert, so dass nur eine 1 vor dem Komma steht
  - diese kann also entfallen!
- **Exponent oder Charakteristik** 8 Bit
  - Kann positiv oder negativ sein
- **Total** 32 Bit

OST

Beispiel für das 32 Bit Short-Format:

0,1,..,22 mit 0 und 22 = 23 bits

- **Im 32 Bit Short-Format werden die Bits 0 bis 22 als Mantisse interpretiert, die Bits 23 bis 30 als Exponent und das Bit 31 als Vorzeichen.**

Bit 23,24,..,30 mit 23 und 30 = 8 Bits

1 Bit

insgesamt 32 Bits

- **Zum Beispiel**
  - 01101101001100101010010101011101 wird zu
  - 0 11011010 01100101010010101011101
  - Vorzeichen Exponent Mantisse

OST

Einfaches Beispiel zur Charakteristik (Exponent)

- **Angenommen wir hätten 4 Bit dann wird der Bias berechnet zu $Bias = 2^{4-1} - 1 = 7$ d.h. die 7 wird zur 0**

- **Extrapolieren wir das auf 8 Bit, so folgt:**

$$2^{8-1} - 1 = 127$$

00000001……01111111……11111110

-126 0 127

Exponent ist bei 32 Bits der Teil mit 8 Bits, deshalb ist dort der Bias 127!

OST

Beispielrechnung

wir wollen die Zahl 18.75 als binäre Gleitkomazahl (32bit) darstellen

Erinnerung: Bits 0 - 22 als Mantisse **, die Bits 23 - 30 als** Exponent **und das Bit 31 als** Vorzeichen**.**

18.75

10010.11

Normierung: hier shift nach rechts: $1.001011 * 2^4$

positive Zahl Sign: 0

Exponent: 127 + 4 = 131 = 01111111 + 00000100 = 10000011

Mantisse: 0010110….0

Exzess GK = $C_{GK\ k=22,\ n=32}(18.75)$ = 0 10000011 00101100000000000000000

OST

Addition

Die Addition basiert auf der Integer-Addition, muss aber Exponenten und Vorzeichen explizit berücksichtigen:

1. Wenn Vorzeichen unterschiedlich: Führe Subtraktion aus
2. Ergänze hidden Bits ganz links (jeweils 1)
3. Wenn Exponenten unterschiedlich:
   - Schiebe Significand der kleineren Zahl um entsprechend viele Bits nach rechts
   - (Erstes eingeschobenes Bit $= 1$ (hidden bit))
   - (Weitere eingeschobene Bits $= 0$)
4. Führe Addition durch
5. Falls Carry $= 1$: Normalisiere Ergebnis
   - Erhöhe Exponent um 1
   - Schiebe Significand um 1 nach rechts

Addition Beispiel

Erinnerung Exponent: 0 liegt bei $2^{8-1} - 1 = 127_{10} = 0111\ 1111$

1 liegt bei 1000 000

-1 liegt bei 0111 1110

- $x = 1.5,\ y = 0.75$
- $x = 0\,|\,0111\ 1111\,|\,100\ 0000\ 0000\ 0000\ 0000\ 0000$
- $y = 0\,|\,0111\ 1110\,|\,100\ 0000\ 0000\ 0000\ 0000\ 0000$
- $x' = 0\,|\,0111\ 1111\,|(1)\ 100\ 0000\ 0000\ 0000\ 0000\ 0000$ mit hidden Bit
- $y' = 0\,|\,0111\ 1111\,|(0)\ 110\ 0000\ 0000\ 0000\ 0000\ 0000$ m.h.B., $a + 1,\ m \cdot 2^{-1}$
- $z' = 0\,|\,0111\ 1111\,|(10)\ 010\ 0000\ 0000\ 0000\ 0000\ 0000 = x' + y'$
- $z'' = 0\,|\,1000\ 0000\,|(1)\ 001\ 0000\ 0000\ 0000\ 0000\ 0000\ a + 1,\ m \cdot 2^{-1}$
- $z = 0\,|\,1000\ 0000\,|\,001\ 0000\ 0000\ 0000\ 0000\ 0000$ ohne hidden Bit
- $z = 2.25$

Gleitkommazahlen - Zusammenfassung

- **Mit k=0…0 und m=0…0 stellt man ±0 dar**
- **Gleitkommazahlen sind weder $\mathbb{R}$ noch $\mathbb{Q}$ sondern limitierte Teilmengen davon**
- **Viele Zahlen können nicht präzise dargestellt werden (z.B. 0.1)**
- **Rundungsfehler können ein Ergebnis komplett unbrauchbar machen**
  - Vergleiche sollten immer über einen Schwellwert abgewickelt werden

Text-Kodierungen

- **Frage**
  - Wenn man nur Nullen und Einsen übertragen kann, wie überträgt man dann Text?

Text-Kodierungen - ASCII

- Text besteht aus einer endlichen Folge von *Zeichen*: Buchstaben, Zahlen, Satzzeichen, etc.
- Alle Zeichen stammen aus einer *endlichen* Menge Z, dem *Zeichensatz* (*character set*)

$\Rightarrow$ Jedem Zeichen kann eineindeutig eine natürliche Zahl zugeordnet werden

- Ein *Encoding* (*character encoding*, *Zeichenkodierung*) ist eine bijektive Funktion E, die jedem Zeichen z eine natürliche Zahl $E(z) < |Z|$ zuordnet

$\Rightarrow$ Text mit n Zeichen $z_0 \dots z_{n-1}$ kann als endliche Folge von kodierten Zeichen kodiert werden:

$$E'(z_0 \dots z_{n-1}) = E(z_0) \dots E(z_{n-1})$$

ASCII

- American Standard Code for Information Interchange
- 1963 von der ASA (American Standards Association, heute ANSI) für den Gebrauch in den USA veröffentlicht
- 1968 in der heutigen Form revidiert
- 1968 durch den US-Präsidenten als Standard für Computer der US-Bundesbehörden bestimmt
- Grösse des Zeichensatzes: $2^7 = 128 \Rightarrow 7$ Bit (nicht 8 Bit!)
- Enthält druckbare (darstellbare) Zeichen und (nicht darstellbare) Steuerzeichen

ASCII – Zeichentabelle

33 Steuerzeichen, viele davon obsolet (ursprünglich zur Übertragung, nicht Speicherung)

10 Ziffern: 0 - 9

33 Interpunktionszeichen: ! " # ...

26 Grossbuchstaben: A - Z

26 Kleinbuchstaben: a - z

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH 1	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SB	ESC	FS	GS	RS	US
2	␣	!	"	#	$	%	&	’	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

OST

ASCII - Steuerzeichen

- 00_h, `\0`: Null, als Marker für das Ende des Textes bzw. Strings
  Beachte: `'\0'` $\neq$ `'0'` (ASCII $00_h \neq 30_h$)
- 08_h, `\b`: Backspace
- 09_h, `\t`: Tabulator
- $0A_h$, `\n`: Line Feed, Newline, eine Zeile «füttern», d.h. Cursor eine Zeile nach unten
- $0D_h$, `\r`: Carriage Return, Cursor (= Carriage) an den Anfang der Zeile
- Interpretation dieser Zeichen hängt von Betriebssystem, Programmiersprache und/oder Laufzeitumgebung ab

Alternativen zu ASCII

- ASCII kann viele Zeichen nicht kodieren
- Alternative: Komplette Neudefinition des Encodings *inkompatibel* zu ASCII
- z.B. IBMs EBCDIC (Extended BCD Interchange Code) von 1964
  - 8-Bit-Encoding kompatibel zu Binary Coded Decimals und IBMs Lochkarten
  - Buchstaben sind nicht fortlaufend kodiert

  $\Rightarrow$ Text ist schwieriger zu sortieren
  - Hat sich ausserhalb IBMs nicht durchgesetzt

Lernziele

- **Sie verstehen die Idee einer mathematischen Struktur**
- **Sie kennen den Unterschied zwischen Gruppe, Ring und Körper**
- **Sie können den Begriff der Abgeschlossenheit und den Zusammenhang zur Modulo-Rechnung erklären**
- **Sie beherrschen die Modulo-Rechnung im $\mathbb{Z}_2$**
- **Sie verstehen, dass man eine Bitfolge als Menge, Tuppel, Polynom oder Vektor betrachten kann und können einfache Berechnungen im $\mathbb{Z}_2$ durchführen**
- **Sie verstehen die Idee der zyklischen Gruppe und können einfache Berechnungen zur Körpererweiterung durchführen und interpretieren**

Eine zyklische Gruppe ist eine Gruppe, in der es ein Element gibt, aus dem man durch wiederholte Anwendung der Gruppenoperation alle anderen Elemente der Gruppe erhält.

Eine zyklische Gruppe ist eine spezielle Art von Gruppe, bei der alle Elemente der Gruppe durch wiederholte Anwendung der Gruppenoperation auf ein einziges Element erzeugt werden können. Dieses besondere Element nennt man Erzeuger oder Generator der Gruppe.

Gruppe: Eine Gruppe ist eine Menge mit einer Verknüpfung (z. B. Addition oder Multiplikation), für die bestimmte Regeln gelten (Abgeschlossenheit, neutrales Element, Inverses, Assoziativität).

Zyklisch: „Zyklisch" bedeutet in diesem Zusammenhang, dass die ganze Gruppe „aus einem einzigen Element heraus" aufgebaut werden kann.

oder Eine zyklische Gruppe ist eine Gruppe, in der jedes Element durch wiederholtes Anwenden der Gruppenoperation auf ein bestimmtes Element (den sogenannten Erzeuger) entsteht.

OST

Eine nicht mathemaische Einführung in die Gruppentheorie
Was ist eine Gruppe, Ring, Körper?

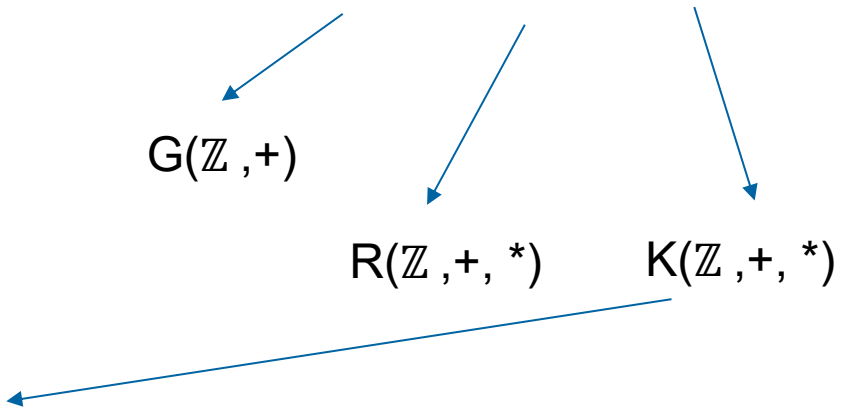
- Algebraische Strukturen und elementares Rechnen

neutrales Element
bei + : 0
bei * = 1
Inverses Element
a + a' = 0, z.B. 1 + (-1) 0
a * a' = 1, z.B. 4 * 1/4 = 1

Mengen | Rechenoperationen | Gruppe / Ring / Körper

$\mathbb{Z} = \{.., -2, -1, 0, 1, 2, ..\}$ +

$\mathbb{Z}$: Ganze Zahlen



- ein Ring ist ein Körper (engl. field), wenn
  a) **jedes** Element des Rings ausser der Null ein (multiplikatives) Inverses hat,
  b) die Multiplikation kommutativ ist: ab = ba (abel'sche Gruppe)
  c) das Distributivgesetz gilt: a(1-b) = a – ab
  d) wenn er eine 1 hat. (multiplikatives neutrales Element)

OST

Wir "basteln" uns eine algebraische Struktur

Mengen Rechenoperationen Körper

Wir definieren Mengen

$\mathbb{Z}_2 = \{0, 1\}$
$\mathbb{Z}_5 = \{0, 1, 2,3,4\}$
$\mathbb{Z}_M = \{0, 1, 2, .., M - 1\}$

Wir definieren Rechenoperationen

- Addition +
- Subtraktion –
- Multiplikation *
- Ganzzahldivision d.h., es entsteht ein Rest (Restklassen)
- Um die Abgeschlossenheit sicherzustellen, werden alle Operationen mit der Operation Modulo M durchgeführt

OST

Was heisst Modulo

- **Wenn wir, nun die Multiplikation durchführen und die Abgeschlossenheit im $\mathbb{Z}_5 = \{0, 1, 2,3,4\}$ haben wollen, müssen wir die Modulo-Operation durchführen!**
- **Allgemein Division mit Rest**
  - $n = a \times m + b, \quad 0 \leq b < |m|$
    - a=Quotient, m=Divisor, b=Rest
- **Modulo**
  - $n \, mod \, m = b$
- **Beispiel:**
  - sei a = 3 und b = 4
  - Dann ergibt $a \cdot b = 12$, damit verlassen wir aber $\mathbb{Z}_5$, d.h. wir sind nicht abgeschlossen
  - Die Modulo Operation definiert nun, wie wir das Ergebnis $a \cdot b = 12$ wieder auf eine Zahl im $\mathbb{Z}_5$ abgebildet wird

12 mod 5 = 2 (da $12 = 2 \times 5 + 2$)

Modulo Rechnung Beispiel 1

- **Beispiele (für z = x mod y)**
- **Es seien x beliebig, positiv und ganzzahlig, y = 2 .**
- **Unsere Ergebnismenge Z ist dann {0,1},**
  d.h. wir stellen sicher, dass wir im $\mathbb{Z}_2$ sind bzw. bleiben
- **0 mod 2 = 0, da 0 < 2**
- **1 mod 2 = 1, da 1 < 2**
- **5 mod 2 = 1, da 1 = 5 – 2 ∗ 2**
- **Interpretation:**
  Der Zahlenraum wird also auf die Menge der Zeichen {0, 1} begrenzt.
  Andere Zeichen sind nicht darstellbar.

Modulo Rechnung Beispiel 2

- **Beispiele (für z = x mod y)**
- **Es seien x beliebig, positiv und ganzzahlig, y = 8 .**
- **Unsere Ergebnismenge Z ist dann in {0,1,2,3,4,5,7}, d.h. im $\mathbb{Z}_8$**
- **0 mod 8 = 0, da 0 < 8**
- **7 mod 8 = 7, da 1 < 8**
- **35 mod 8 = 3, da 1 = 35 – 4 ∗ 8**

Interpretation

- **Der Zahlenraum wird also auf die Zeichen {0, .. ,7} begrenzt**

Modulo Rechnung

- **Ist eine wichtige Rechenoperation in der Codierung**
- **Die Modulo-Funktion begrenzt einen Zahlenbereich**
- **z.B. haben wir im $\mathbb{Z}_2$ gesehen,**
  dass nur die Menge der Ziffern {0,1} existiert
- **im $\mathbb{Z}_8$ existiert nur die Menge der Ziffern {0,1, 2, 3, 4, 5, 6, 7}**
- **Um im vorgegebenen Bereich zu bleiben, verwenden wir die Modulo-Operation. Dabei berechnen wir den Rest z, der sich ergibt, wenn man x durch y teilt. Konkret bedeutet das, wir subtrahieren y so lange von x, bis das Ergebnis z kleiner als y ist.**

Modulo Rechnung negativ

- **Was passiert, wenn x negativ ist, aber wir z.B. im $\mathbb{Z}_2$ bleiben müssen**
- **Auch hier setzen wir die Modulo Operation ein**
- **z = -x mod y für die beiden ganzen Zahlen x und y den Rest z,**
  - Bei positivem x haben wir definiert:
    - Um im vorgegebenen Bereich zu bleiben, verwenden wir die Modulo-Operation. Dabei berechnen wir den Rest z, der sich ergibt, wenn man x durch y teilt. Konkret bedeutet das, wir subtrahieren y so lange von x, bis das Ergebnis z kleiner als y ist.
  - Jetzt mit negativem x:

Modulo Rechnung negativ Beispiel 1

- **Beispiele (für z = x mod y)**
- **Es seien x beliebig, ganz negativ und ganzzahlig, y = 2 .**
- **Unsere Ergebnismenge Z ist dann {0,1}, d.h. wir stellen sicher, dass wir im $\mathbb{Z}_2$ sind**
- **0 mod 2 = 0, da 0 < 2**
- **-1 mod 2 = 1, da 1 = -1 + 2**
- **-5 mod 2 = 1, da 1 = -5 + 3 ∗ 2**
- **Interpretation:**
  Der Zahlenraum wird also auf die Zeichen 1 und 0 begrenzt.
  Andere Zeichen sind nicht darstellbar.

Modulo Rechnung Modul negativ

- **Was passiert, wenn y negativ ist,**

- **Auch hier können wir die Modulo Operation einsetzen**
- **Aber jetzt beschreiben wir einen anderen Zahlenraum**
  z = x mod -y
- **Mit –y = -2 kommen wir zu Z_{-2} = {0, -1}**

- **x, y können auch beliebige Dezimalwerte annehmen, diese Überlegungen sowie negative y spielen für die Codierung jedoch keine Rolle und werden daher hier nicht weiter betrachtet.**

Interpretation eines Codeworts: Der Ganzzahlenkörper

- **In der Informatik werden alle Informationen in sogenannten Codewörtern abgelegt.**
- **Codewörter wiederum können als Elemente eines endlichen Ganzzahlenkörpers betrachtet werden**
- **Die Anzahl der darstellbaren Codewörter wird durch die Codewortlänge bestimmt.**
  - Ein Byte besteht aus 8 Bit,
  - ein Word besteht aus 16, 32 Bit oder 64 Bit,
  - ein TCP-Paket besteht maximal aus 1024 Bit.

Interpretation eines Codeworts: Unendlichkeit

- **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 unserem Modell.**
- **Praktisch gesehen kommen wir diesem Begriff aber schon sehr nahe. Nehmen wir z.B. ein TCP/IP Paket der Länge 1024 Bit. Mit einem derartigen Paket sind ca. 10^{310} verschiedene Codewörter darstellbar. Das scheint auf den ersten Blick gar nicht so viel zu sein, aber bedenken sie, dass die Anzahl aller im Universum beobachtbarer Atome auf „nur“ 10^{84} bis 10^{89} Atome geschätzt wird (ohne schwarze Materie).**

Interpretation eines Codeworts

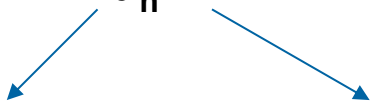
- **Das Codewort 1001 kann betrachtet werden**
  - Als geordnetes Tupel (1,0,0,1)
  - Als Zahl $1001_2 = 9_{10}$. Es gelten die üblichen Operationen der Ganzzahlrechnung.
  - Als Vektor: z.B. $[1,0,0,1]^T$. Es gelten die üblichen Operationen der Vektorrechnung.
  - Als Polynom: z.B. $g(u) = u^3 + u^0$. Es gelten die üblichen Operationen der Polynomrechnung.
- **Die drei oben vorgestellten Darstellungsformen sind äquivalent und beschreiben im Beispiel alle das gleiche Codewort.**
- **Alle Berechnungen erfolgen im $\mathbb{Z}_2$**

Interpretation eines Codeworts als Tupel

- **gegeben sei das Tupel (1,0,0,1, …. 0,1,1,1,0)**
- **das könnte ein empfangenes Codewort sein.**
- **In der Interpretation könnte das jedoch wieder aus dem Tupel (Zieladresse, Herkunftsadresse, CRC, Daten) bestehen,**
- **wobei jedes Tupel eine definierte Länge hat**

Interpretation eines Codeworts als Zahl

- **Beispiel Dualsystem**
- **Allgemein: $N = d_n R^n + \ldots + d_1 R^1 + d_0 R^0$**



Ziffer $\in Z_2$ **Wertigkeit**

- **$N_2 = 110$**
  - $N_2 = 1 * 2^2 + 1 * 2^1 + 0 * 2^0$ Dezimal: 6
- **oder**
- **$N_2 = 101$**
  - $N_2 = 1 * 2^2 + 0 * 2^1 + 1 * 2^0$ Dezimal: 5

OST

Interpretation eines Codeworts als Vektor

- **Allgemein können wir in $\mathbb{R}$ unserer Grösse neben dem Betrag auch eine Richtung geben.**
- **Zum Beispiel in der Fläche $\mathbb{R}^2$ oder im Raum $\mathbb{R}^3$**
- **Befinden wir uns im $\mathbb{Z}_2$ würde ein Vektoraddition im $\mathbb{Z}_2^3$ folgendermassen aussehen:**

$$\begin{pmatrix}1\\1\\0\end{pmatrix}+\begin{pmatrix}0\\0\\1\end{pmatrix}+\begin{pmatrix}0\\0\\1\end{pmatrix}\equiv\begin{pmatrix}1\\1\\0\end{pmatrix} mod\ 2$$

- **Das benötigen wir in der Codierung zur Fehlererkennung und -Behebung**

Interpretation eines Codeworts als Polynom

- **Sei das Codewort [100101] gegeben, dann würde das zugehörig Polynom wie folgt aussehen:**
  - $1u^5 + 0u^4 + 0u^3 + 1u^2 + 0u^1 + 1u^0 = u^5 + u^2 + u^0$
  - Ein Polynom 5-ten Grades mit Koeffizienten aus Z_2
  - u ist nicht 2, was bedeutet das? Diskutieren Sie!

    u^n gibt Position im Codewort an

- **Die Multiplikation zweier Polynome mod 2, wäre nun wie folgt:**
- **$(u^5 + u^2 + u^0)\,(u^2 + u^0) \bmod 2 = (u^7 + u^4 + u^2 + u^5 + u^2 + u^0) \bmod 2$**

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

mit $u^0 = 1$ folgt **$= u^7 + u^5 + u^4 + 1$**

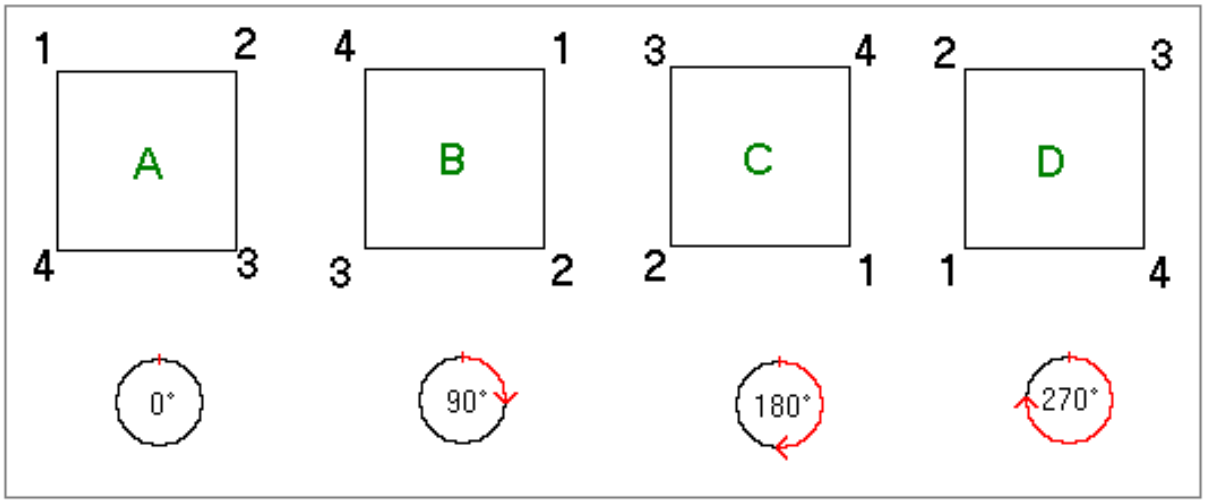
und das neue Codewort ergibt [1011001]

OST

Was ist eine 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.

Veranschaulichung:



Quelle: https://de.wikipedia.org/wiki/Zyklische_Gruppe

OST

Lösung des Polynoms $F(x) = x^3 + x + 1$ im $\mathbb{Z}_2$

- **Betrachten wir das prime (irreduzible) Polynom:**
  - $F(x) = x^3 + x + 1$
  - Nach dem Hauptsatz der Algebra hat jedes Polynom so viele Nullstellen, wie durch die höchste Potenz angezeigt sind, hier 3 Nullstellen.
  - Hat diese Polynom in $\mathbb{Z}_2 = \{0, 1\}$ eine Lösung?
  - Hier kommt dem französischen Mathematiker Évariste Galois (1811- 1832) eine geniale Idee: die **zyklischen Gruppe**

OST

Lösung des Polynoms $F(x) = x^3 + x + 1$ im $\mathbb{Z}_2$

eine **zyklische Gruppe**,

- wird von einem einzelnen Element erzeugt, wie bereits oben gezeigt
- besteht nur aus Potenzen des Erzeugers.
- Galois setzt a als Lösung in $F(x) = x^3 + x + 1$ ein, und definiert :

$F(a) = a^3 + a + 1 = 0$, ***das erzeugende Element*** **a**

F(x)=0 hat oben keine Lösung
Galois hat Körper erweitert mit a, so dass es eine Nullstelle gibt bei x^3 +x + 1
a ist wie eine komplexe Zahl

a ist definiert als : a^3+a+1 = 0 in Z2 (mod 2)

OST

Beispiel

- **Es sei $F(a) = a^3 + a + 1 = 0$,**
  - Dann können wir zunächst festhalten
  - a = a
  - $a^2 = a^2$ aber
  - $a^3 = a+1$ = -a-1 mod 2 (damit a3+a+1 = 0 stimmt)
  - $a^4 = a(a+1) = a^2 + a$
  - $a^5 = a(a^2 + a) = a^3 + a^2 = a^2 + a+1$
  - $a^6 = a(a^2 + a +1) = a^3 + a^2 + a = a+1 + a^2 + a = a^2 + 1$
  - $a^7 = a(a^2 + 1) = a^3 + a = a+1+ a = 1$
  - a^8 = a : der Zyklus beginnt von vorne!

2*a^3 = 0 wegen mod 2, aber a^3 ist nicht 0!

OST

Beispiel

- *Damit entsteht aus $\mathbb{Z}_2 = \{0, 1\}$ der Erweiterungskörper*
- $\{\mathbf{0}, \mathbf{1}, \boldsymbol{a}, \mathbf{a^2}, \boldsymbol{a}\mathbf{+1}, \mathbf{a^2 + a}, \mathbf{a^2 +}\ \boldsymbol{a}\mathbf{+1}, \boldsymbol{a}^{\mathbf{2}}\mathbf{+ 1}\}$
- *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

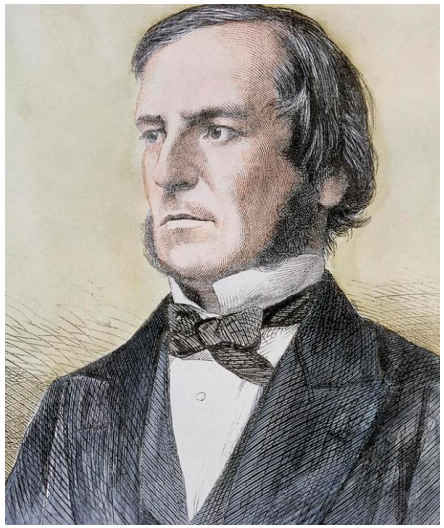
Ziel der Vorlesung

- **Sie verstehen, dass die boolesche Algebra eine mathematische Struktur ist, die auf Logik und Mengenlehre basiert**
- **Sie kennen und verstehen die in der booleschen Algebra definierten Funktionen**
- **Sie können mit Hilfe der booleschen Algebra einfache logische Strukturen entwerfen**
- **Sie können einfache algebraischen Ausdrücke mit Hilfe der booleschen Algebra vereinfachen**
- **Sie kennen einfache Regeln und Verfahren, die in der booleschen Algebra Anwendung finden (De Morgan, KV-Diagramme)**
- **Sie können einfache Vereinfachung durchführen**
- **Sie können die boolesche Algebra nutzen, um einfache System zu entwerfen und optimieren**

Inhalt

- **Einführung und Motivation**
- **Definition der Algebra**
- **Die Funktionen der Schaltungsalgebra**
- **Das Kanonische disjunktive Normalform (KDNF)**
- **Karnaugh-Veitch-Diagramm oder KV-Diagramm**

- **Unsere heutigen Computer arbeiten intern mit Bitfolgen.**
- **Die Rechenvorschriften (den Kalkül) wurden schon 1847 von dem englischen Mathematiker George Boole in dessen Logikkalkül entwickelt.**
- **Er interessierte sich für den Umgang mit den Wahrheitswerten falsch und wahr.**
- **Es zeigte sich, dass der Umgang mit den Wahrheitswerten genau denselben Gesetzen folgt, wie der Umgang mit 0 und 1 im Computer.**
- **Daher kann man heute 0 beziehungsweise falsch sowie 1 (oder besser > 0) und wahr synonym benutzen.**
- **Aus technischer Sicht benutzt man den Namen Boolesche Algebra, und als Logiker eher den Namen Aussagenlogik.**



George Boole 1815-1864

- **Aussagen sind formulierte Feststellungen, zum Beispiel:**
  - „Tür geschlossen“
  - Bedingungen, zum Beispiel x > 5
  - Relationen, zum Beispiel a[i] < a[i+1]
  - «berechnete» Aussagen, wie z.B. ‘2024 ist ein Schaltjahr’

- **Aussagen;**
  - Können den Ablauf eines Programms beeinflussen
  - Können logisch verknüpft werden und ergeben neue Aussagen
  - Sind möglicherweise Eingabe oder Ausgabe von Programmen

Einführung und Motivation: Übersicht

- **Boolesche Logik**
  - Teilgebiet der Algebra, welches sich mit der Manipulation von zwei Werten befasst: Wahr und Falsch
  - Grundlage der digitalen Welt und ermöglichen die Darstellung und Manipulation von Daten in Computern
- **Logische Operationen**
  - Operationen, um boolesche Werte zu verarbeiten
  - grundlegende logische Operationen: UND (AND), ODER (OR), NICHT (NOT)
  - ermöglichen es, komplexe logische Ausdrücke zu erstellen und Entscheidungen basierend auf Eingangsdaten zu treffen
- **Digitale Codierung**
  - Boolesche Logik legt die Grundlage, um Daten zu codieren und digital zu verarbeiten
  - Kombination der Basisoperationen der booleschen Logik
    - AND/OR/NOT → NAND → Flip-Flop/Register/RAM → ALU → Computer
      https://nandgame.com/

Aufbau der Algebra

Menge der Zeichen:

- $Z_{Boolean}$ = {wahr, falsch} oder
- $Z_{Boolean}$ = {0, {1,2,..∞}}
  0 steht für falsch

Die Axiome (nächste Folie) und Funktionen:

- ∧ **Und**
- ∨ **Oder**
- ¬ **Negation**

- Boolesche Algebra

  ({0,1}, ∧, ∨, ¬)

OST

Axiome der Booleschen Algebra nach dem Mathematiker Peano

Name	Gesetz	Duales Gesetz
Kommutativgesetze	$x \wedge y = y \wedge x$	$x \vee y = y \vee x$
Assoziativgesetze	$x \wedge (y \wedge z) = (x \wedge y) \wedge z$	$x \vee (y \vee z) = (x \vee y) \vee z$
Idempotenzgesetze	$x \wedge x = x$	$x \vee x = x$
Distributivgesetze	$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$	$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$
Neutralitätsgesetze	$x \wedge 1 = x$	$x \vee 0 = x$
Extremalgesetze	$x \wedge 0 = 0$	$x \vee 1 = 1$
Doppelnegation (Involution)	$\neg\neg x = x$	
De Morgansche Gesetze	$\neg(x \wedge y) = \neg x \vee \neg y$	$\neg(x \vee y) = \neg x \wedge \neg y$
Komplementärgesetze	$x \wedge \neg x = 0$	$x \vee \neg x = 1$
Dualitätsgesetze	$\neg 0 = 1$	$\neg 1 = 0$
Absorptionsgesetze	$x \vee (x \wedge y) = x$	$x \wedge (x \vee y) = x$

Es fällt auf dass die Gesetze in der rechten Spalte dadurch entstehen, dass man im entsprechenden Gesetz in der linken Spalte

- ∧ mit ∨ vertauscht, und gleichzeitig **0** mit **1** vertauscht.
- Das nennt man die Dualität in der Booleschen Algebra.
- Die Dualität bedeutet, dass man auch in allen abgeleiteten Gesetzen die Vertauschungen machen kann
- Die Gültigkeit dieser Gesetze kann man leicht mit Hilfe der Wahrheitstabellen nachrechnen. → versuchen Sie das mal bei den Apsorptionsgesetzen

De Morgan‘sche Gesetze

- **Theorem 1: Die Negation einer Konjunktion ist die Disjunktion der Negationen.**
  - Algebraische Form für Theorem 1:

$$\neg(A \wedge B) = \neg A \vee \neg B$$

- **Theorem 2: Die Negation einer Disjunktion ist die Konjunktion der Negationen.**
  - Algebraische Form für Theorem 2:

$$\neg(A \vee B) = \neg A \wedge \neg B$$

- **So lassen sich ODER zu UND umformen (und umgekehrt)**

Logische Funktion Funktion f von n Bits auf 1 Bit:

$$f : \mathbb{B}^n \to \mathbb{B}$$

Parameter Variable (Platzhalter) zur Übergabe von Werten an eine Funktion

Argument Wert eines Parameters bei einer konkreten Verwendung der Funktion

Beispiel für Grundbegriffe der Logik

$$f(x, y) := x \wedge g(x, y)$$

- f und g sind *Logische Funktionen* von 2 Bit auf 1 Bit:

$$f, g : \mathbb{B}^2 \to \mathbb{B}$$

- x und y sind *Parameter* von f
- x und y sind *Argumente* für die Verwendung von g.

Arität von Funktionen

Arität: Stelligkeit

Unäre Funktion	Funktion mit einem Parameter (einstellig, *unary function*) Beispiel: $f(x) = x$
Binäre Funktion	Funktion mit zwei Parametern (zweistellig, *binary function*) Beispiel: $f(x, y) = x \wedge y$
Ternäre Funktion	Funktion mit drei Parametern (dreistellig, *ternary function*) Beispiel: $f(x, y, z) = x \wedge y \wedge z$
n-äre Funktion	Funktion mit n Parametern n-stellig, *n-ary function*) Beispiel: $f(x_0, \ldots, x_{n-1}) = x_0 \wedge x_1 \wedge \ldots \wedge x_{n-1}$
Nulläre Funktion	Funktion mit null Parametern (nullstellig, *nullary function*) Beispiel: $f() = 1$

Wahrheitstabellen

Wegen der beschränkten Anzahl an Argumenten ist es üblich, Funktionen bis zu 3 (seltener 4) Parametern in Wahrheitstabellen darzustellen.

Unäre Funktion:

x	$f(x)$
0	1
1	0

Binäre Funktion:

x	y	$f(x,y)$
0	0	0
0	1	1
1	0	0
1	1	0

- Welche Funktionen verbergen sich hinter den beiden Wahrheitstabellen?

Unäre Funktionen

- Es gibt genau 4 unäre Funktionen:

x	$F(x)$	$\mathrm{id}(x)$	$\mathrm{not}(x)$	$T(x)$
0	0	0	1	1
1	0	1	0	1

- Nur id (Identität) und not (Negation) hängen echt von x ab
- F und T sind eigentlich nulläre Funktionen F: False, T: True
- Man schreibt statt F auch 0, statt T auch 1, statt $\mathrm{id}(x)$ x
- Statt $\mathrm{not}(x)$ schreibt man $\overline{x}$ (oder $\neg x$, $-x$ oder $\sim x$).
- Aus der Tabelle leicht ersichtlich ist die doppelte Negation $\overline{\overline{x}} = x$

Disjunktion und Konjunktion

Die beiden binären Basisfunktionen sind

Disjunktion $x \vee y$ (Inklusives Oder, OR)

Konjunktion $x \wedge y$ (Und, AND, oft ohne Symbol: $xy = x \wedge y$)

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

Binäre Funktionen

Es gibt genau 16 binäre Funktionen:

x	y	0	xy	$x\overline{y}$	x	$\overline{x}y$	y	$x\overline{y} \vee \overline{x}y$	$x \vee y$	$\overline{x \vee y}$	$xy \vee \overline{x}\overline{y}$	$\overline{y}$	$x \vee \overline{y}$	$\overline{x}$	$\overline{x} \vee y$	$\overline{xy}$	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

- Nur 10 Funktionen hängen echt von x und y ab
- 4 sind unär (x, y, $\overline{x}$ und $\overline{y}$) und 2 nullär (0 und 1)
- Alle Funktionen können mittels Negation, Konjunktion und Disjunktion dargestellt werden

Binäre Funktionen – einige haben spezielle Namen

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

x	y	0	AND	$x\bar{y}$	x	$\bar{x}y$	y	XOR	OR	NOR	EQUIV	$\bar{y}$	$y \nearrow x$	$\bar{x}$	$x \nearrow y$	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

OST

NAND

$x \mid y = \overline{x \wedge y} = \overline{xy}$ heisst NAND (NOT AND, «nicht alle wahr»):

Nur dann 0, wenn $x = 1$ und $y = 1$.

x	y	$x \wedge y$	$x \mid y$	$\overline{x} \wedge \overline{y}$
0	0	0	1	1
0	1	0	1	0
1	0	0	1	0
1	1	1	0	0

Das Symbol $\mid$ heisst in diesem Kontext «Shefferscher Strich» oder «*Sheffer stroke*».

NAND sind die Grundbausteine der Computertechnik

Die Basisoperationen können ausschliesslich aus $\mid$ dargestellt werden:

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

NAND-Bausteine sind technisch leicht als Transistoren zu realisieren.

NOR

Analog zu NAND können auch alle logischen Funktionen nur mit NOR ($\overline{x \vee y}$) dargestellt werden.

Nur dann 1, wenn $x = 0$ und $y = 0$.

x	y	$x \vee y$	$\overline{x \vee y}$	$\overline{x} \vee \overline{y}$
0	0	0	1	1
0	1	1	0	1
1	0	1	0	1
1	1	1	0	0

Exklusive Disjunktion

$x \oplus y$ heisst Exklusive Disjunktion (Exklusives Oder, XOR):

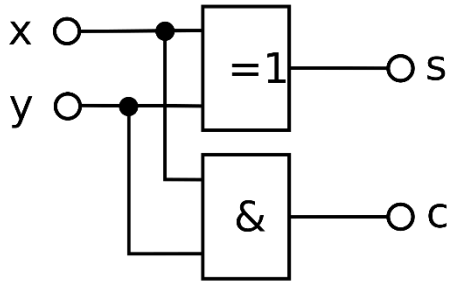
x	y	$x \oplus y$	$x \leftrightarrow y$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

- Entweder $x = 1$ oder $y = 1$ aber nicht beide gleichzeitig.
- Negation der Äquivalenz: $x \oplus y = \overline{x \leftrightarrow y}$

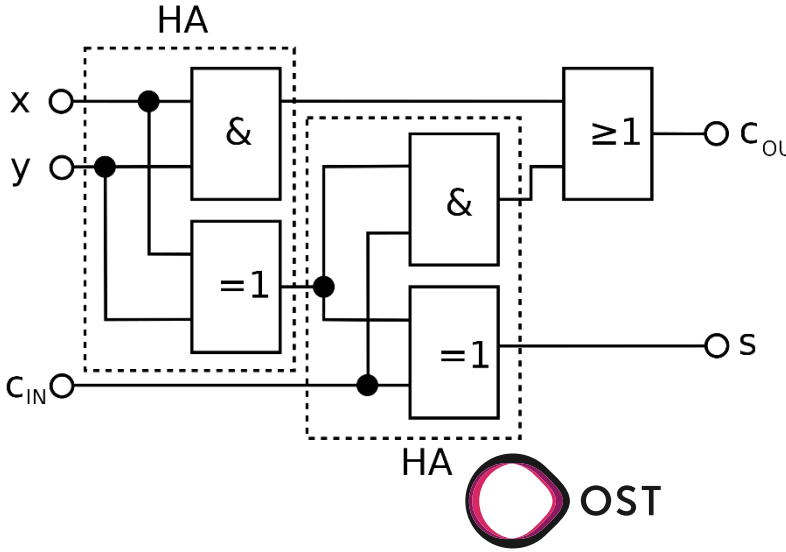
Wichtige logische Funktionen: Basis der Addition

XOR bildet die Addition zweier Bits ab, AND den Übertrag:

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



- **Halbaddierer (half adder)**
  - Kann 2 einstellige Binärzahlen addieren
  - Hat 2 Eingänge (x, y) und 2 Ausgänge (s=Summe x ⊕ y, c=Übertrag x ∧ y)
- **Volladdierer**
  - Hat 3 Eingänge (x, y, c_{IN})
- **Paralleladdierer, Serienaddierer**
  - → Übungen



OST

Terme

Literal	Variable oder Negation einer Variablen, bspw. x_3 (positives Literal) oder $\overline{x_1}$ (negatives Literal)
Konjunktionsterm	Konjunktion von Literalen, bspw. $\overline{x_1}x_3x_4 = \overline{x_1} \wedge x_3 \wedge x_4$
Disjunktionsterm	Disjunktion von Literalen, bspw. $\overline{x_1} \vee x_3 \vee x_4$
Minterm	Konjunktionsterm, der *alle* Parameter der Funktion enthält, z.B. $x_0\overline{x_1}\overline{x_2}x_3x_4$
Maxterm	Disjunktionsterm, der *alle* Parameter der Funktion enthält, z.B. $x_0 \vee \overline{x_1} \vee \overline{x_2} \vee x_3 \vee x_4$

Disjunktive Normalformen

- Eine disjunktive Normalform (DNF) ist eine Disjunktion von Konjunktionstermen, bspw.

$$\overline{x_1}x_3x_4 \vee x_1x_3x_4 = x_3x_4$$

- Funktionen werden oft als DNF dargestellt, weil sie dann nur die drei «üblichen» Operatoren $\vee$, $\wedge$ und $\neg$ verwenden
- Die kanonische DNF (KDNF) ist die Disjunktion der Minterme, bspw.

$$x_0\overline{x_1}\,\overline{x_2}x_3x_4 \vee \overline{x_0}x_1\overline{x_2}x_3x_4$$

Vereinfachung der KDNF

x	y	$x \mid y$	$x \oplus y$
0	0	1	0
0	1	1	1
1	0	1	1
1	1	0	0

- In der KDNF kann man oft Terme zusammenfassen, wenn es gemeinsame Literale gibt, und eine vereinfachte DNF erzeugen:

$$x|y = \overline{x}\overline{y} \vee \overline{x}y \vee x\overline{y} = \overline{x}(\overline{y} \vee y) \vee x\overline{y} = \overline{x} \vee x\overline{y} = \overline{x} \vee \overline{y}$$

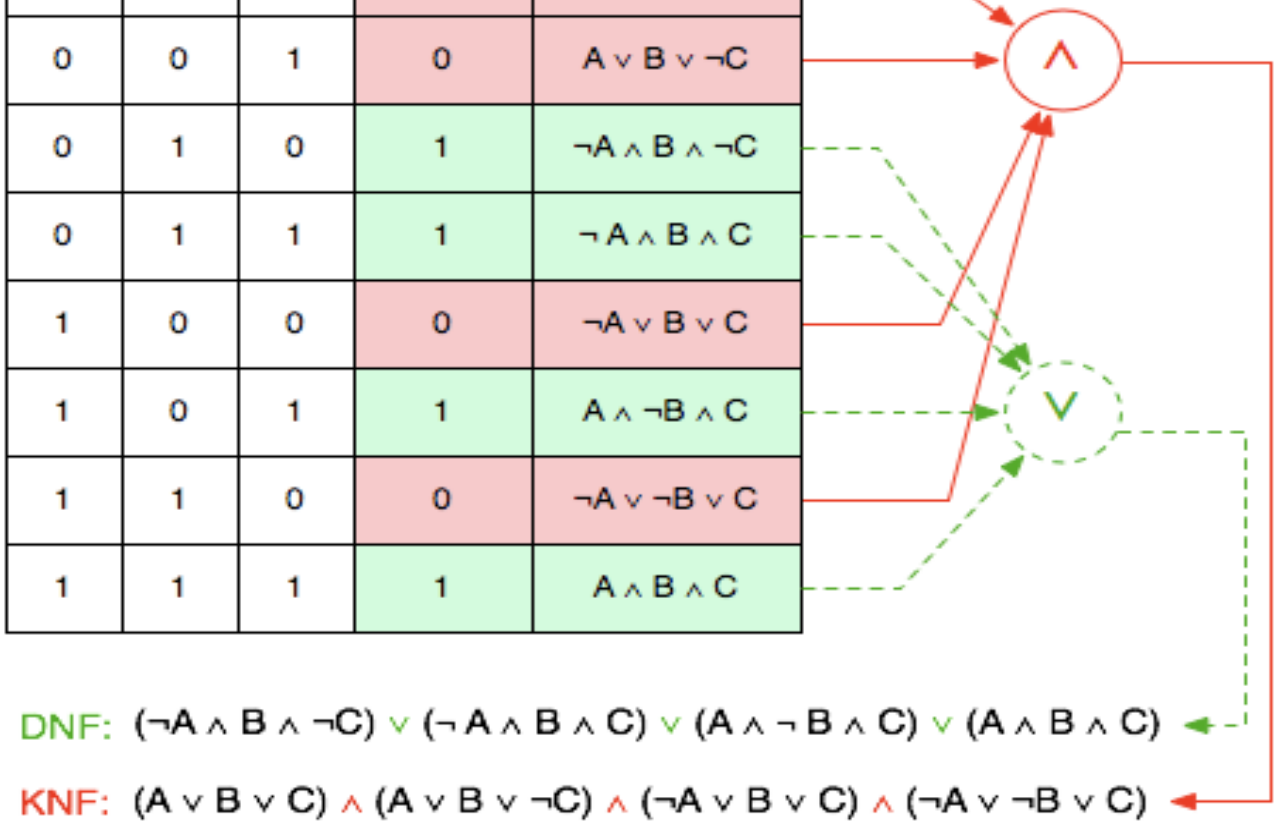
- . . . aber nicht immer:

$$x \oplus y = \overline{x}y \vee x\overline{y}$$

- Hilfreich, um komplizierte `boolean`-Ausdrücke in Programmen zu vereinfachen: Komplizierter Ausdruck $\rightarrow$ Wahrheitstabelle $\rightarrow$ KDNF $\rightarrow$ DNF
- Für die Optimierung grosser Schaltungen gibt es algorithmische Verfahren, z.B. Quine-McCluskey

Beispiel für Bildung der DNF (Disjunktive Normal Form)

A	B	C	Ergebnis	Klausel
0	0	0	0	$A \vee B \vee C$
0	0	1	0	$A \vee B \vee \neg C$
0	1	0	1	$\neg A \wedge B \wedge \neg C$
0	1	1	1	$\neg A \wedge B \wedge C$
1	0	0	0	$\neg A \vee B \vee C$
1	0	1	1	$A \wedge \neg B \wedge C$
1	1	0	0	$\neg A \vee \neg B \vee C$
1	1	1	1	$A \wedge B \wedge C$



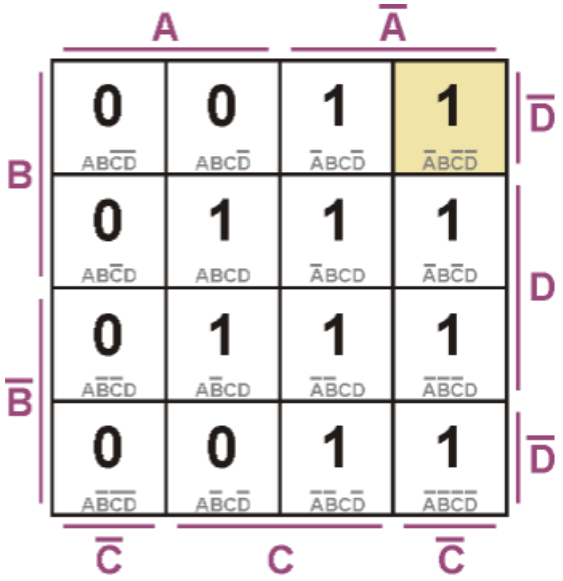
DNF: $(\neg A \wedge B \wedge \neg C) \vee (\neg A \wedge B \wedge C) \vee (A \wedge \neg B \wedge C) \vee (A \wedge B \wedge C)$

KNF: $(A \vee B \vee C) \wedge (A \vee B \vee \neg C) \wedge (\neg A \vee B \vee C) \wedge (\neg A \vee \neg B \vee C)$

OST

Karnaugh-Plan oder auch KV-Diagramm erstellen

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1



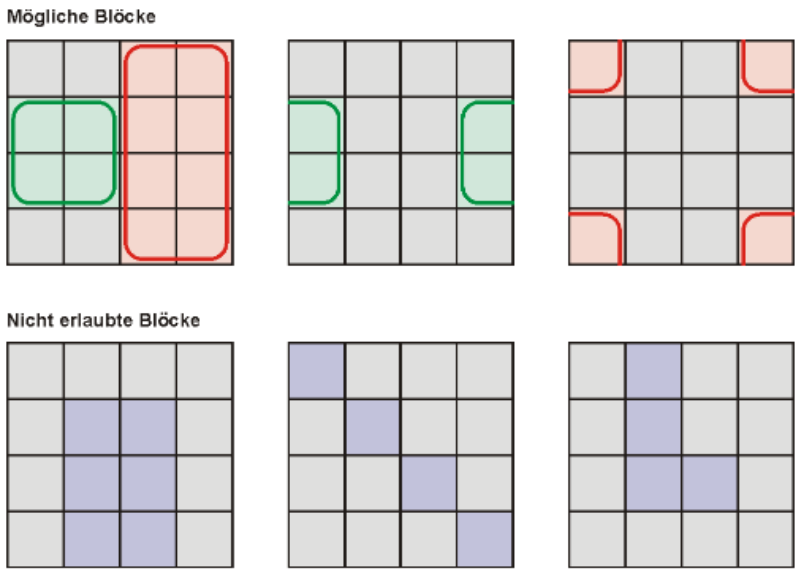
Ohne Vereinfachung würde die Logikgleichung aus der Tabelle wie folgt lauten:

$$Y = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}C\overline{D} + \overline{A}\overline{B}CD + \overline{A}B\overline{C}\overline{D} + \overline{A}B\overline{C}D + \overline{A}BC\overline{D} + \overline{A}BCD + A\overline{B}\overline{C}D + ABCD$$

OST

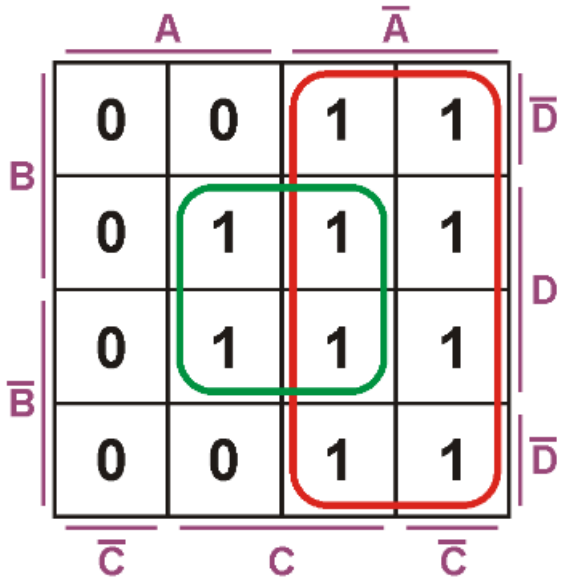
Karnaugh-Plan oder auch KV-Diagramm: Blockbildung

- Um die Gleichung zu vereinfachen, wird versucht, im KV-Diagramm Zellen mit gleichem Inhalt zu Blöcken zusammenzufassen.
- Je größer die Blöcke, desto einfacher wird das Ergebnis. Dabei müssen aber bestimmte Regeln eingehalten werden.
  - Die Blöcke müssen immer rechteckig sein und
  - die Größe einer Zweierpotenz haben, also 2, 4, 8, 16, 32, ...
  - Die Blöcke können auch über den Rand hinaus gehen und mit der gegenüberliegenden Seite verbunden werden,
  - Blöcke können sich teilweise überlappen. Das kann sinnvoll sein, wenn dadurch grössere Blöcke entstehen.
  - Werte, die sowohl einfach als auch negiert vorkommen, werden gestrichen



OST

Karnaugh-Plan oder auch KV-Diagramm: Blockbildung



- Wir finden zwei Blöcke grün und rot
- Der grüne Block umfasst die Werte: $A\bar{A}B\bar{B}CD$. A und B sind überflüssig also: CD
- Der rote Block umfasst $\bar{A}B\bar{B}C\bar{C}D\bar{D}$. Vereinfacht ergibt sich $\bar{A}$.
- Es bleibt die ODER-Verknüpfung grüner und roter Block: $\mathrm{Y} = \bar{A} + CD$
- **Aus** $Y = \overline{ABCD} + \overline{ABCD} + \overline{ABCD} + \overline{ABCD} + \overline{ABCD} + \overline{ABCD} + \overline{ABCD} + \overline{ABCD} + \overline{ABCD} + ABCD$
- **wird** $\mathrm{Y} = \bar{A} + CD$

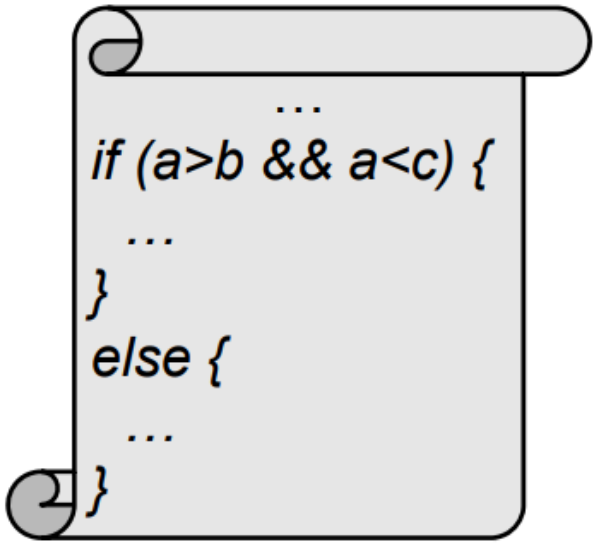
OST

Veranschaulichung der Booleschen Algebra als Schaltalgebra

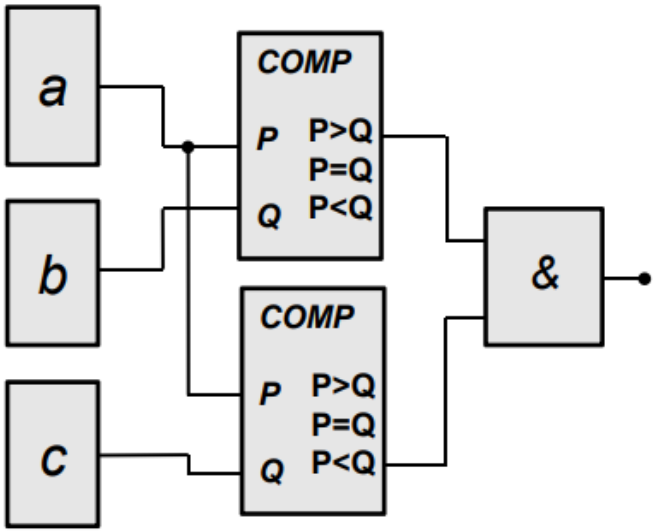
- **Die Schaltalgebra ist eine spezielle Boolesche Algebra**
- **B:= {0,1}**
- **Nullelement: 0 … bedeutet Schalter geöffnet**
- **Einselement: 1 … bedeutet Schalter geschlossen**
- **Operation +: ODER**
- **Operation *: UND**
- **Negation : NICHT( )**
- **Es gelten die Regeln der Booleschen Algebra**

Veranschaulichung der Booleschen Algebra als Schaltalgebra

Aussagenlogik für Programme (Software)



Schaltalgebra zur Realisierung von Schaltungen (Hardware)

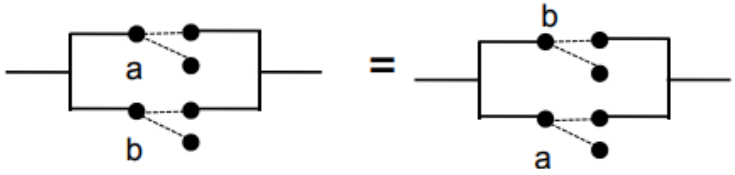


OST

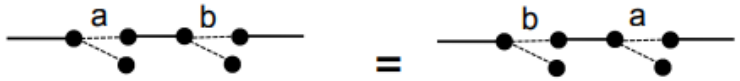
Veranschaulichung der Boolschen Algebra als Schaltalgebra

Die Axiome der Booleschen Algebra gelten.

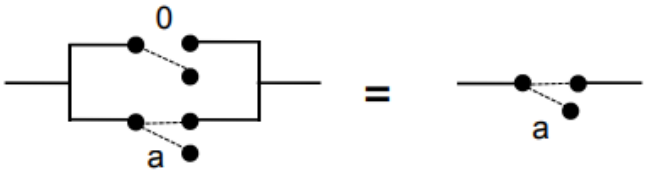
(1) a+b = b+a



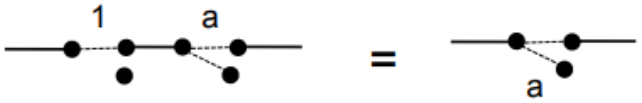
a*b = b*a



(2) 0+a=a



1*a=a

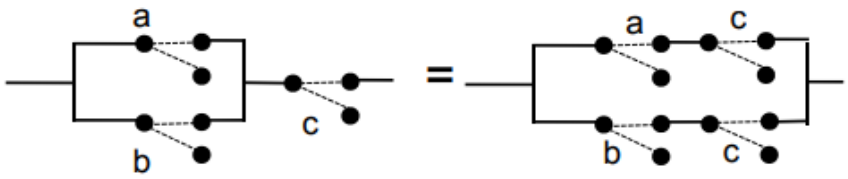


OST

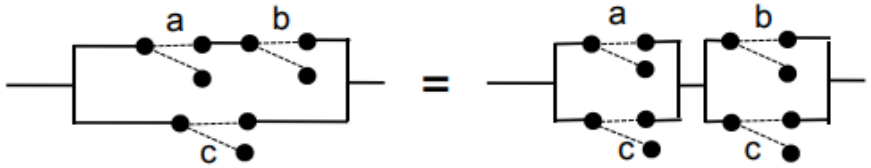
Veranschaulichung der Booleschen Algebra als Schaltalgebra

Die Axiome der Booleschen Algebra gelten.

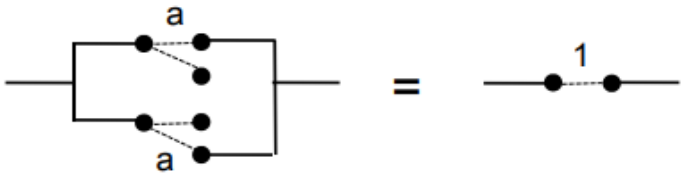
(3) $(a+b)*c = (a*c)+(b*c)$



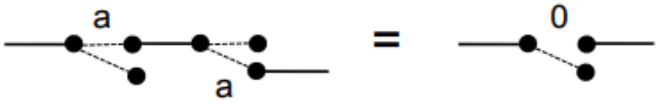
$(a*b)+c = (a+c)*(b+c)$



(4) $a+k(a)=1$



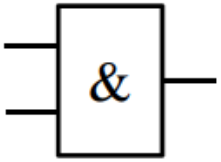
$a*k(a)=0$

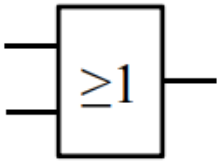


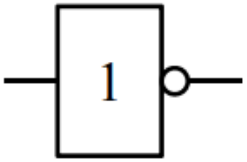
OST

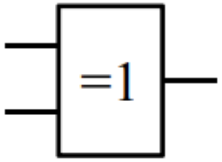
Veranschaulichung der Booleschen Algebra als Schaltalgebra

Schaltsymbole für die gebräuchlichsten Funktionen:

UND 

ODER 

NEGATION 

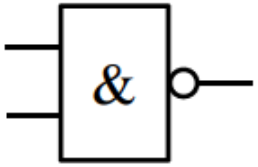
XOR 

OST

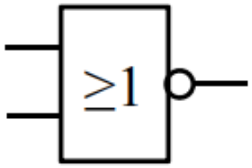
Veranschaulichung der Booleschen Algebra als Schaltalgebra

Schaltsymbole

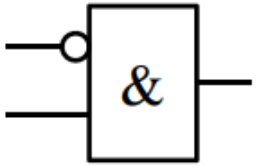
NAND



NOR



Negierte Eingänge,
Am Beispiel NICHT(a) UND b

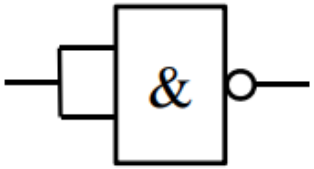


OST

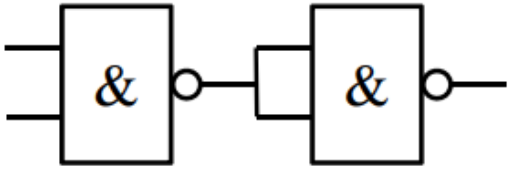
Vollständige Verknüpfungsbasis

Mit NAND lässt sich jede beliebige Schaltfunktion realisieren.

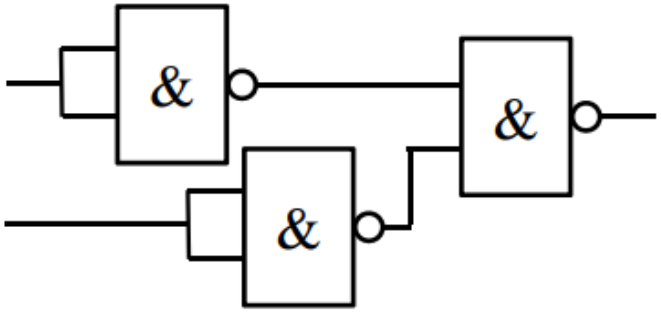
Negation



UND



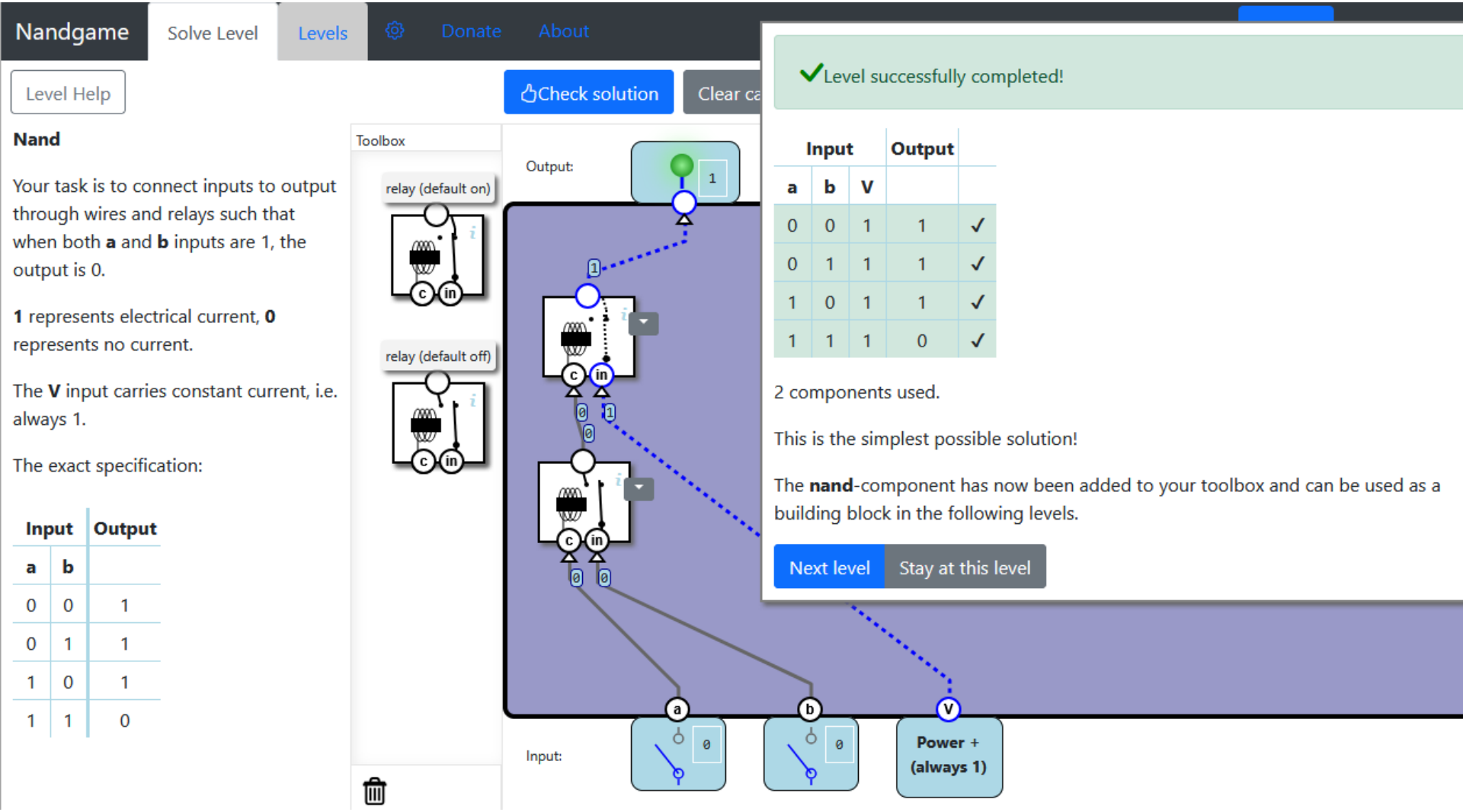
ODER: A ODER B = NICHT (NICHT(A) UND NICHT(B))



Peter Sobe

OST

NAND mit Relais



OST

NAND-Game

Zusätzlich zu Vorlesung und Übung können Sie auch das NAND-Game spielen:

- Hardware spielerisch erlernen
- Konstruktion einfacher Logik aus NAND-Gattern
- Bis zur Konstruktion eines einfachen Prozessors mit Speicher

- Zum Game: Das Nand-Game
- Besser, aber richtig Arbeit! : From Nand to Tetris

Thema:

- **Zufallsvorgänge & Wahrscheinlichkeiten**
- **Ergebnisse und Ereignisse**
- **Kombinatorik, eine Einführung**

Lernziele

- **Sie können den Begriff Zufallsexperiment erklären und Beispiele benennen**
- **Sie können zwischen Ergebnis und Ereignis unterscheiden**
- **Sie kennen das Laplace - Experiment und können es anwenden**
- **Sie können die Binomialverteilung herleiten und an einfachen Beispielen anwenden**
- **Sie kennen einige Regeln der Kombinatorik und können sie an einfachen Beispielen anwenden.**

Ziel der Wahrscheinlichkeitsrechnung:

- **Modellierung & Vorhersage von zufälligen Vorgängen, wie z.B.**
  - Auftreten eines Bitfehlers
  - Auftreten eines Zeichens in einem Code
  - Definition des Informationsgehaltes eines Zeichens

Alles was lediglich
wahrscheinlich ist,
ist wahrscheinlich falsch.

René Descartes

***Definition*: (Zufallsvorgang, Zufallsexperiment)**

Unter einem *Zufallsvorgang* verstehen wir einen Vorgang, bei dem

- im Voraus feststeht, welche möglichen 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).

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

Zufallsvorgänge Beispiele

- **Ziehung der Lottozahlen**
- **Roulette, Münzwurf, Würfelwurf**
- **Ermitteln der Bitfehlerrate**
- **Bestimmen, das 1, 2, .. Fehler in einem Datenwort auftreten**
- **Wahrscheinlichkeit, dass genau ein Teilnehmer (host) in Δt auf einen Ethernet Strang zugreift**
- **Ermitteln des Informationsgehalts eines Zeichens**
- **Ermittlung der Kanalmatrix**

Ergebnis & Ergebnismenge

***Definition*: (Ergebnismenge)**

Die Menge aller möglichen Ausgänge (Ergebnisse) eines Zufallsvorgangs heisst *Ergebnismenge* und wird mit Ω bezeichnet.

Ein einzelnes Element $\omega \in \Omega$ heisst Ergebnis. Wir notieren die Anzahl aller Elemente von Ω, d.h. die Anzahl aller Ergebnisse, mit $|\Omega|$.

OST

Ergebnismengen Beispiele

Zufallsvorgang: ‘Werfen eines Würfels’

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

Zufallsvorgang: ‘Werfen einer Münze so lange, bis Kopf erscheint’:

$$\Omega = \{K, ZK, ZZK, ZZZK, ZZZZK, \dots\}$$

OST

Eigenschaften von Wahrscheinlichkeiten

Wahrscheinlichkeit des Komplementärereignisses:

- $P(\overline{A}) = 1 - P(A)$

Wahrscheinlichkeit des unmöglichen Ereignisses:

- $P(\theta) = 0$

Wertebereich der Wahrscheinlichkeit:

- $0 \leq P(A) \leq 1$

Wahrscheinlichkeitsdefinition nach Laplace

Pierre-Simon Marquis de Laplace, 1749 - 1827:

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

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

OST

Beispiel fairer Würfel

Es ist:

- $\Omega = \{\omega_i\}$ **mit** $i = 1 \ldots 6 = \{1, 2, 3, 4, 5, 6\}$

Laplace-Wahrscheinlichkeit für das Ereignis A:

- **Würfeln einer beliebigen Zahl ω_i aus der Menge Ω.**
- $P(\{\omega_i\}) = \frac{Anzahl\ der\ guenstigen\ Ergebnisse}{Anzahl\ aller\ Ergebnisse\ \Omega} = \frac{|\omega_i|}{|\Omega|} = \frac{1}{6}$
- **Wie gross ist die Laplace-Wahrscheinlichkeit eine gerade Zahl zu würfeln?**

Laplace-Wahrscheinlichkeit erfordert Berechnung von Anzahlen

Mathematische Technik hierfür: *Kombinatorik*

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.**

Geordnete Proben

- Im Folgenden sei M eine Menge mit n Elementen.

- Bilden wir nun k-Tupel, deren Einträge aus der Menge M stammen, so gibt es zwei Möglichkeiten:

a) Die Einträge dürfen sich wiederholen

b) Die Einträge sind alle verschieden.

Geordnete Proben mit Wiederholung

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

- **Beispiel**
- **Bei einem Ziffernschloss muss man eine 5-stellige Zahl einstellen, die aus den Ziffern 0,1,…,9 gebildet wird.**
- **Wie viele Kombinationen gibt es?**

$$10^5 = 100.000$$

Geordnete Proben ohne Wiederholung

- Die Anzahl der k-Tupel aus einer n-Menge ohne Wiederholung ist
- Anzahl = n · (n-1) · (n - 2) · …. · (n – k + 1)
- Formel: $\text{Anzahl} = \frac{n!}{(n-k)!}$
- oder besser:

  $\text{Anzahl} = \prod_{n}^{n-k+1} n$

  warum?

Geordnete Proben ohne Wiederholung

Beispiel

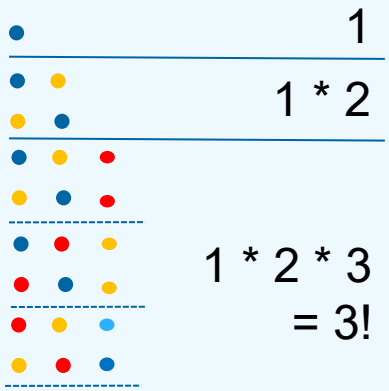
Beim Pferde-Toto “3 aus 18” muss man von 18 Pferden 3 gemäss **der Reihenfolge ihres Zieleinlaufs** tippen.

Wieviele Tippmöglichkeiten gibt es?

Permutationen

Die Zahl der Permutationen einer n-Menge ist n!

- **Bemerkung:**
  - Dies ist ein Spezialfall eines k-Tupels aus einer n-Menge ohne Wiederholung für n = k.



Beispiel

Zur Festlegung einer Sitzordnung bei einer Feier mit 10 Personen gibt es

10 ! = 3.628.800 Möglichkeiten.

OST

Ungeordnete Proben

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

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

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

Kompakte aber nicht praktikable Formel. Warum?

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

OST

- **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?

$$\binom{25}{2} = \frac{25 \cdot 24}{2!} = 300$$

Kombinatorik: Zusammenfassung

Anzahl A der Möglichkeit			
n Optionen k Auswählen		Beachtung der Reihenfolge	
		MIT	OHNE
zurücklegen	MIT	$A = n^k$	$A = \binom{n+k-1}{k}$
	OHNE	$A = n\,(n-1)..(n-k+1)$ $A = \frac{n!}{(n-k)!}$	$A = \frac{n!}{k!\,(n-k)!} = \binom{n}{k}$

OST

Die Binomial-Verteilung

- Anwendung:
  - Das Zufallsexperiment unterscheidet nur zwei Ergebnisse
  - Das Experiment wird n-mal wiederholt (Zufallsstichprobe vom Umfang n)
  - Gesucht: Die Wahrscheinlichkeit, dass bei n-maliger Durchführung des Experimentes das Ereignis
    - wahrscheinlich
    - genau
    - mindestens
    - Höchstens x-mal

    eintrifft

- **Bei n Wiederholungen έ genau x Mal auftritt z.B.**
- **0 Mal: es gibt nur 1 Möglichkeit**
- **1 Mal: es gibt n Möglichkeiten, hier 6**

- **2 Mal:** $\binom{n}{2}$ *hier* $\binom{6}{2} = 3 * 5 = 15$

OST

- **Damit ergibt sich die Wahrscheinlichkeit, dass x zum Beispiel in n = 6 genau 2 Mal Auftritt und die Auftrittswahrscheinlichkeit gleich 0.8 ist zu:**

$$f(x) = P(X = x) = \binom{6}{2} 0.8^2 (1 - 0.8)^{6-2} = 0.015136$$

Es gibt 15 Kombinationen

Ist genau 2 Mal aufgetreten

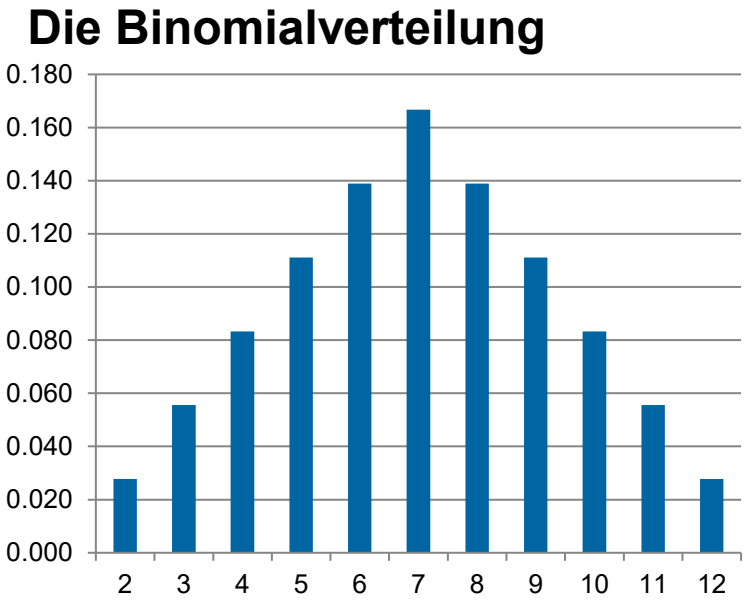
Ist genau 4 Mal nicht aufgetreten

OST

Zusammenfassung Binomialverteilung

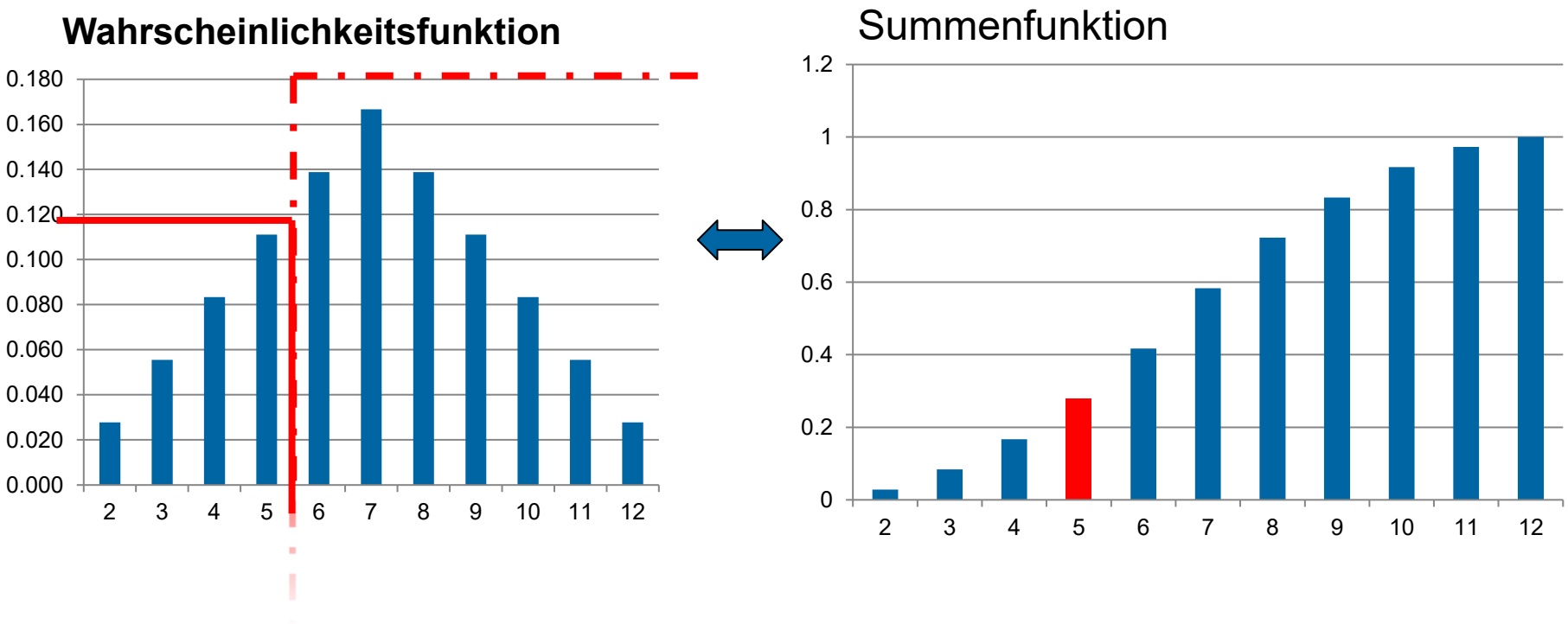
$$f(x) = P(X = x) = \binom{n}{x} p^x (1-p)^{n-x}$$

$$E(X) = np$$



OST

Mindestens - Höchstens



OST

Die Binomial-Verteilung: Beispiel

- **Gegen eine Krankheit wurde ein neues Medikament entwickelt. Die Heilungschance liegt bei 90%. Wie gross ist die Wahrscheinlichkeit, dass bei 5 zufällig gewählten Patienten mindestens 4 geheilt werden?**

$$f(x) = P(X = x) = \binom{n}{x} p^x (1-p)^{n-x}$$

$$E(X) = np$$

Für $P(X = 4)$:

$$P(X = 4) = \binom{5}{4} \cdot (0.9)^4 \cdot (0.1)^1 = \frac{5!}{4!(5-4)!} \cdot \frac{81}{100} \cdot \frac{1}{10}$$

Für $P(X = 5)$:

$$P(X = 5) = \binom{5}{5} \cdot (0.9)^5 \cdot (0.1)^0 = \frac{5!}{5!(5-5)!} \cdot \frac{59049}{100000} \cdot 1$$

P(4) = 0.32805
P(5) = 0.59049
P(5+4)= 0.91854

E(X) =5 × 0.9=4.5

Selbststudium

Lesen Sie das Skript: *Wahrscheinlichkeitsrechnung ohne Ballast*

- **Zur Vertiefung und Selbststudium**
- **Zum Erlernen der Grundbegriffe und Fertigkeiten der Kombinatorik**
  - Insbesondere wichtig: Bedingte Wahrscheinlichkeit

- **Sie finden das Skriptlet auf Teams bei den Kursmaterialien.**

Ziel d

er Wahrscheinlichkeitsrechnung:

- **Modellierung & Vorhersage von zufälligen Vorgängen, wie z.B.**
  - Auftreten eines Bitfehlers
  - Auftreten eines Zeichens in einem Code
  - Definition des Informationsgehaltes eines Zeichens

Alles was lediglich wahrscheinlich ist,
ist wahrscheinlich falsch.

René Descartes

Laplace Experiment:

$$p(x_i) = \frac{Anzahl\ f\ddot{u}r\ x_i\ günstige\ Fälle}{Alle\ Fälle}$$

OST

Eigenschaften von Wahrscheinlichkeiten

Wahrscheinlichkeit des sicheren Ereignisses

- $P(A) = 1$

Wahrscheinlichkeit des Komplementärereignisses:

- $P(\overline{A}) = 1 - P(A)$

Wahrscheinlichkeit des unmöglichen Ereignisses:

- $P(\theta) = 0$

Wertebereich der Wahrscheinlichkeit:

- $0 \leq P(A) \leq 1$

Additionssatz für Wahrscheinlichkeiten:

- $P(A \cup B) = P(A) + P(B) - P(A \cap B)$

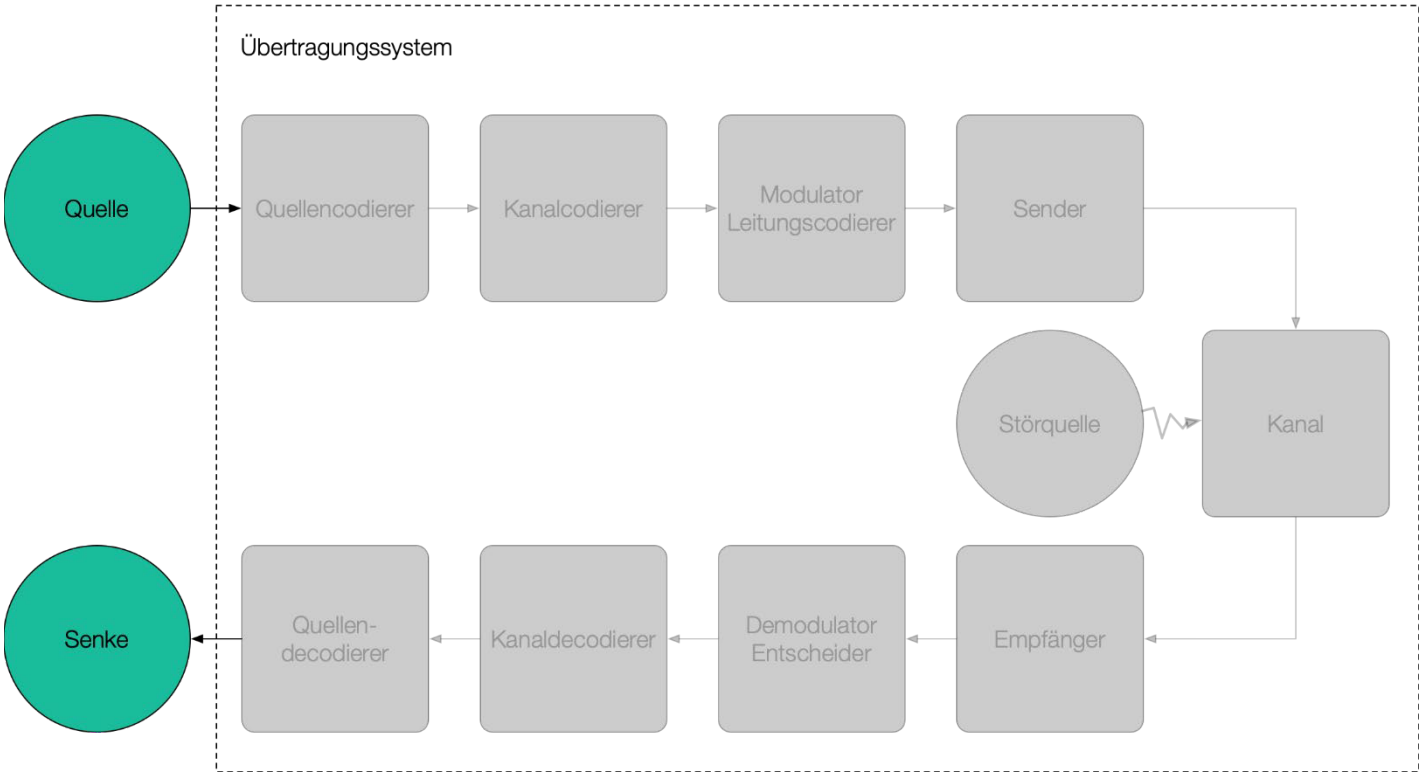
Additionssatz für 3 Ereignisse (Wahrscheinlichkeit, dass A, B oder C eintritt):

- $P(A \cup B \cup C) = P(A) + P(B) + P(C) - P(A \cap B) - P(A \cap C) - P(C \cap B) + P(A \cap B \cap C)$

Lernziele

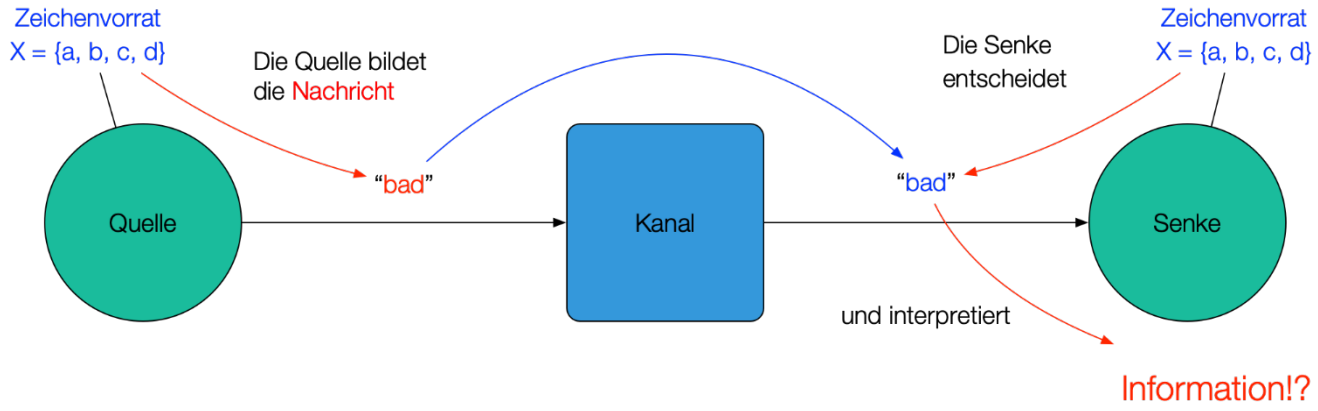
- **Sie können den Zusammenhang zwischen Information und Wahrscheinlichkeit erklären**
- **Sie kennen und verstehen das Modell der Kommunikation**
- **Sie können die Begriffe Redundanz, Irrelevantes und Information erklären**
- **Sie können an einfachen Beispielen die Begriffe Entscheidungsgehalt, Informationsgehalt und Entropie rechnerisch erläutern**
- **Sie verstehen die Bedeutungen von Codes, die die Präfix-Eigenschaft haben**
- **Sie verstehen den Unterschied von Codes mit und ohne Gedächtnis und können einfache Berechnungen durchführen**

Modell der Informationsverarbeitung



OST

Information



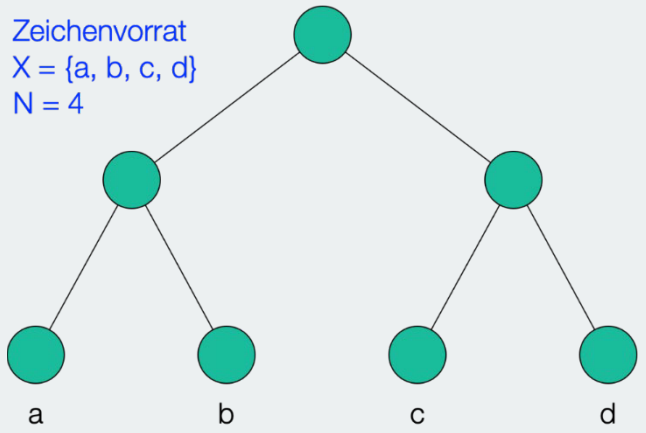
Nachricht (Darstellung & Bedeutung)	redundant	nicht-redundant
irrelevant	Zeichenvorrat bei Quelle und Senke verschieden	
relevant	vorhersagbar	Information

OST

***Definition*: (Entscheidungsgehalt)**

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

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



OST

***Definition*: (Informationsgehalt)**

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

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

Entropie

***Definition*: (Entropie)**

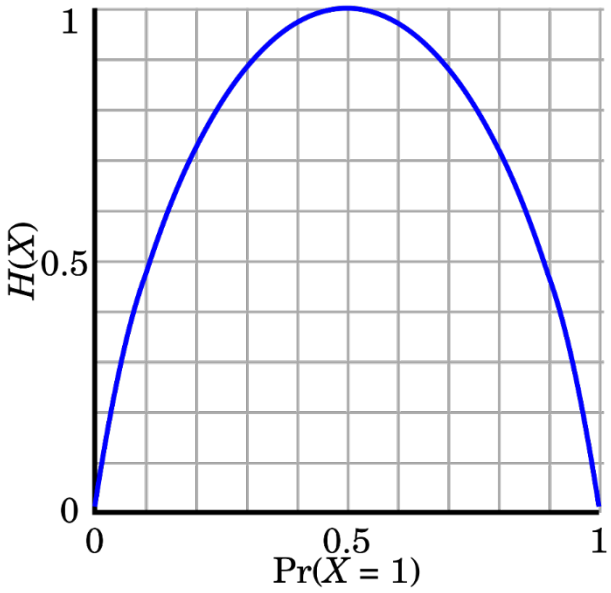
Die ***Entropie*** bezeichnet den **mittleren Informationsgehalt** der Quelle. Sie zeigt also auf, wie viele Elementarentscheidungen die Quelle/Senke im Mittel pro Zeichen treffen muss

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

OST

Wann wird der mittlere Informationsgehalt, die Entropie, einer Quelle/Senke maximal? (Ansatz: Binäre Quelle!)

$X = \{x_1, x_2\}$
$p(x_1) = p$
$p(x_2) = 1 - p$
$\Rightarrow H(X) = p * log_2\left(\frac{1}{p}\right) + (1-p) * log_2\left(\frac{1}{1-p}\right)$



Redundanz der Quelle:

$absolut: \quad R_{Qabsolut} = H_0 - H(X)\ [bit/Zeichen]$

$relativ: \quad R_{Qrelativ} = \frac{R_{Qabsolut}}{H_0} = \frac{H_0 - H(X)}{H_0} = 1 - \frac{H(X)}{H_0}\ [\%]$

OST

Zusammenhang Informationsgehalt und Codewortlänge

Es galt: ***Informationsgehalt*** **eines Zeichens ist:**

$$I(x_k) = log_2\left(\frac{1}{p(x_k)}\right)\ [bit],$$ **kann eine reelle Zahl sein!**

Damit folgt für die Entropie als Mittelwert des Informationsgehalt:

$$H(X) = \sum_{k=1}^{N} p(x_k) * I(x_k) = \sum_{k=1}^{N} p(x_k) * log_2(\frac{1}{p(x_k)})\ \left[\frac{bit}{Zeichen}\right],$$ **kann eine reelle Zahl sein!**

Die tatsächliche Codewortlänge L**:**

$$\mathrm{L}(x_k) = \left\lceil log_2\left(\frac{1}{p(x_k)}\right)\right\rceil\ [bit] = \lceil I(x_k)\rceil,$$ **muss ein Integer sein!**

Damit folgt für die Entropie des Codes als Mittelwert der Codewortlänge:

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

kann eine reelle Zahl sein!

Mittlere Codewortlänge

***Definition*: (Mittlere Codewortlänge)**

Die *Mittlere Codewortlänge* berechnet sich wie folgt

$$H_c(X) = \sum_{i=1}^{N} p(x_i) \cdot L(x_i) \ [bit/Zeichen]$$

- Bei der Quellencodierung werden die diskreten Zeichen der Quelle auf binäre CW abgebildet.
- Günstig ist, wenn die mittlere Codewortlänge L möglichst klein ist.
- Beispiele für CW:
  - *ASCII*: Blockcode mit fester Wortgrösse $L = 8\,[bit]$ (7 bit ASCII + 0)
  - *Morsecode*: variable Wortgrösse $L = 1\ bis\ 4\,[bit]$ (*Berücksichtigung der Auftritts-wahrscheinlichkeit der einzelnen Zeichen)*

OST

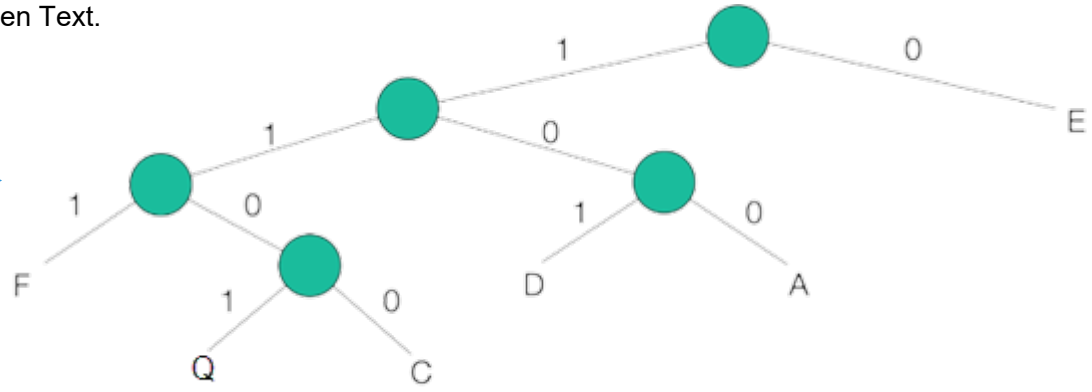
Binärcodierung: Präfixeigenschaft

Zeichen	Wahrscheinlichkeit*	ASCII Code	Morse Code
A	0.0642	(0)1000001	01
B	0.0127	(0)1000010	1000
C	0.0218	(0)1000011	1010
D	0.0317	(0)1000100	100
E	0.1031	(0)1000101	0
F	0.0208	(0)1000110	0010
Q	0.0002	(0)1010001	1101

*Annahme: Wahrscheinlichkeitsangabe für englischen Text.

→ **Problem?**

Besitzt die Präfixeigenschaft bzw.
ist ein kommafreier Code.



OST

Shannon’sches Codierungstheorem

Für jede beliebige zugehörige Binärcodierung mit Präfixeigenschaft ist die mittlere Codewortlänge nicht kleiner als die Entropie $H(X)$:

$$H(X) \leq L$$

Für jede beliebige Quelle kann eine Binärcodierung gefunden werden, so dass die folgende Ungleichung 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:* $R_C = L - H(X)\ [bit/Zeichen]$

Shannon'sches Codierungstheorem - Beispiel

Gegeben seien die Zeichen a, b, c und d mit den Auftrittswahrscheinlichkeiten
P{a} = 0.5, P{b} = 0.25, P{c} = 0.125 und P{d} = 0.125.

1. Bestimmen Sie den Entscheidungsgehalt H_0 und den Informationsgehalt $I(x_i)$ der Zeichen, sowie den mittleren Informationsgehalt (Entropie) H(X) der Quelle.
2. Wie gross ist die Redundanz der Quelle?
3. Welche Bedingung muss erfüllt sein, damit ein Code redundanzfrei ist?
4. Entwickeln Sie für das gegeben Beispiel einen Code der redundanzfrei ist! Besitzt dieser die Präfixeigenschaft?
5. Ist das immer möglich?

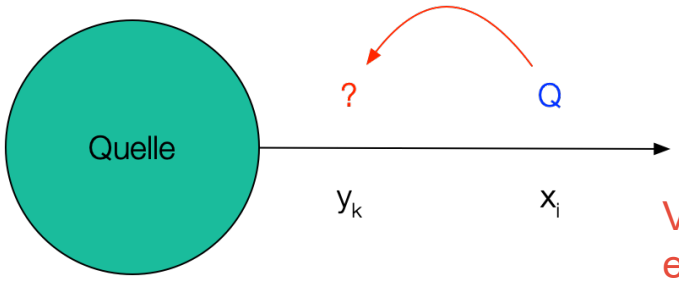
Bisher: **Quelle ohne Gedächtnis**

- Die Auftrittswahrscheinlichkeit eines Zeichens ist unabhängig von dem zuvor emittierten Zeichen bzw. der zuvor emittierten Zeichenfolge, d.h. die Verbundwahrscheinlichkeit für die beiden Zeichen x_i und y_k ist:

$$p(x_i, y_k) = p(x_i) \cdot p(y_k)$$

- Allgemein kann nicht von einer gedächtnislosen Quelle ausgegangen werden!

$$p(x_i, y_k) = p(x_i) \cdot p(y_k|x_i)$$



Deutsche Sprache

Verbundentropie zweier Zeichen einer gedächtnisbehafteten Quelle:

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

Ohne Gedächtnis:

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

OST

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

Um $H(X,Y)$ *zu bestimmen*

setzen wir statt $p(x_i)$, $p(x_i, y_k) = p(x_i) \cdot p(y_k|x_i)$ ***in die obige Gleichung ein, es folgt:***

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

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

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

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

$$= \qquad H(X) \qquad + \qquad H(Y|X)$$

Es kann gezeigt werden, dass gilt:

$$H_0 \geq H(Y) \geq H(Y|X)$$

Für die Redundanz gilt:

$$R_Q = H_0 - H_{\boldsymbol{oG}}(X) \leq H_0 - H_{\boldsymbol{mG}}(X) = H_0 - H(Y|X)$$

Interpretation:

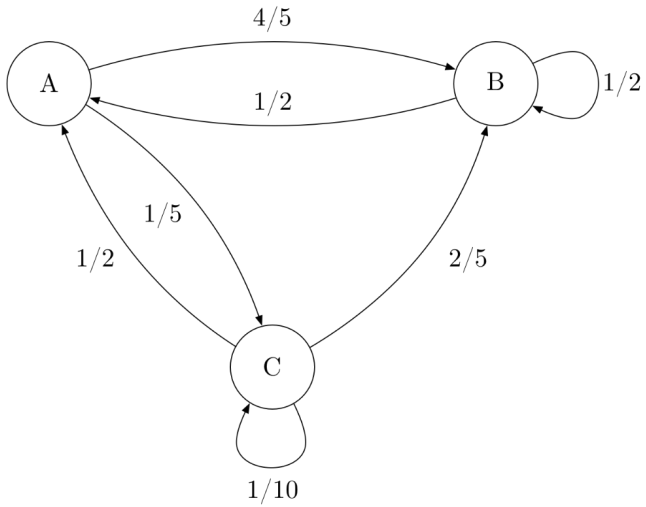
- Die mittlere Entropie einer Quelle ohne Gedächtnis ist stets grösser oder gleich der Entropie einer Quelle mit Gedächtnis.
  $R_Q = H_0 - H_{\boldsymbol{oG}}(X) \leq H_0 - H_{\boldsymbol{mG}}(X)$
- In der Quellencodierung sind daher nicht Einzelzeichen zu codieren, sondern stets Zeichenketten.
- Beispiel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

Markov Model hat Zusammenhang zu Viterbi Algorithmus.
Der Viterbi Algorithmus ist am Schluss bei Faltungscodes in Nutzung.

The Viterbi algorithm is a fundamental dynamic programming technique widely used in the context of Hidden Markov Models (HMMs) to uncover the most likely sequence of hidden states given a sequence of observed events.

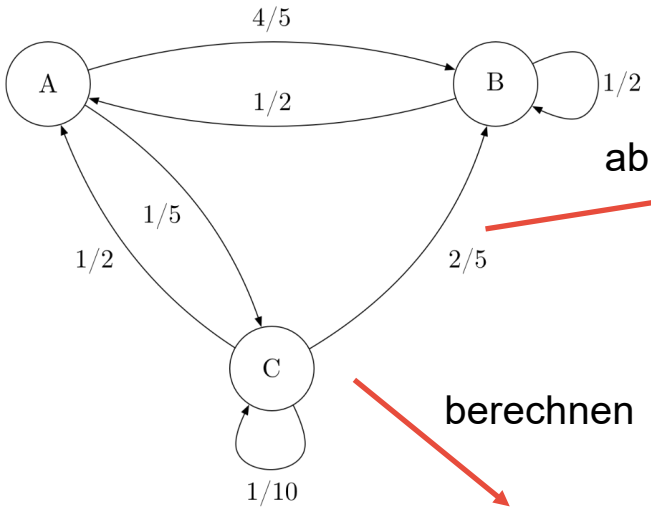
Gegeben sei eine Quelle mit dem Alphabet A, B, C. Die Abhängigkeiten werden durch ein Markov-Diagramm 1. Ordnung beschrieben.



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

OST

Beispiel: Diskrete Quelle mit Gedächtnis



ablesen

$p(y\|x)$ bedingte Wahrscheinlichkeit		$y=$ A	$y=$ B	$y=$ C
$x=$	A	0	4/5	1/5
	B	1/2	1/2	0
	C	1/2	2/5	1/10

berechnen

$xi=$	$p(x)$
A	9/27
B	16/27
C	2/27

berechnen

berechnen

$p(x,y)$ Verbundwahrscheinlichkeit		$y=$ A	$y=$ B	$y=$ C
$x=$	A	0	4/15	1/15
	B	8/27	8/27	0
	C	1/27	4/135	1/135

$$Verbundentropie\ H(X,Y) = \sum_{i=1}^{N}\sum_{k=1}^{N} p(x_i, y_k) \cdot \log_2\left(\frac{1}{p(x_i, y_k)}\right) = 2.1878bit$$

$$bedingte\ Entropie\ H(X|Y) = H(X,Y) - H(X) = 2.1878 - 1.2454 = 0.9424bit$$

$$Redundanz = H_0 - H(X|Y) = 1.584 - 0.9424 = 0.6416bit$$

OST

Quellencodierung & Komprimierung

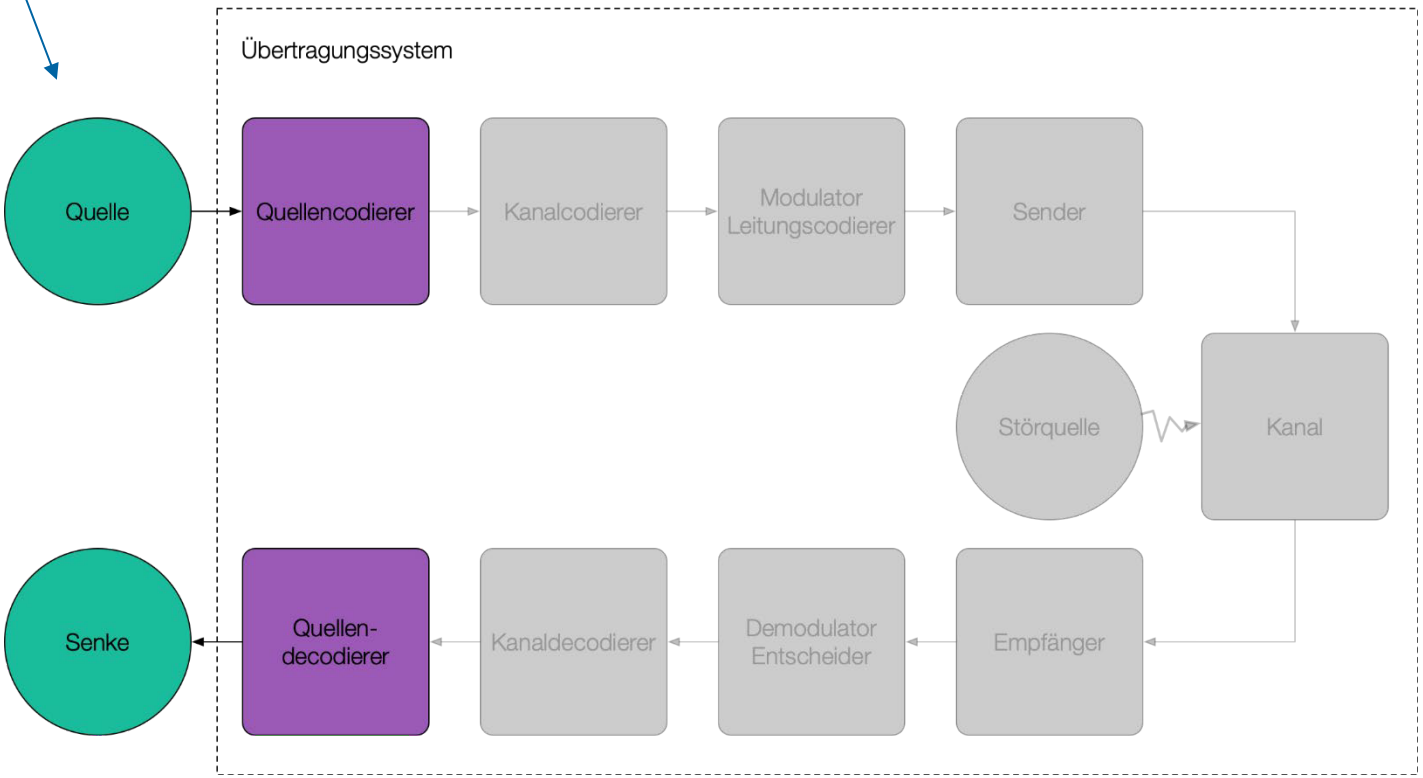
- **Rückblick**
- **Anforderungen an Komprimierungsverfahren**
- **Huffman-Codierung**
- **Lauflängen-Codierung**
- **Lempel Ziv Algorithmus**

Modell der Informationsverarbeitung

Hat Eigenschaften

Das sind:

- Informationsgehalt → erhalten
- Entropie → maximieren
- Redundanz → entfernen
- Präfixeigenschaft



OST

Informationsgehalt & Entropie: Wiederholung

***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

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

OST

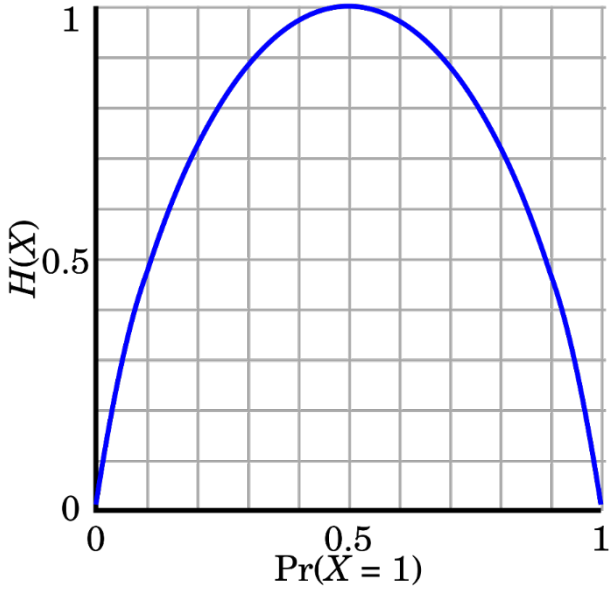
Wann wird der mittlere Informationsgehalt, die Entropie, einer Quelle/Senke maximal? (Ansatz: Binäre Quelle!)

$X = \{x_1, x_2\}$
$p(x_1) = p$
$p(x_2) = 1 - p$

$$\Rightarrow H(X) = p * log_2\left(\frac{1}{p}\right) + (1-p) * log_2\left(\frac{1}{1-p}\right)$$

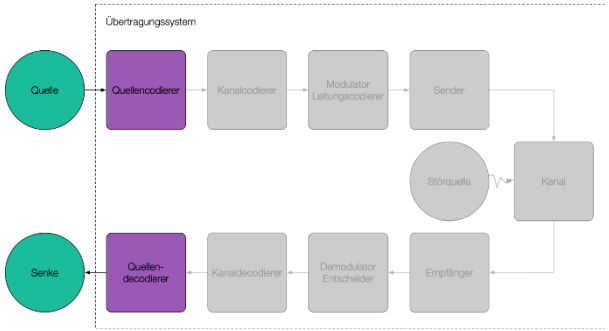
Redundanz der Quelle:

$$R_Q = H_0 - H(X)\ [bit/Zeichen]$$



OST

Beispiele zur Quellencodierung



Quellencodierung

Datenkomprimierung
- Verlustfrei
- Verlustbehaftet

Verschlüsselung
- Symmetrisch
- Asymmetrisch

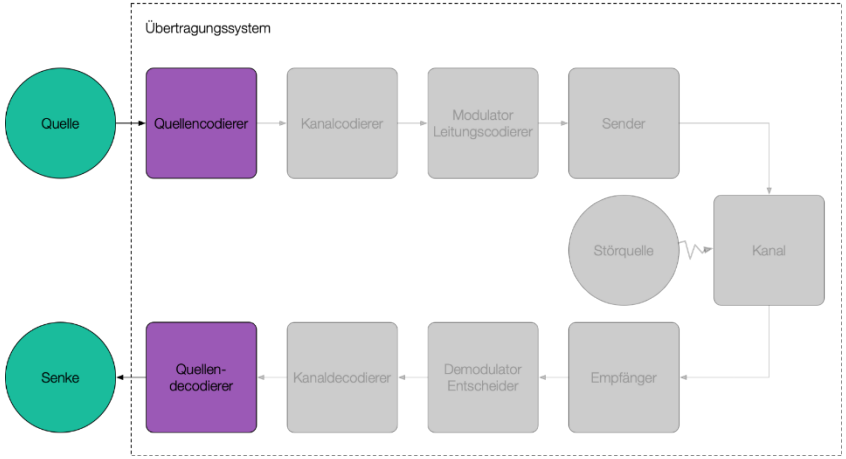
Zusätzlich Literatur Verschlüsselungsverfahren:
Einführung unter:
http://andreas-romeyke.de/Projekt1/ffbr/web.html

OST

Verlustfreie Datenkomprimierung

Anforderungen:

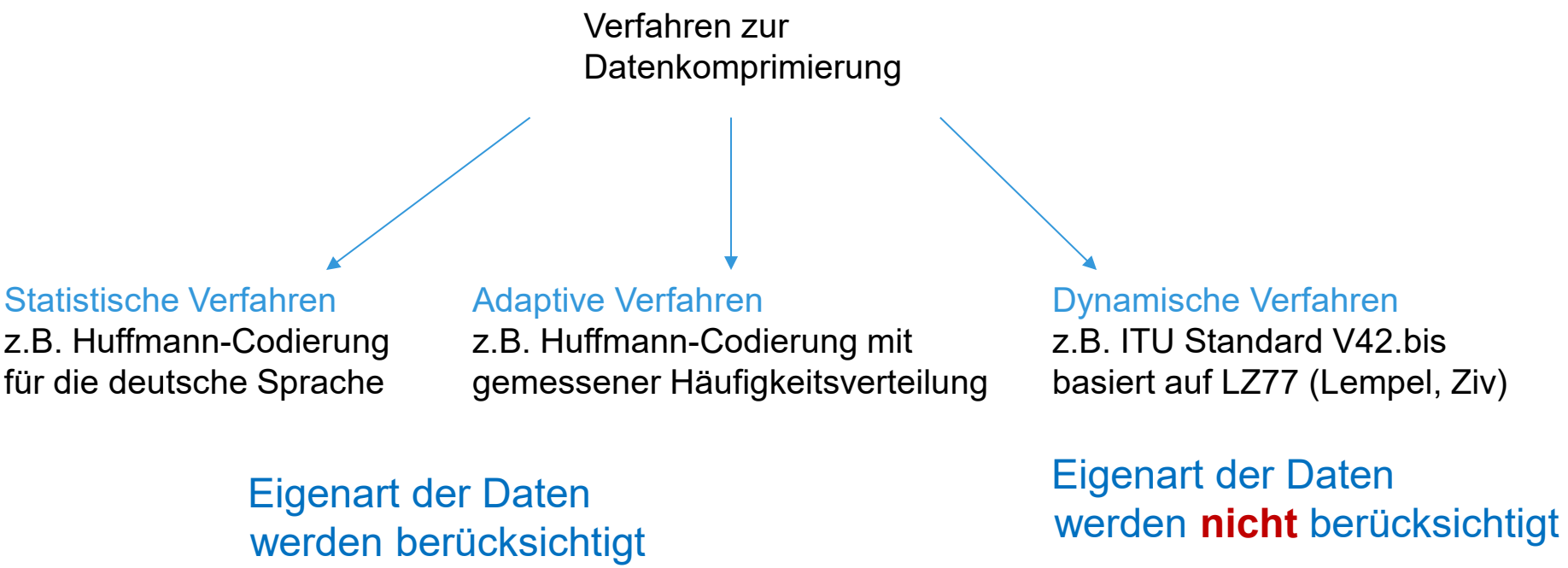
- hohe Komprimierungsrate
  für alle Typen von Daten (idealerweise ohne Kenntnis über die Eigenart der Daten)
- hohe Encode- und Decode-Geschwindigkeit
- geringe Ansprüche an die Hardware



OST

Das Ziel der Datenkomprimierung ist, den Aufwand der Datenspeicherung und Datenübertragung zu reduzieren.

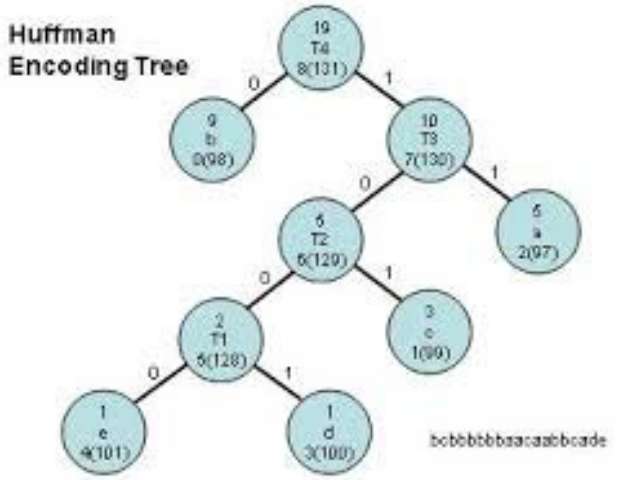
D.h. Entfernen von Redundanz und Irrelevanz



OST

Datenkomprimierung nach Huffman

- Entfernen der Redundanz der Quelle



OST

Shannon'sches Codierungstheorem: Wiederholung als Basis für Huffman

1. **Für jede beliebige zugehörige Binärcodierung mit Präfixeigenschaft ist die mittlere Codewortlänge nicht kleiner als die Entropie $H(X)$:**

$$H(X) \leq L$$

2. Für jede beliebige Quelle kann eine Binärcodierung gefunden werden, so dass die folgende Ungleichung 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:* $R_C = L - H(X)\ [bit/Zeichen]$

OST

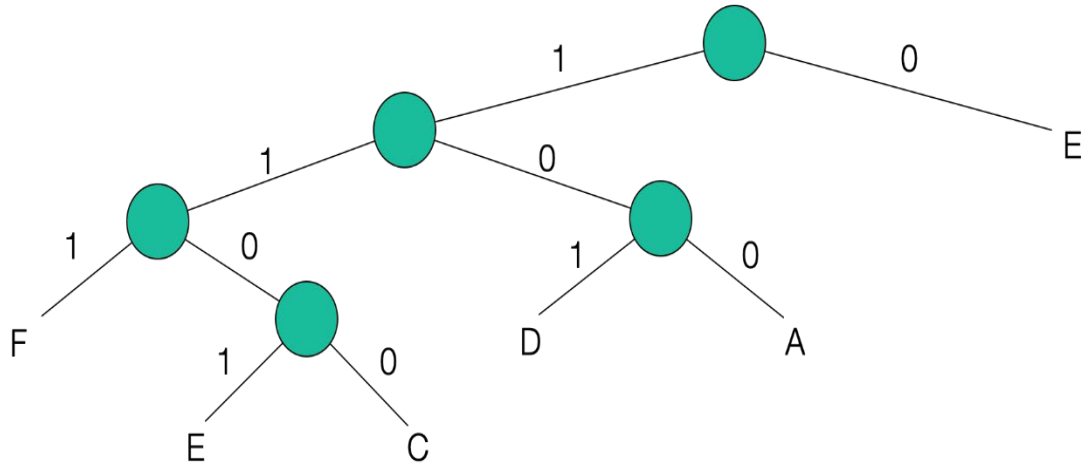
Binärcodierung: Präfixeigenschaft: Wiederholung

Zeichen	Wahrscheinlichkeit*	ASCII Code	Morse Code
A	0.0642	01000001	01
B	0.0127	01000010	1000
C	0.0218	01000011	1010
D	0.0317	01000100	100
E	0.1031	01000101	0
F	0.0208	01000110	0010

→ **Problem?**

*Annahme: Wahrscheinlichkeitsangabe für englischen Text.

Besitzt die Präfixeigenschaft bzw.
ist ein kommafreier Code.



OST

Huffman-Codierung

Verfahren zur Entwicklung eines kommafreien Codes mit minimaler mittlerer Codewortlänge

Rekursives Verfahren, d.h. der Binärbaum wird nicht von der Wurzel, sondern von den Blättern aus entwickelt

Huffman-Codierung

Verfahren:

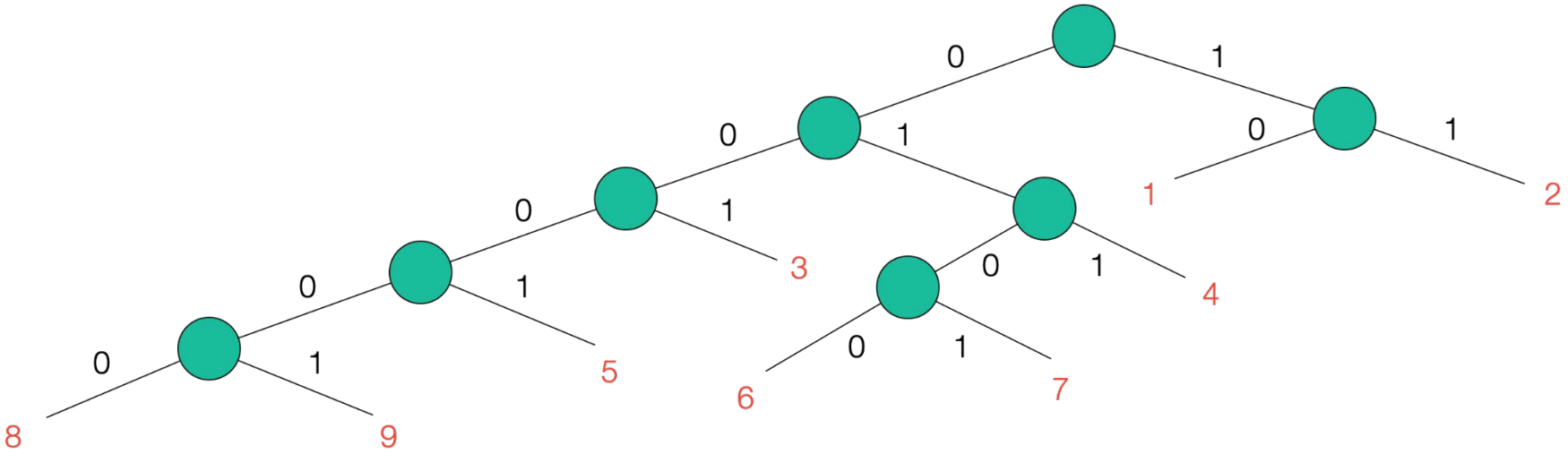
- **Ordne die Zeichen gemäss ihrer Auftrittswahrscheinlichkeit**
- **Die beiden Zeichen mit der kleinsten Auftrittswahrscheinlichkeit haben die gleiche CW-Länge L_N**
- **Sei L_N die mittlere CW-Länge für eine Quelle mit N Zeichen und L_{N-1} die mittlere CW-Länge für den Fall, dass die beiden letzten zu einem einzigen Zeichen zusammengefasst werden, dann gilt:**

$$L_N - \big(p(x_{N-1}) + p(x_N)\big) \cdot L(x_N) = L_{N-1} - \big(p(x_{N-1}) + p(x_N)\big) \cdot (L(X_N) - 1)$$

$$\Rightarrow L_N = L_{N-1} + p(x_{N-1}) + p(x_N)$$

Beispiel: Huffman-Codierung

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



OST

Huffman-Code: Animation

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

x_i	1	2	3	4	8	9	5	6	7
Code					0	1			
$p(x_i)$	0.22	0.19	0.15	0.12	0.1		0.08	0.07	0.07

x_i	1	2	3	6	7	4	8	9	5
Code				0	1		0	1	
$p(x_i)$	0.22	0.19	0.15	0.14		0.12	0.1		0.08

OST

Huffman-Code: Animation

x_i	1	2	8	9	5	3	6	7	4
Code			00	01	1		0	1	
$p(x_i)$	0.22	0.19	0.18			0.15	0.14		0.12

x_i	6	7	4	1	2	8	9	5	3
Code	00	01	1			00	01	1	
$p(x_i)$	0.26			0.22	0.19	0.18			0.15

x_i	8	9	5	3	6	7	4	1	2
Code	000	001	01	1	00	01	1		
$p(x_i)$	0.33				0.26			0.22	0.19

OST

x_i	1	2	8	9	5	3	6	7	4
Code	0	1	000	001	01	1	00	01	1
$p(x_i)$	0.41		0.33				0.26		

x_i	8	9	5	3	6	7	4	1	2
Code	0000	0001	001	01	100	101	11	0	1
$p(x_i)$	0.59							0.41	

x_i	8	9	5	3	6	7	4	1	2
Code	00000	00001	0001	001	0100	0101	011	10	11
$p(x_i)$	1.0								

OST

Einfache Lauflängenkomprimierung

- **Erkennen von Wiederholungen**
  - RLE (*Run Length Encoding)*, oder
  - RLC genannt (*Run Length Coding*),
  - wird bei vielen Bildformaten benutzt
    zum Beispiel BMP, PCX und TIFF).
  - Diese Komprimierungsmethode beruht auf der Verkürzung von Wiederholungen aufeinanderfolgender Elemente.

  - Quelltext w: Agggbbehfffgggg => |w|=15

  - Codiert w_c : A3g2beh3f4g => | w_c |= 11

OST

Lauflängenkomprimierung: 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 ebenfalls.
  - Ausnahme ist, wenn das Ende der Nachricht erreicht ist**. eof**

Lauflängenkomprimierung: Bit-Folgen

- Der Kodierer einigt sich nun mit dem Dekodierer darauf, mit welchem Bit begonnen wird, "0" ODER "1"
  - Das kann entweder durch Konvention sein oder
  - bspw. durch ein zusätzliches Bit zu Beginn.
  - Anschließend werden abwechselnd die Längen der Null- und Eins-Folgen übertragen.
  - Der Dekodierer muss anschliessend nichts anderes tun, als zu jedem empfangenen Wert entsprechend viele Null- oder Eins-Bits auszugeben

Lauflängenkomprimierung: Bit-Folgen

- Beispiel:
- Originalnachricht: w = 1111 1110 0000 1000 0001 1111 => |w| = 24 bit
- Start mit einer "1"

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 => $|w_c|$ = 15

OST

Datenkomprimierung nach Lempel Ziv

- **Erkennen von wiederkehrenden Mustern**

Die Ziv-Lempel Verfahren sind, in der Gruppe der verlustfrei packenden Algorithmen, die, die diesem Ideal noch am nächsten kommen. Die ersten Verfahren von Jacob Ziv und Abraham Lempel sollen in dieser Arbeit vorgestellt werden. Die Grundidee all dieser Verfahren ist es eine Zeichenkette, die schon einmal gesendet/gespeichert wurde und nun wieder auftaucht, durch einen Zeiger auf die Stelle wo sie eher auftauchte oder durch einen Zeiger auf ein Wörterbuch zu ersetzen. Damit soll die Länge der zu codierenden Zeichenkette beträchtlich gekürzt werden.

André Lichei, TU-Chemnitz

OST

Datenkomprimierung nach Lempel Ziv

- **Erkennen von wiederkehrenden Mustern**

LZ77 ist die Abkürzung für das Komprimierungsverfahren, welches Jacob Ziv und Abraham Lempel 1977 in dem Artikel "A Universal Algorithm for Sequential Data Compression" in "IEEE Transactions on Information Theory“ vorgestellt haben

Der LZ77 Algorithmus war das erste vorgestellte tabellengesteuerte Kompressionsverfahren.

Es komprimierte 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 (bezüglich LZ77) ab. Bei gleichen nah beieinander liegenden Textsequenzen kommt es sehr schnell zur Kompression

OST

Lempel-Ziv Encoder: Grundüberlegung

- **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 oder Wörterbuch gespeichert und codiert werden**
  - **die Grösse des Wörterbuchs und des „look ahead buffers“ muss bestimmt werden**
- **LZ77 wurde 1977 von Jacob Ziv und Abraham Lempel entwickelt**
- ***Problem: „Effiziente Umsetzung des Phrasenspeichers?“***

Lempel-Ziv Encoder: 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 geparst, so kann auf den entsprechenden Knoten des Wörterbuches referenziert werden.**
- **LZ77 wurde 1977 von Jacob Ziv und Abraham Lempel entwickelt.**

Lempel-Ziv Encoder: Umsetzung

- **Die Datenstruktur besteht aus**
  - einem Textfenster, dem **search buffer**
    - hier stehen die schon kodierten Symbole
    - wird dynamisch aufgebaut
  - einem nach vorn gerichteten Puffer **look-ahead buffer** ,
    - Der Puffer zeigt auf die als nächstes zu kodierenden 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.

Lempel-Ziv Encoder: Beispiel

search buffer/Textfenster	Look-ahead buffer/Puffer	Coding Position, Länge, Zeichen
	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	

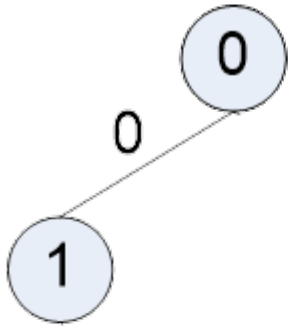
Codierung:

- 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
- Die codierte Sequenz von Triples wird anschliessend übertragen.

OST

Beispiel Lempel-Ziv Encoder für Binärzahlen

0	0	1	1	0	1	0	1	0	1	0	1	1	1	0	1	0	0	1	0

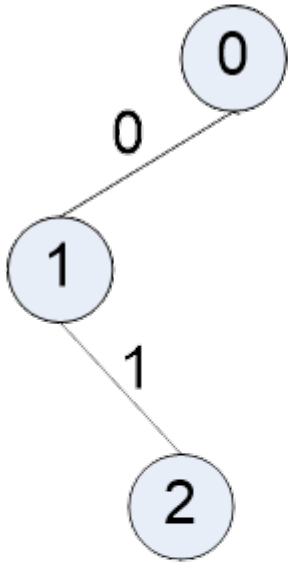


(0,0)

OST

Beispiel Lempel-Ziv Encoder für Binärzahlen

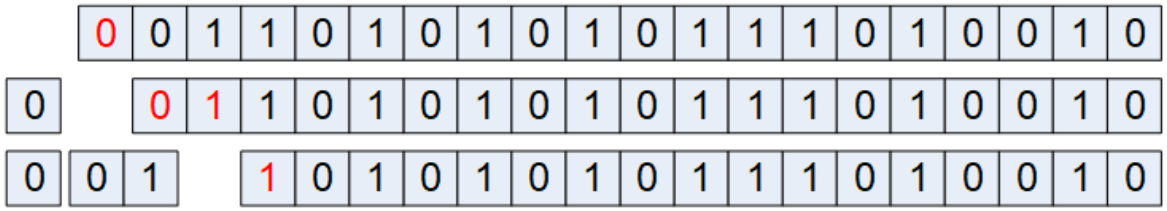
	0	0	1	1	0	1	0	1	0	1	0	1	1	1	0	1	0	0	1	0
0		0	1	1	0	1	0	1	0	1	0	1	1	1	0	1	0	0	1	0

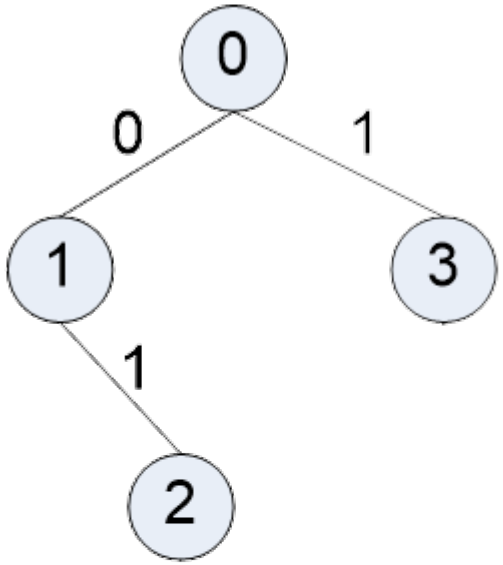


(0,0) (1,1)

OST

Beispiel Lempel-Ziv Encoder für Binärzahlen

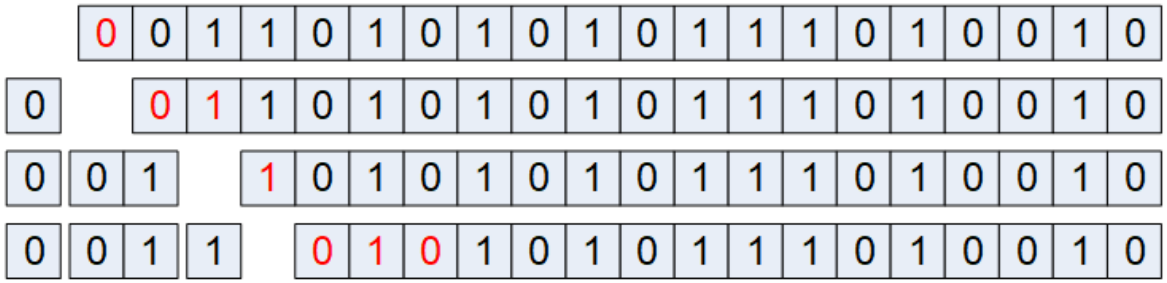


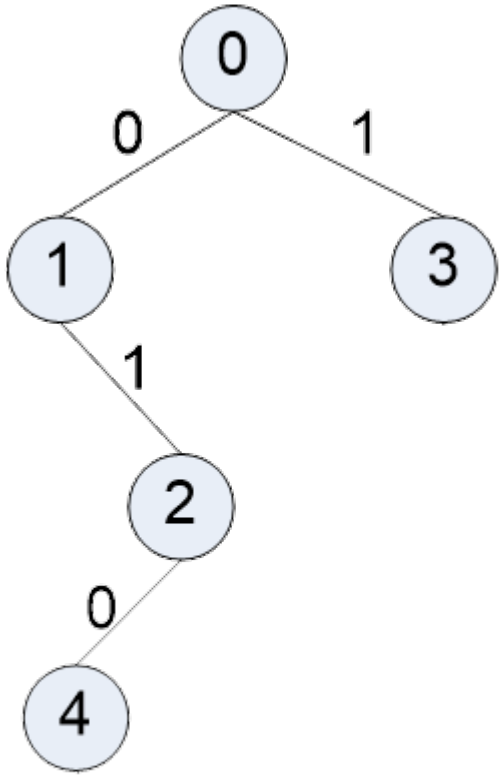


(0,0) (1,1) (0,1)

OST

Beispiel Lempel-Ziv Encoder für Binärzahlen

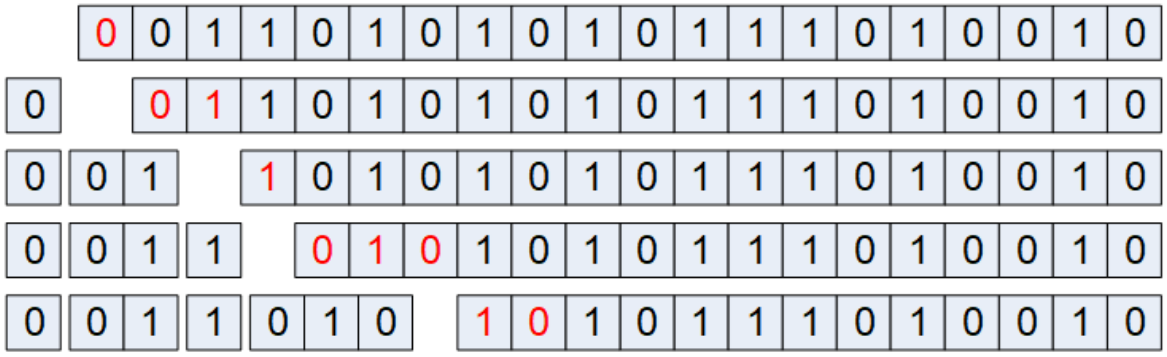


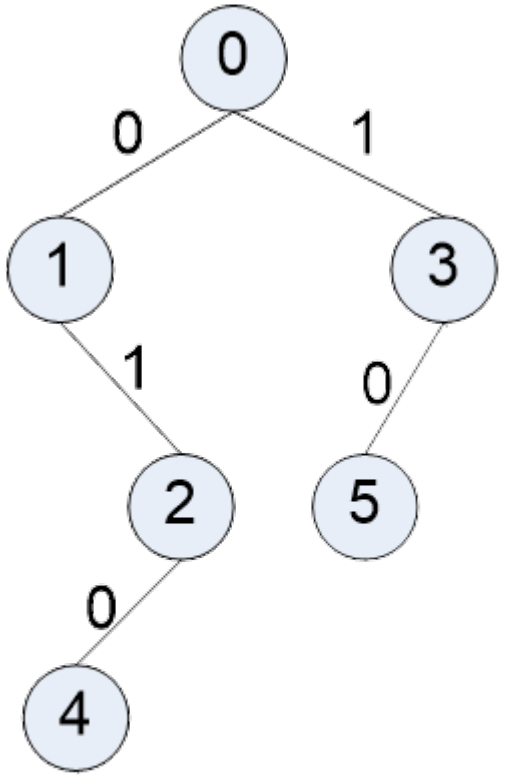


(0,0) (1,1) (0,1)(2,0)

OST

Beispiel Lempel-Ziv Encoder für Binärzahlen

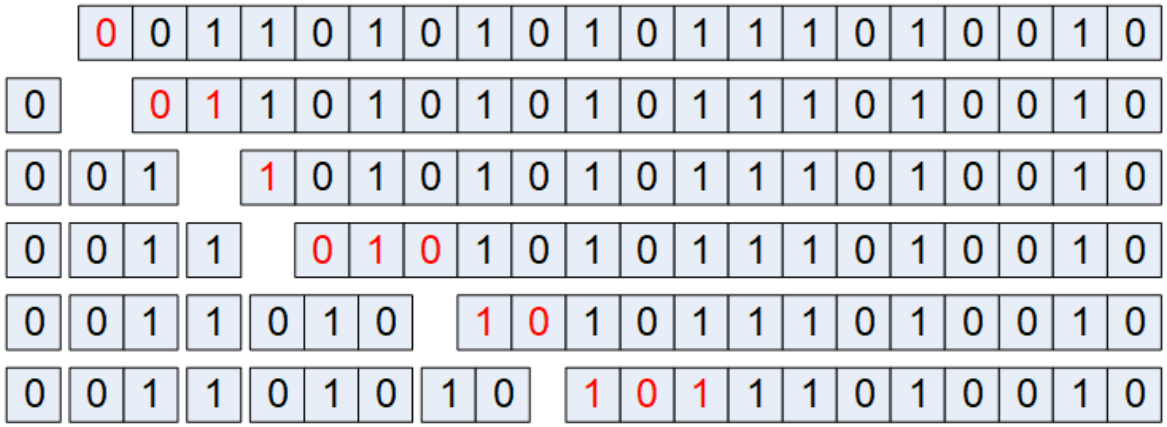


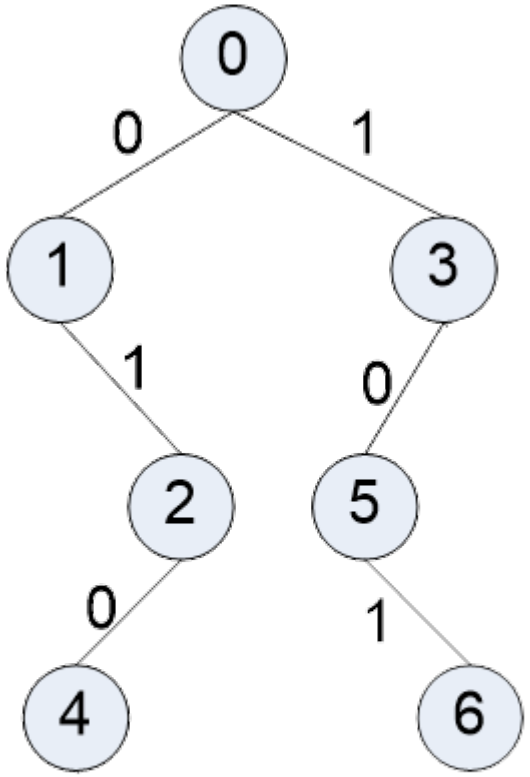


(0,0) (1,1) (0,1)(2,0) (3,0)

OST

Beispiel Lempel-Ziv Encoder für Binärzahlen

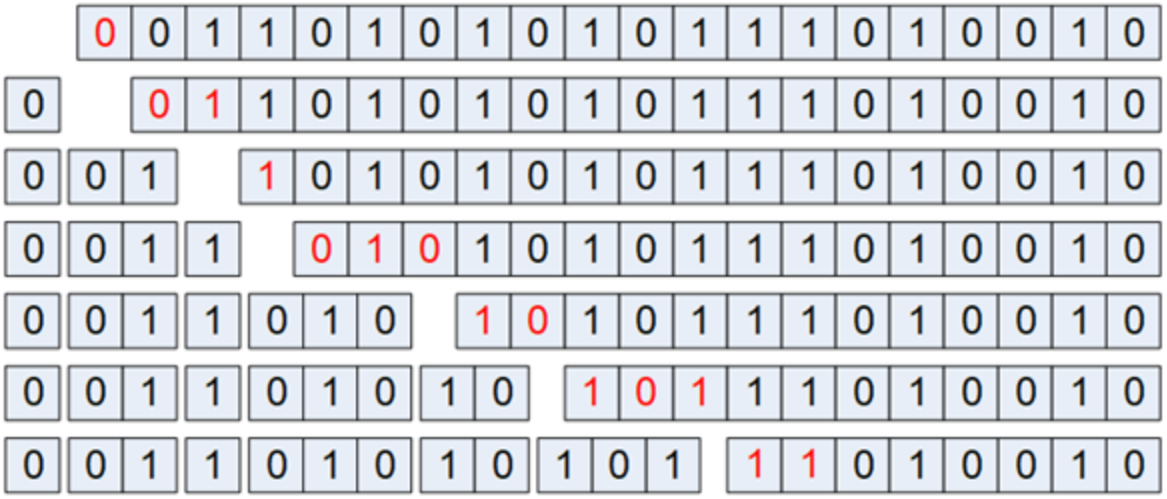


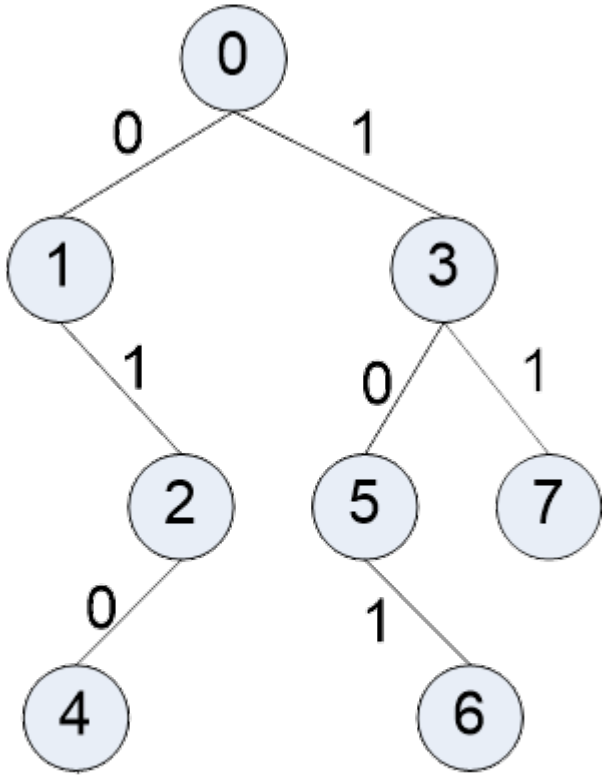


(0,0) (1,1) (0,1)(2,0) (3,0) (5,1)

OST

Beispiel Lempel-Ziv Encoder für Binärzahlen

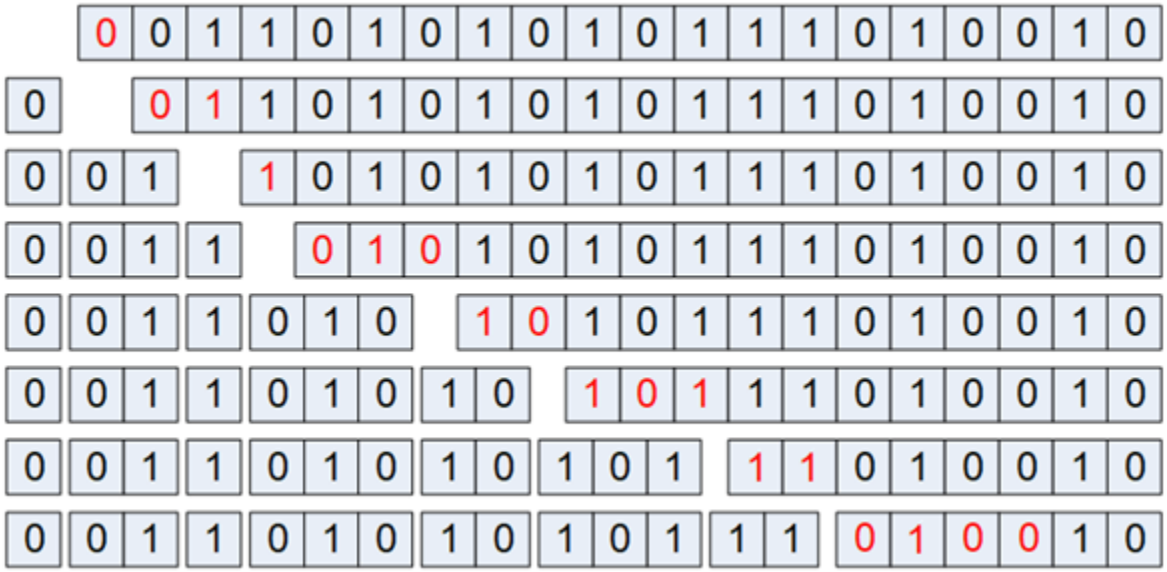


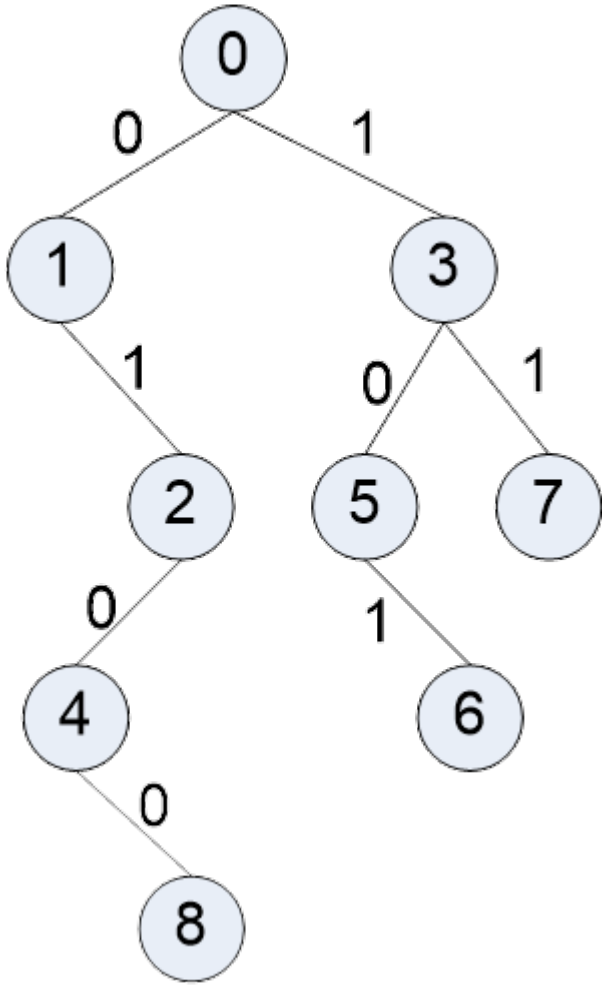


(0,0) (1,1) (0,1)(2,0) (3,0) (5,1) (3,1)

OST

Beispiel Lempel-Ziv Encoder für Binärzahlen

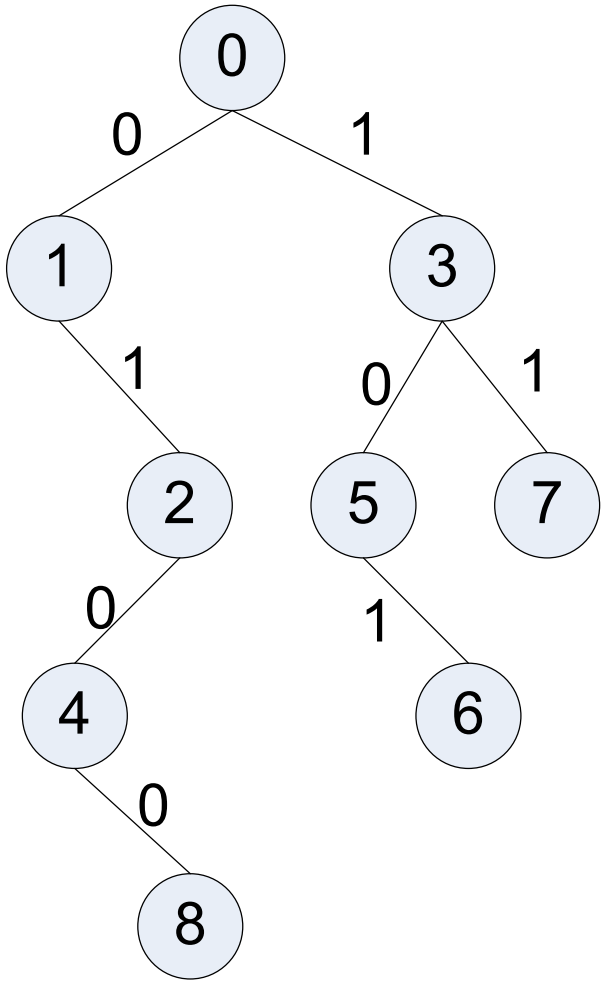




(0,0) (1,1) (0,1) (2,0) (3,0) (5,1) (3,1) (4,0)

OST

Beispiel Lempel-Ziv Encoder für Binärzahlen



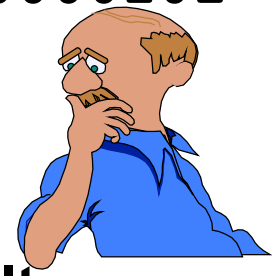
"0" "01" "1" "010" "10" "101" "11" "0100" "10"

(0,0) (1,1) (0,1) (2,0) (3,0) (5,1) (3,1) (4,0) (5,eof)

OST

Komprimierung?

- **Komprimierter Text:** `01100101001101011011110000101`
- **Original:** `00110101010111010010`



- **Voraussetzung: Text muss Regelmässigkeit beinhalten.**
- **Am effizientesten, wenn sich lange Pfade entwickeln, z.B.** `{00000}`

Lempel-Ziv-Welch LZW: Optimierung von LZ77

- **LZW initialisiert ein Wörterbuch mit allen möglichen einzelnen Zeichen (oder Byte-Werten), die im Eingabetext vorkommen können.**
- **Während der Kodierung sucht es nach der längsten Sequenz, die bereits im Wörterbuch vorhanden ist (nicht im Eingabestring wie LZ77), und fügt diese als eine einzige Einheit zum Wörterbuch hinzu.**
- **Wenn die Sequenz gefunden wird, wird sie durch einen einzelnen Code ersetzt, der auf das Wörterbuch verweist.**
- **Neue Sequenzen aus benachbarten Zeichen, die noch nicht im Wörterbuch stehen, werden kontinuierlich hinzugefügt.**
- **Bei LZW werden also Indizes aus dem Wörterbuch übertragen, nicht die direkten Zeiger wie bei LZ77.**
- **Bei LZ77 werden die Puffer begrenzt. Somit kann nicht beliebig weit zurück nach Muster gesucht werden (Search Buffer) bzw. besteht eine Grenze die bestimmt, wie lang die maximal erkennbare wiederkehrende Phrase ist (Look Ahead Buffer)**

Inhalt

Verschlüsselungsverfahren

- **Kryptographie**
- **Historie: Ein kurzer Überblick**
- **Symmetrische Verfahren: eine grobe Übersicht**
  - Ein einfaches Substitutionsverfahren: Der Caesar Chiffre
  - Transpositionsverfahren
  - Vigenère-Chiffre
  - DES
- **Asymmetrisches Verfahren**
  - Modulo Rechnung und inverse Zahlen
  - Eulerfunktion
  - Satz von Euler
  - RSA
    - Euklidscher Algorithmus und Inverser Euklidscher Algorithmus
    - Einsatzmöglichkeiten
    - Grosse Zahlen?

OST

Kryptographie

- **Altgriechisch: krypto für verborgen, geheim und grafie für schreiben, Schrift**
- **Die heutige Veranstaltung gibt nur einen qualitativen und sehr unvollständigen Überblick gängiger Verfahren**
- **Ein bisschen genauer werden wir die Grundlagen des RSA-Verfahrens anschauen**
- **Einen guten Überblick und mehr finden sie auf der Seite "kryptografie.de", ein Teil der hier vorgestellten Inhalte basieren auf Inhalten dieser Seite**
- **Eine vertiefte praktische Anwendung der Verfahren erhalten sie in den Security-Modulen der OST**

OST

Historie: Kryptologie im Altertum

(Quelle: http://kryptografie.de/kryptografie/index.htm)

- Ca. 1500 v. Chr fertigt ein mesopotamischer Töpfer eine Tontafel mit einem Rezept für eine Glasur an, bei dem die Buchstaben verändert und vertauscht waren.
- Ca. 500 v. Chr benutzten die Spartaner eine Skytale, einen Stock, um den ein Stück Leder gewickelt und dann quer über die Wicklungen geschrieben wurde. Abgewickelt war das Leder nicht mehr zu lesen. Es wurde ein Stock mit gleichem Durchmesser benötigt Den Schlüssel.
- Ca. 160 v. Chr. ersann der Grieche Polybios die nach ihm benannte Polybios Chiffre, bei der ein Schlüsselwort in einem 5x5 Quadrat Grundlage der manuellen Verschlüsselung ist.
- Um 60 v. Chr. benutzte Julius Cäsar eine Buchstabenverschiebung um 3 Stellen und machte aus einem A ein D, einem B ein E usw. Dieser Verfahren wurde nach ihm Cäsar-Chiffre benannt.
- Ca. 250 n. Chr. findet sich im indischen Kamasutra auch eine Vorgehensweise zum Verschlüsseln mittels Buchstabenneuordnung.

... ab dem Mittelalter ...

OST

Historie: Kryptologie ab dem Mittelalter

(Quelle: http://kryptografie.de/kryptografie/index.htm)

- Im frühen Mittelalter wurden die Verfahren nicht sehr weiter entwickelt, sondern es wurden eher die vorhandenen Verfahren (einfache Substitution) verwandt.
- Ab dem 15ten Jahrhundert erfuhr die Kryptografie wieder einen Aufschwung. Gerade Italien war hier Vorreiter. So erschuf Leon Battista Alberti (1404–1472) eine Chiffrierscheibe, die das Chiffrieren vereinfachte. Auch Johannes Trithemius (1462–1516) war in dieser Zeit aktiv.
- Im 16ten Jahrhundert entwickelten u. a. Giambattista della Porta (1535-1615) und Blaise de Vigenère (1523–1596) die Substitutionsverfahren (Folie 11) weiter und wandten auch polyalphabetische Verfahren an.
- Im 17. Jh. verfeinerte Johann Franz Graf Gronsfeld zu Bronkhorst und Eberstein (1640-1719) den Vigenère Chiffre zu einem Chiffre, der nach ihm benannt wurde.
- Im 19. Jh. entstand z. B. die Playfair-Chiffre, die auf einem 5x5 Polybios-Quadrat basiert.
- Ende des 19. Jh. / Anfang des 20. Jh. wurde die von Polybios erfundenen 5x5 Quadrate (A bis Z ohne J) immer noch eingesetzt und verfeinert. So erweiterte der Franzose Felix Delastelle 1901 das Verfahren um eine anschließende Transposition und nannte es Bifid. Eine weitere Dimension erhielt es später als Trifid Chiffre. Außerdem erfand er den Four-Square Chiffre mit 4 Polybios-Quadraten.

OST

- **Symmetrische Verfahren: eine grobe Übersicht**
  - Ein einfaches Substitutionsverfahren: Der Caesar Chiffre
  - Transpositionsverfahren
  - Vigenère-Chiffre
  - DES

  - In allen Fällen benötigt man zum Ver- und Entschlüssel den gleichen Schlüssel.
  - Daraus folgt: wollen 100 Mitglieder paarweise geheime Botschaften austauschen, muss für jedes Paar ein eigener Schlüssel erstellt werden.
  - Anzahl der erforderlichen Schlüssel ist dann: $N = \binom{100}{2} = \frac{100*99}{2} = 4950$
  - Ein grosses Problem: das Schlüsselmanagement

Substitutionsverfahren

- **Das einfachste und bekannteste Verfahren ist der sogenannte Caesar Chiffre.**
- **Der Schlüssel ist im gegebenen Beispiel gleich 4. Das heisst, das Chiffrieralphabet wird um 4 Zeichen verschoben.**
- **Symmetrisches Verfahren**

Schlüssel $k = 4$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

Klartext: bald sind sommerferien

Schlüssel $k = 4$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

Chiffretext: feph wmrh wsqqivjivmir

OST

Substitutionsverfahren

- **Auf den ersten Blick sieht das Verfahren ja interessant aus, "feph wmrh wsqqivjivmir" ist schliesslich nicht lesbar!**

Aber:

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

➢ **Was leiten Sie daraus für die Entwicklung von Chiffrieralgorithmen?**

Transpositionsverfahren

Beim Transpositionverfahren werden nach gegebenen Regeln die Zeichenfolge des Klartextes «verwürfelt», d.h. es findet keine Ersetzung (Substitution) der Zeichen statt.

Ein einfaches Beispiel:

Klartext:
DIE WORTE HOER ICH WOHL
ALLEIN MIR FEHLT DER
GLAUBE

Erstellen einer Tabelle zeilenweise

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

Auslesen spaltenweise

Chiffretext:
DTILNHGIECAMLLEHHLITAW
OWLRDUOEOEFEBRRHIERE

Hier sind zusätzlich Permutationen der Spalten möglich!

OST

Polyalphabetisches Verfahren

- **In der ersten Hälfte des 16ten Jahrhunderts entwickeltes polyalphabetisches Verfahren**
- **Nicht mehr Verwürfeln (Transposition) sondern Ersetzen mit wechselnden Alphabeten (also Substitution auf höherem Niveau).**
  - Bei einem einfachen Substitutionsverfahren wird immer dasselbe Alphabet für alle Buchstaben benutzt.
  - Bspw. bei Vigenère dagegen wird je nach Position ein anderes Alphabet verwendet (bestimmt durch das Schlüsselwort).
- **Das Verfahren konnte erst 300 Jahre später von Charles Babbage geknackt werden (1850)**
- **Der erste der Verfahren zum Angriff auf den Code beschrieb war Friedrich Wilhelm Kasiski 1863 (Babbage veröffentlichte seine Technik nicht)**
- **Weitere Infos auch unter: https://de.wikipedia.org/wiki/Vigenère-Chiffre**

Vigenère-Chiffre

CODE
ABBA
CPEE

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Der Algorithmus (Quelle: http://kryptografie.de/kryptografie/chiffre/vigenere.htm)

Klassischerweise gibt es eine feste Verschlüsselungstabelle, auch Tabula recta genannt, die zum händischen Chiffrieren benutzt wird.
Sie funktioniert so: man sucht den Klartextbuchstaben links in der 1. Spalte und geht in dieser Zeile soweit nach rechts, bis man in der obersten Zeile den Buchstaben des Schlüssels gefunden hat. Nun kann man dort den Chiffre-Buchstaben ablesen. Praktikablerweise schreibt man das Chiffrat neben die Tabelle, damit man immer einen Überblick hat, in welcher Zeile man ist.

Mathematisch gesehen kann man aber auch mit Offsets rechnen, die sich aus dem Schlüssel ergeben. Dabei hat A den Wert 0, B den Wert 1 usw. Der Offset wird dem Klartextbuchstaben dann hinzuaddiert, um den Chiffratbuchstaben zu erhalten. Ist das Alphabet zu Ende, wird wieder bei A begonnen.

Wenn das Schlüsselwort kürzer als der Klartext ist, wird es mehrmals hintereinander geschrieben.

Die Polygraphiae des Johannes Trithemius (1508) enthält die erste Darstellung der Tabula recta

OST

DES: Data Encrcryption Standard

- **Der DES-Algorithmus**

  wurde als offizieller Standard für die US-Regierung im Jahr 1977 bestätigt und wird seither international vielfach eingesetzt

- **Seine Entstehungsgeschichte**

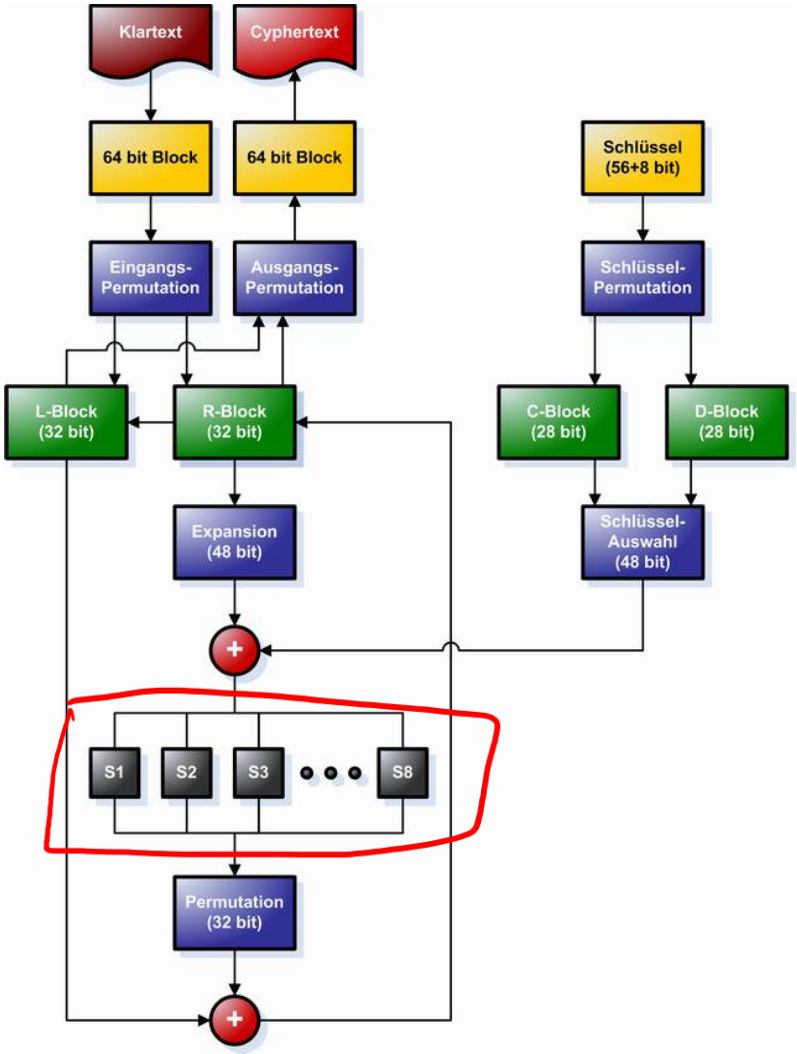
  hat wegen der Beteiligung der NSA am Design des Algorithmus immer wieder Anlass zu Spekulationen über seine Sicherheit gegeben

- **Heute**

  wird DES aufgrund der verwendeten Schlüssellänge von nur 56 Bits für viele Anwendungen als nicht ausreichend sicher erachtet.

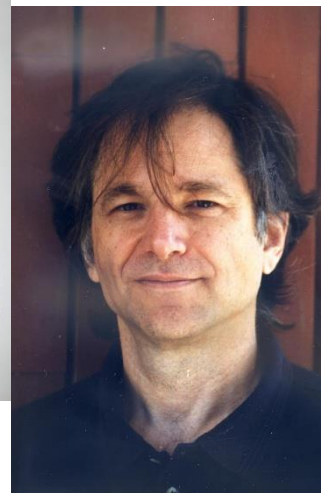
- Quelle: https://de.wikipedia.org/wiki/Data_Encryption_Standard

AES → variable Schlüssellänge



IDEE: Gibt es ein asymmetrisches Kryptoverfahren?

- **Das heisst, lässt sich ein Verfahren finden, dass es erlaubt**
  - einen Schlüssel zur Codierung und
  - einen zweiten Schlüssel zur De-Codierung benutzen?
- **Als Rahmenbedingung ist es weiterhin erforderlich, dass die Kenntnis eines Schlüssels nichts über die Identität des anderen "Partner"-Schlüssels verrät.**



Ursprünglich sind die drei Mathematiker ***Rivest***, ***Shamir*** und ***Adleman*** **(RSA)** angetreten, um zu zeigen, dass das **nicht** möglich ist.

Aber: Sie haben gezeigt, dass ein asymmetrisches Kryptoverfahren möglich ist (heute auch bekannt als RSA-Verfahren).

O S T

↓ ↓ ↓

14 18 19

$A=0, B=1, \ldots, Z=25, \_=26$

$\Rightarrow$ 27 Zeichen

Cäsar: $c = t + k \mod m$

$t = c - k \mod m$

Hill: $c = t \cdot k \mod m$

$k = 4$

$m = 27$

$k \cdot k^{-1} \mod m = 1$

$4 \cdot 7 \mod 27 = 1$

$t = c \cdot ? \mod m$

O: $14 \cdot 4 \mod 27 = 2$

S: $18 \cdot 4 \mod 27 = 18$

T: $19 \cdot 4 \mod 27 = 22$

$$\left(t^{k}\right)^{k^{-1}} = t^{\overbrace{k \cdot k^{-1}}^{\sim 1}}$$

OST

$4 \cdot ? \bmod 27 = 1$

Voraussetzung: $\checkmark$ $ggT(4, 27) = 1 = \underset{?}{r} \cdot 4 + s \cdot 27$

EEA: $27 = 6 \cdot 4 + 3$ ←

$4 = 1 \cdot 3 + \boxed{1}$

$1 = 4 - 1 \cdot 3$

$1 = 4 - 1 \cdot (27 - 6 \cdot 4)$

$1 = 4 - 1 \cdot 27 + 6 \cdot 4$

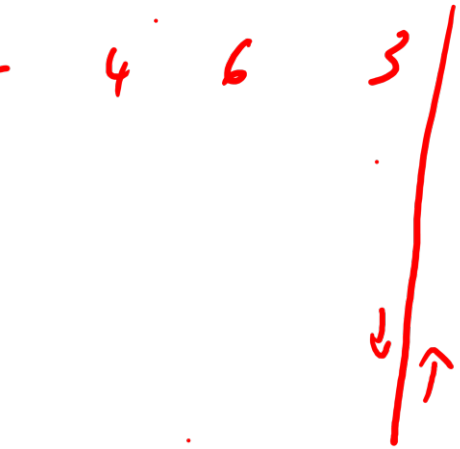
$\boxed{1 = 7 \cdot 4 - 1 \cdot 27}$

r s

$27 : 4 = 6 \quad R\ 3$

$\Rightarrow 27 = 6 \cdot 4 + 3$

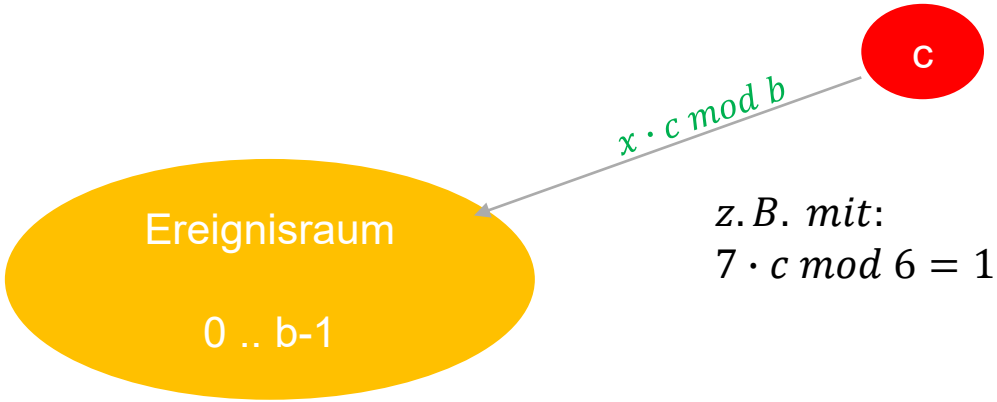
27 4 6 3



OST

Inverse Zahlen: Was ist eine inverse Zahl?

- **Im Reellen ist a^{-1} die zu a inverse Zahl, da die Operation $a \cdot a^{-1} = a^0 = 1$ ergibt.**
- **Durch mod b wird ein Ergebnisraum von 0 … b-1 möglich, d.h. es lässt sich eine Zahl c (ausserhalb des Ergebnisraumes) finden, die in dieser Rechenvorschrift eine inverse Zahl zu a ist.**
- **Bemerkenswert ist, dass sich die Zahl c nicht unmittelbar oder einfach aus der Zahl a ergibt.**
- **Ein Ansatz für ein asymmetrisches Verfahren?**



OST

Inverse Zahlen: Beispiel

Für zwei teilerfremde Zahlen a, b ($ggT = 1$) existiert eine Zahl c, so das gilt:

$$\boldsymbol{a \cdot c \; mod \; b = 1}$$

Beispiel: Sei $b = 6$ und $a = 7$ wobei a und b teilerfremd sind, dann ergibt die Rechnung

für den Fall, dass $c = 13$ ist.

$$\boldsymbol{7 \cdot c \; mod \; 6 = 1}$$

OST

Eulerfunktion

Die Eulerfunktion gibt die Anzahl der zu einer Zahl n teilerfremden Zahlen an, d.h. die Zahlenpaare (n, x) mit $x < n$, die keinen gemeinsamen Teiler haben.

$$\varphi(n) = \sum_{\substack{1 \le a \le n \\ \mathrm{ggT}(a,n)=1}} 1$$

$\boldsymbol{\Phi(n)}$ **= Anzahl der relativ primen Zahlen zu** n

$\boldsymbol{\Phi(18) = 6}$ ⟶ 18 ist teilerfremd zu 1, 5, 7, 11, 13, und 17

Für Primzahlen p gilt:

$\boldsymbol{\Phi(p) = p - 1}$ ⟶ $\boldsymbol{\Phi(5) = 4}$ 5 ist teilerfremd zu 1, 2, 3 und 4

Für das Produkt der Primzahlen $p \cdot q$ gilt:

$\boldsymbol{\Phi(p \cdot q) = (p - 1) \cdot (q - 1)}$ ⟶

$= \varphi(p) \cdot \varphi(q) = (p-1) \cdot (q-1)$

Beispiel:
Mit $p = 5$, $\Phi(5) = 4$ und
$q = 3$, $\Phi(3) = 2$ ergibt sich
$\Phi(15) = 4 * 2 = 8$

OST

Satz von Euler

***Satz von Euler*: Für zwei teilerfremde Zahlen a und b gilt: $a^{\Phi(b)} \; mod \; b = 1$.**

Beispiel: Sei $b = 5$, dann ist a für die Werte $1, 2, 3, 4$ teilerfremd zu b oder relativ prim.

- $1^4 \; mod \; 5 = 1$
- $2^4 \; mod \; 5 = 16 \; mod \; 5 = 1$
- $3^4 \; mod \; 5 = 81 \; mod \; 5 = 1$
- $4^4 \; mod \; 5 = 256 \; mod \; 5 = 1$
- $\mathbf{5}^4 \; mod \; 5 = \mathbf{625} \; mod \; 5 = \mathbf{0}$
- $6^4 \bmod 5 = 1296 \bmod 5 = 1$
- …
- $\mathbf{10}^4 \bmod 5 = \mathbf{10000} \bmod 5 = \mathbf{0}$

➢ Solange der Wert a also kleiner als b ist, gilt die Aussage immer.

***Satz von Euler*:**

Für zwei teilerfremde Zahlen a und b gilt: $a^{\Phi(b)}\ mod\ b = 1$.

Für b können wir jetzt zwei Primzahlen p, q wählen, dann folgt:

$$\boldsymbol{a^{\Phi(p\cdot q)}\ mod\ (p\cdot q) = 1}$$

$$\boldsymbol{a^{(p-1)\cdot(q-1)}\ mod\ (p\cdot q) = 1}$$

Und das gilt in jedem Fall, solange a < pq erfüllt ist!

OST

Es galt: nach dem Satz von Euler: ist b das Produkt zweier Primzahlen, dann gilt:

$$a^{\Phi(b)} \bmod b = 1$$

Jetzt zeigen wir noch das: $a^y \bmod (p \cdot q) = a^{y \bmod \Phi(p \cdot q)} \bmod (p \cdot q)$

d.h. wir können den Exponenten y modulo $\Phi(p \cdot q)$ kürzen, ohne dass sich das Ergebnis ändert.

$$\begin{aligned} a^y \bmod (p \cdot q) &\stackrel{!}{=} a^{y \bmod \cdot \Phi(p \cdot q)} \bmod (p \cdot q) \\ &= a^{y - n \cdot \Phi(p \cdot q)} \bmod (p \cdot q) \\ &= a^y \cdot a^{-n \cdot \Phi(p \cdot q)} \bmod (p \cdot q) \\ &= [a^y \bmod (p \cdot q)] \cdot \left[a^{\Phi(p \cdot q)} \bmod (p \cdot q)\right]^{-n} \\ &= a^y \bmod (p \cdot q) \quad \text{q.e.d.} \end{aligned}$$

OST

RSA Grundlagen: Interpretation

Es gilt:

- $\boldsymbol{a^y \bmod (p \cdot q) = a^{y \bmod \Phi(p \cdot q)} \bmod (p \cdot q)}$ (mit a teilerfremd zu $p \cdot q$).
- Nehmen wir an, wir haben das Codewort a wobei |a| < pq ist
- Setzen wir y = e*d wobei "e" der encryption- und "d" der decryption Schlüssel sei

Dann folgt

- $a^{ed} \bmod (p \cdot q) = a^{e*d \bmod \Phi(p \cdot q))} \bmod (p \cdot q)$
- Wenn wir jetzt noch e und d so bestimmen, dass gilt: $1 = ed \bmod \Phi(p \cdot q)$ gilt, dann

- Können wir mit e verschlüsseln $a \succ a^e \bmod (pq)$ und später
- mit d entschlüsseln $\succ a^{ed} \bmod (pq) = a$, da ja $e * d \bmod \Phi(p \cdot q) = 1$ wird!
- D.h. wir haben jetzt einen Schlüssel zum Zuschliessen und

  einen Schlüssel zum Aufschliessen, ein **asymmetrisches Verfahren**

OST

Beispiel

Seien die beiden Primzahlen $p = 47$, $q = 59$ gegeben.

Dann folgt daraus: $n = p \cdot q = 2773$ und $\Phi(n) = 2668$.

- **Wir wählen den Wert $e = 17$, relativ prim zu $\Phi(n)$, $1 < e < \Phi(n)$.**
  - öffentlicher Schlüssel: (e, n)
- **Wir berechnen den zu e inversen Wert d mit 157.**
  - privater Schlüssel: (d, n)

Encrypt:	m	$\rightarrow$	m^e	$\rightarrow$ $c = m^e \bmod n$
	12		**2218611106…**	**336**
Decrypt:	c	$\rightarrow$	c^d	$\rightarrow$ $m = c^d \bmod n$
	336		**4317840049…**	**12**

***Problem*: Wie kann der zu e inverse Wert d berechnet werden?**

OST

Euklidischer Algorithmus: Basis zur Berechnung des inversen Wertes d

Der euklidsche Algorithmus dient zur Bestimmung des grössten gemeinsamen Teilers zweier Zahlen oder Polynome.

Sei r_0 und r_1 gegeben, dann ergibt sich der $ggT\ (r_0, r_1) = r_n$:

- $r_0 = q_1 \cdot r_1 + r_2$
- $r_1 = q_2 \cdot r_2 + r_3$
- $r_2 = q_3 \cdot r_3 + r_4$
- $r_{n-2} = q_{n-1} \cdot r_{n-1} + r_n$
- $r_{n-1} = q_n \cdot r_n + 0$

Beispiele: Wenden Sie das Verfahren an auf die Zahlenpaare $(21, 9)$ und $(48, 18)$ und $(19{,}13)$

OST

Ausgangslage für die Berechnung des inversen Wertes d

- **Ausgangslage: Bestimme d mit $e \cdot d \equiv 1 \; mod \; \Phi(n)$**
  - $\Leftrightarrow \; \Phi(n) \; teilt \; (e \cdot d - 1)$
  - $\Leftrightarrow \; \exists \; q \; in \; \mathbb{Z} \; mit \; e \cdot d - 1 = \Phi(n) \cdot q$
  - $\Leftrightarrow e \cdot d - q \cdot \Phi(n) = 1 \quad (zu \; lösende \; Linearkombination, geg: e, \Phi(n), ges: d)$
  - Anwendung: erweiterter euklidischer Algorithmus

  - Beispiel: e=17, $\Phi(n)$=2668, ggT(17,2668)=1
  - Euklidischer Algorithmus:
    - $2668 = 156 \cdot 17 + 16$
    - $17 = 1 \cdot 16 + 1$ ←
    - $16 = 16 \cdot 1 + 0$

  - Erweiterter Euklidischer Algorithmus berechnet nun rückwärts die Linearkombinationen

$1 = 17 - 1 \cdot 16 = 17 - 1(2668 - 156 \cdot 17)$

$= 17 - 2668 + 156 \cdot 17$

$= 157 \cdot 17 - 1 \cdot 2668$

d

OST

Gesucht: $20 \cdot d \; mod \; 43 = 1 \; (oder \; 20 \cdot d \equiv 1 \; mod \; 43)$

$ggt(20, 43) = 1$:

$r_0 = 43 = 2 \cdot 20 + 3$

$r_1 = 20 = 6 \cdot 3 + 2$

$r_2 = 3 = 1 \cdot 2 + 1$

$r_3 = 2 = 2 \cdot 1 + 0$

$1 = 3 - 1 \cdot 2$

$1 = 3 - 1 \cdot (20 - 6 \cdot 3) = 3 - 1 \cdot 20 + 6 \cdot 3$

$= -1 \cdot 20 + 7 \cdot 3$

$1 = -1 \cdot 20 + 7 \cdot (43 - 2 \cdot 20)$

$= -1 \cdot 20 + 7 \cdot 43 - 14 \cdot 20 = 7 \cdot 43 - 15 \cdot 20$

$1 = 7 \cdot 43 - 15 \cdot 20$

Gefragt war d damit gilt: $\mathbf{20 \cdot b \; mod \; 43 = 1}$

also: $(7 \cdot 43 - 15 \cdot 20) mod \; 43 = 1$

Dabei gilt:

- $(7 \cdot 43 \; mod 43 - 15 \cdot 20 \; mod \; 43) = 1$
- $(0 \quad - 15 \cdot 20 \; mod \; 43) = 1$

- D.h., die zu 20 inverse Zahl heisst -15 mod 43 = 28.

OST

Beispiel 2

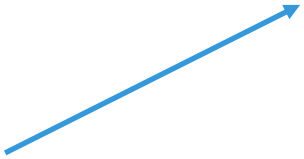
Seien die beiden Primzahlen $p = 11$, $q = 7$ gegeben.
Dann folgt daraus $n = p \cdot q = 77$ und $\Phi(n) = \Phi(10 \cdot 6) = 60$.

- Wir bestimmen den Wert $e = 17$ mit $1 < e < \Phi(n)$, relativ prim zu $\Phi(n)$.

$$60 = 3 \cdot 17 + 9$$

$$17 = 1 \cdot 9 + 8$$

$$9 = 1 \cdot 8 + 1$$



$$1 = 9 - 1 \cdot 8$$

$$1 = 9 - 1 \cdot (17 - 1 \cdot 9) = 9 - 17 + 9 = -17 + 2 \cdot 9$$

$$1 = -17 + 2 \cdot 9 = -17 + 2 \cdot (60 - 3 \cdot 17)$$
$$= -17 + 2 \cdot 60 - 6 \cdot 17 = 2 \cdot 60 - 7 \cdot 17$$
$$1 = 2 \cdot 60 - 7 \cdot 17$$

Das heisst $(-7)\ mod\ 60 = 53$ ist der zu 17 inverse Schlüssel.

OST

Erweiterter Euklidischer Algorithmus tabellarisch

Gesucht: $20 \cdot d \; mod \; 43 = 1 \; (oder \; 20 \cdot d \equiv 1 \; mod \; 43)$

$ggt(20, 43) = 1$:

$r_0 = 43 = 2 \cdot 20 + 3$

$r_1 = 20 = 6 \cdot 3 + 2$

$r_2 = 3 = 1 \cdot 2 + 1$

$r_3 = 2 = 2 \cdot 1 + 0$

$1 = 3 - 1 \cdot 2$

$1 = 3 - 1 \cdot (20 - 6 \cdot 3) = 3 - 1 \cdot 20 + 6 \cdot 3$

$= -1 \cdot 20 + 7 \cdot 3$

$1 = -1 \cdot 20 + 7 \cdot (43 - 2 \cdot 20)$

$= -1 \cdot 20 + 7 \cdot 43 - 14 \cdot 20 = 7 \cdot 43 - 15 \cdot 20$

$1 = 7 \cdot 43 - 15 \cdot 20$

$ggT(20,43) = 1 = x \cdot a + y \cdot b = x \cdot 43 + y \cdot 20$

a	b	q	r	x	y	Probe
43	20	2	3	7	-1-2*7=-15	1=7*43+(-15)*20
20	3	6	2	-1	1-6*(-1)=7	1=-1*20+7*3
3	2	1	1	1	$x_{i+1} - q_i * y_{i+1}$ 0-1*1 = -1	1=1*3+(-1)*2
2	1	2	0	0	1	ggT=x*a+y*b 1=0*2+1*1

OST

Ausgangslage für die Berechnung des inversen Wertes d

- **Ausgangslage: Bestimme d mit $\boldsymbol{e \cdot d \equiv 1\ mod\ \Phi(n)}$**
  - Beispiel: e=17, $\Phi(n)$=2668, ggT(17,2668)=1
  - Euklidischer Algorithmus:
    - $2668 = 156 \cdot 17 + 16$
    - $17 = 1 \cdot 16 + 1$
    - $16 = 16 \cdot 1 + 0$
  - Erweiterter Euklidischer Algorithmus berechnet nun rückwärts die Linearkombinationen
    - $1 = 17 - 1 \cdot 16 = 17 - 1 \cdot (2668 - 156 \cdot 17) = 17 - 2668 + 156 \cdot 17$

      $= -2668 + 157 \cdot 17$
  - Das heisst 157 ist der zu 17 inverse Schlüssel.

OST

Beispiel

Seien die beiden Primzahlen $p = 47$, $q = 59$ gegeben.

Dann folgt daraus: $n = p \cdot q = 2773$ und $\Phi(n) = 2668$.

- **Wir wählen den Wert $e = 17$, relativ prim zu $\Phi(n)$, $1 < e < \Phi(n)$.**
  - öffentlicher Schlüssel: (e, n)
- **Wir berechnen den zu e inversen Wert d mit 157.**
  - privater Schlüssel: (d, n)

↳ mod. Inv. von e mod φ(n)

Encrypt:	m	$\rightarrow$	m^e	$\rightarrow$ $c = m^e \bmod n$
	12		**2218611106…**	**336**
Decrypt:	c	$\rightarrow$	c^d	$\rightarrow$ $m = c^d \bmod n$
	336		**4317840049…**	**12**

OST

Grosse Zahlen

Wahrscheinlichkeit

- **Für 6 richtige im Lotto:** $7.1 \cdot 10^{-8}$
- **Jährlich vom Blitz getroffen zu werden:** 10^{-7}
- **Von einem Meteoriten erschlagen zu werden:** $16 \cdot 10^{-12}$

Anzahl Atome

- **Erde:** 10^{51}
- **Sonne:** 10^{57}
- **Unsere Galaxis:** 10^{67}
- **Im Weltall (ohne dunkle Materie):** 10^{77}

Die Zeit bis

- **zur nächsten Eiszeit:** 14000 Jahre
- **die Sonne zur Nova wird:** 10^{9} Jahre
- **Alter des Universums:** 10^{10} Jahre

Lebensdauer

- **des Weltalls (falls geschlossen):** 10^{15}Jahre
- **des Weltalls (falls offen):** 10^{19} Jahre

Ein Computer, der pro Sekunde $2.000.000.000 = 2 \cdot 10^{9}$ AES Verschlüsslungen berechnen kann, benötigt zum Durchprobieren aller 2^{128} Schlüssel ca. $5.3 \cdot 10^{21}$ Jahre!

Grosse Zahlen im Zusammenhang mit RSA

- **1024-Bit RSA-Schlüssel**
  - Zwei Primzahlen mit ca. 512Bits
  - **Anzahl** Primzahlen (Abschätzung mit dem Primzahlsatz): $\frac{2^{512}}{\ln(2^{512})} \approx 3.77 \times 10^{151}$
- **2048-Bit RSA-Schlüssel**
  - Zwei Primzahlen mit ca. 1024Bits
  - **Anzahl** Primzahlen (Abschätzung mit dem Primzahlsatz): $\frac{2^{1024}}{\ln(2^{1024})} \approx 3.53 \times 10^{305}$
- **4096-Bit RSA-Schlüssel**
  - Zwei Primzahlen mit ca. 2048Bits
  - **Anzahl** Primzahlen (Abschätzung mit dem Primzahlsatz): $\frac{2^{2048}}{\ln(2^{2048})} \approx 2.27 \times 10^{613}$

OST

- **Wenn ein Computer pro Sekunde 100×10^9 Faktorisierungen ausprobieren kann**
  - Die Gesamtzahl der möglichen Kombinationen von zwei Primzahlen aus 3.77×10^{151} Primzahlen ist gegeben durch die Kombination ohne Wiederholung (da die Reihenfolge nicht wichtig ist und immer zwei verschiedene Primzahlen verwendet werden):
    $\frac{1}{2} \times (3.77 \times 10^{151}) \times (3.77 \times 10^{151} - 1) \approx \frac{1}{2} \times (3.77 \times 10^{151})^2$
    Der Faktor ½ ist da, weil das Produkt $p \cdot q$ das gleiche ist wie $q \cdot p$, also muss jede Kombination nur einmal betrachtet werden.
  - $T = \frac{Kombinationen}{Faktorisierungen\ pro\ Sekunde} = \frac{\frac{1}{2} \times (3.77 \times 10^{151})^2}{100 \times 10^9} = \frac{(3.77 \times 10^{151})^2}{200 \times 10^9} = \frac{3.77^2 \times 10^{302}}{200 \times 10^9} = \frac{3.77^2}{2} \times \frac{10^{302}}{10^{11}} \approx 7.1 \times 10^{291}$
  - $\frac{7.1 \times 10^{291} Sekunden \times Jahre}{3.154 \times 10^7 Sekunden} \approx 2.25 \times 10^{283} Jahre$
  - Bei 2048-Bit: $\approx 6.23 \times 10^{599} Sekunden \approx 1.97 \times 10^{592} Jahre$
  - Bei 4096-Bit: $\approx 2.57 \times 10^{1215} Sekunden \approx 8.16 \times 10^{1207} Jahre$

OST

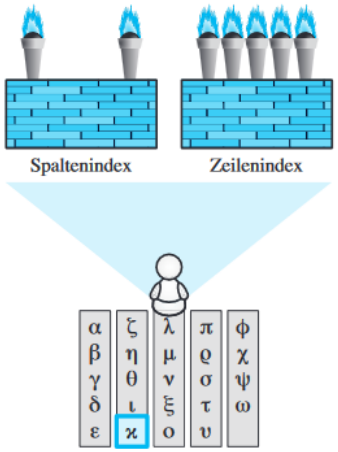
Motivation

Troja, Agamemnon (siehe griechische Mythologie)

- **Leuchtfeuer**
  - (Einführung in die Informations- und Codierungstheorie – Dirk W. Hoffmann – S.15ff)

3. Punischer Krieg

- **Polybius: technisch wissenschaftlicher Berater von Scipio**
  (1. nachweisliche Veröffentlichung zur Codierungs- und Protokollproblematik)
- **Nachrichtensystem mit 5 Bit Wortlänge**
  - (Einführung in die Informations- und Codierungstheorie – Dirk W. Hoffmann – S.18)



OST

Motivation

Bis der Kunstmaler Samuel Morse 1837 seinen ersten Apparat zur elektromagnetischen Datenübertragung vorführt gibt es keine wesentlichen Verbesserungen.

- (Einführung in die Informations- und Codierungstheorie – Dirk W. Hoffmann – S.33ff)

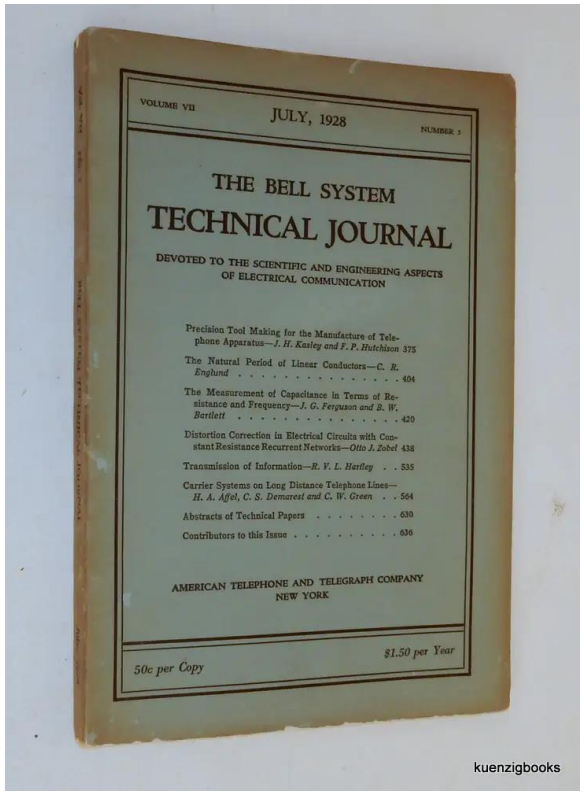
Anekdote Rothschild zur Bedeutung der Sicherheit in der Informationsübertragung

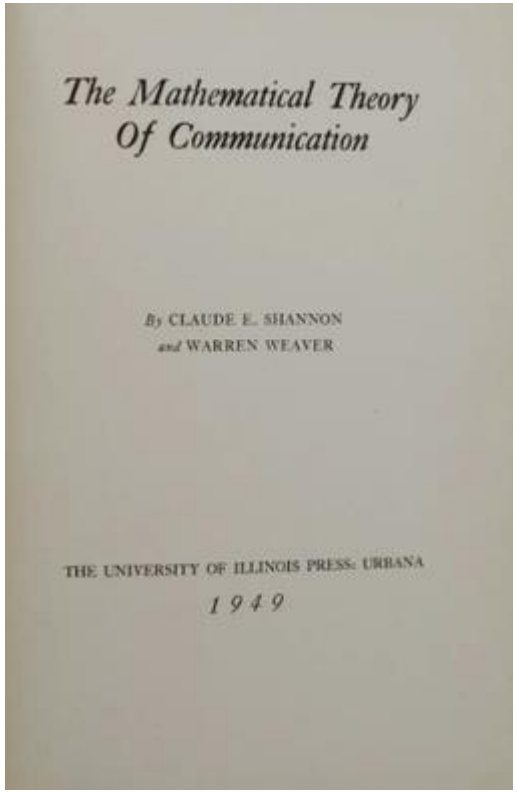
- (https://en.wikipedia.org/wiki/Nathan_Mayer_Rothschild#Waterloo_legend)

Motivation

Grundlegende Veröffentlichungen zur technischen Informationstheorie von:

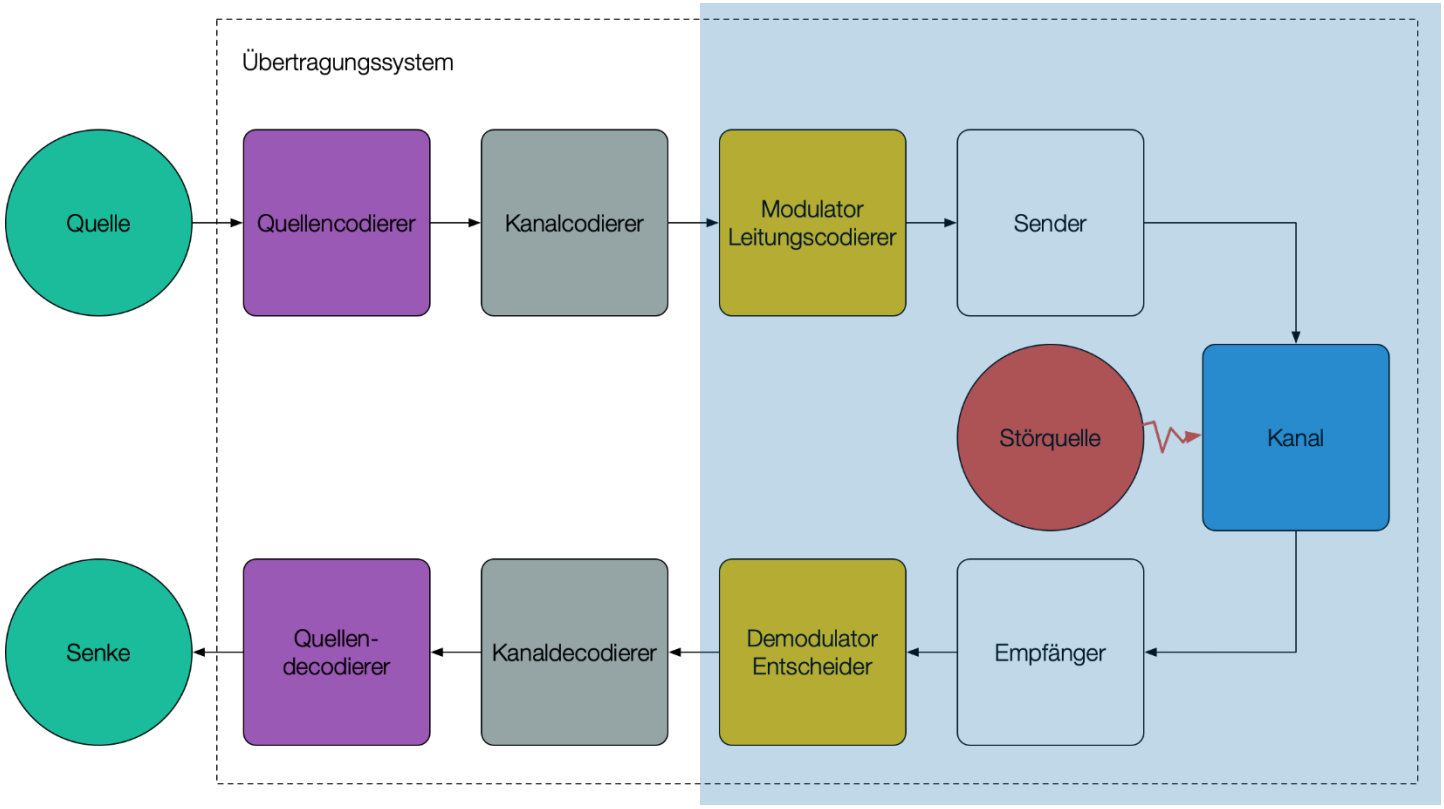
- **Hartly, 1928: *Transmission of Information***
- **Shannon, 1948: *A Mathematical Theory of Communication***





OST

Modell der Informationsverarbeitung

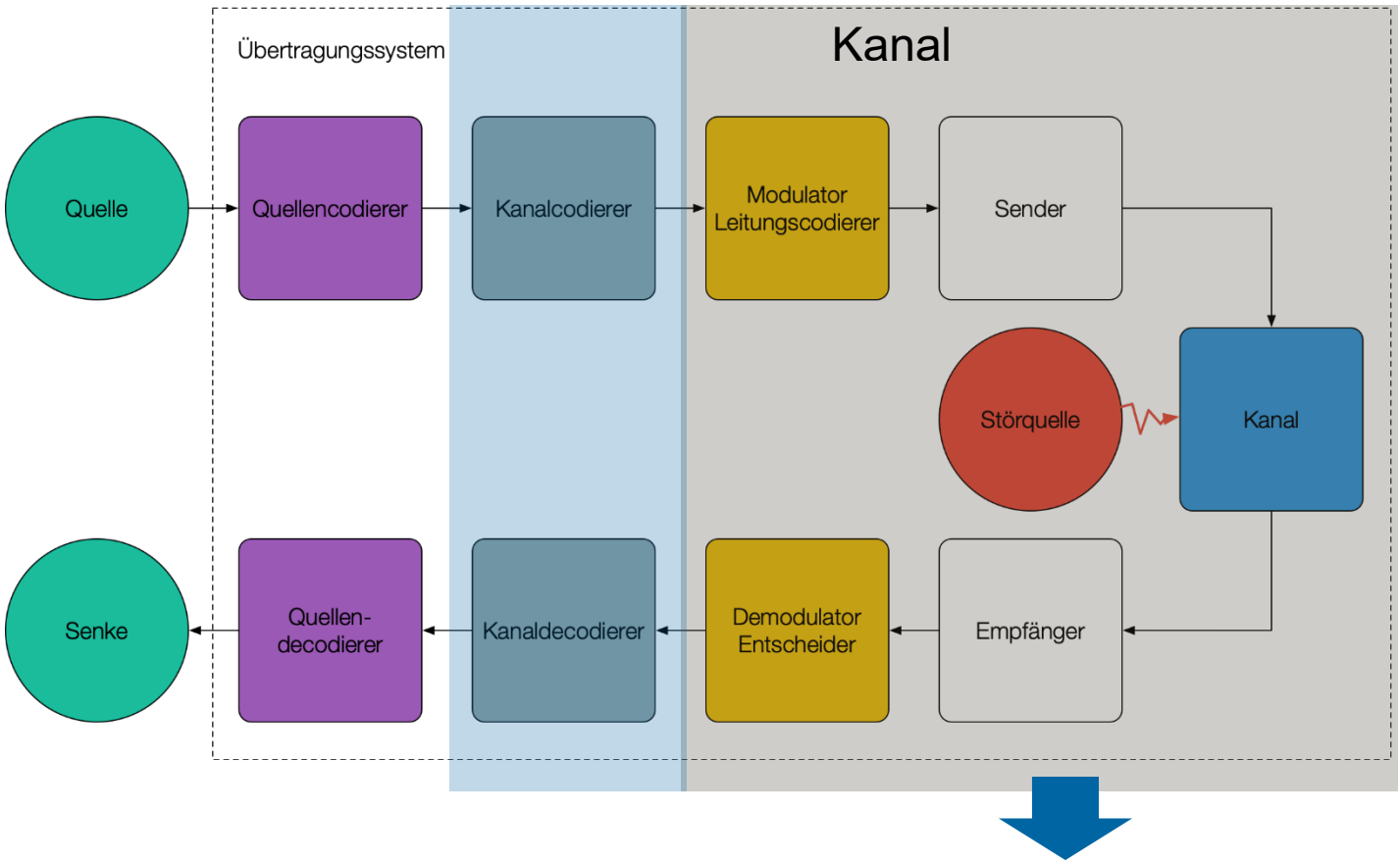


OST

Kanalmodell

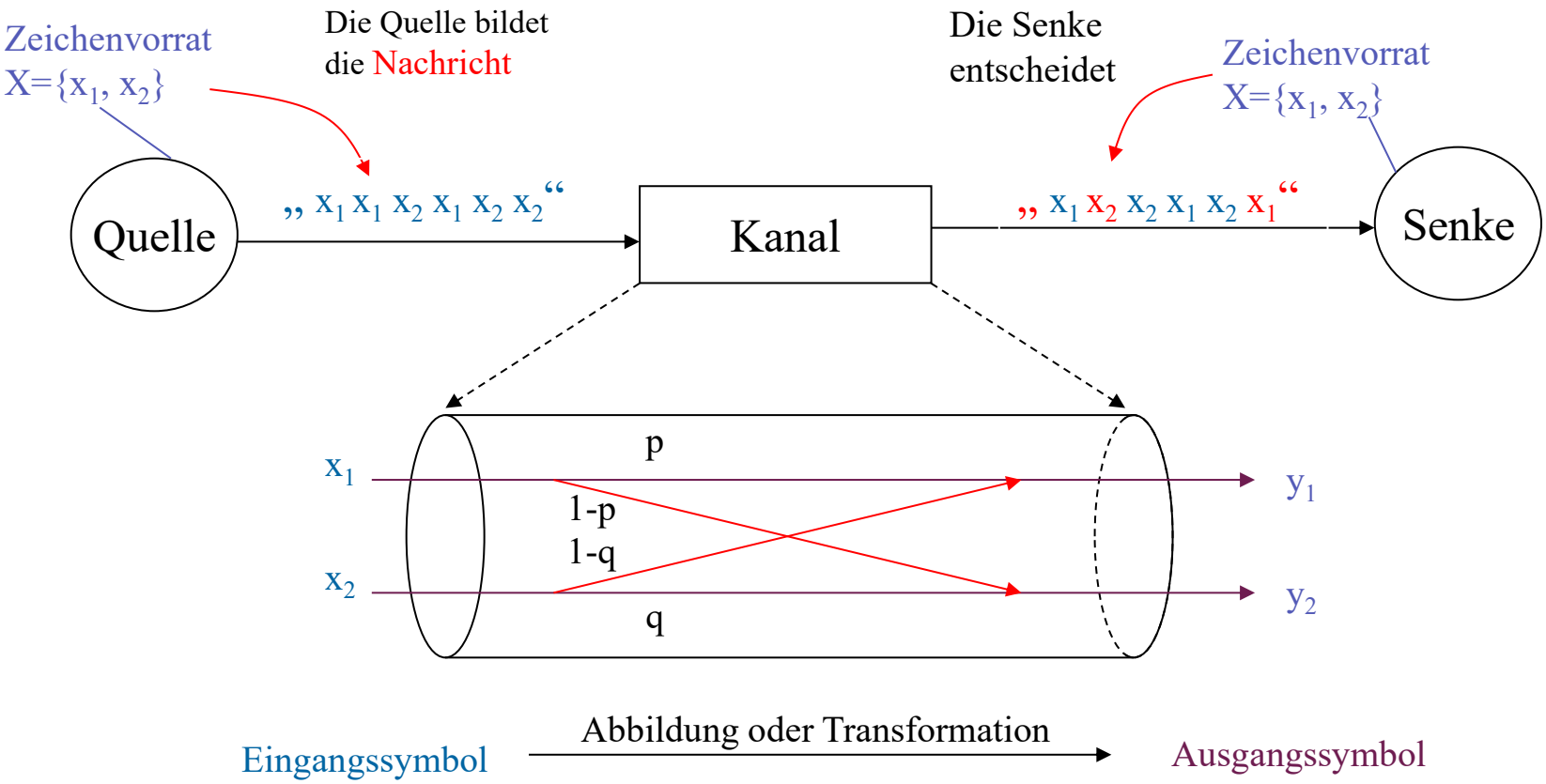
- **Was ist ein abstrakter Kanal?**
- **Kanalmatrix**
- **Maximum Likelihood**
- **Transinformation**

Modell der Informationsverarbeitung



Um einen Kanalkodierung zu erstellen, brauchen wir ein **abstraktes Modell** des Kanals!

Kanalmodell



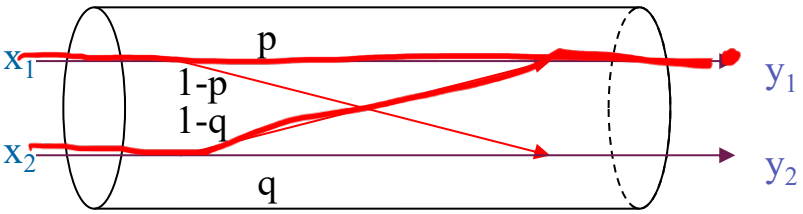
Kanalmatrix

$$H(X) = \sum p(x_i) \cdot \log\left(\frac{1}{p(x_i)}\right) = 0.5 \cdot \log(2) + 0.5 \cdot \log(2) = 1$$

Eingangssymbol

$p(x_1) = 0.5$

$p(x_2) = 0.5$



Ausgangssymbol

$$p(y_1) = p(x_1) \cdot p + p(x_2) \cdot (1 - q)$$
$$p(y_2) = p(x_1) \cdot (1 - p) + p(x_2) \cdot (q)$$

Abbildung oder Transformation

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

Kanalmatrix

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

OST

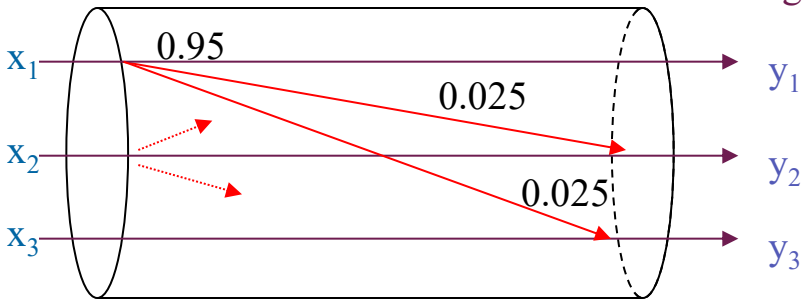
Beispiel: Kanalmatrix des symmetrischen Kanals

Eingangssymbol

$p(x_1)=0.5$

$p(x_2)=0.25$

$p(x_3)=0.25$

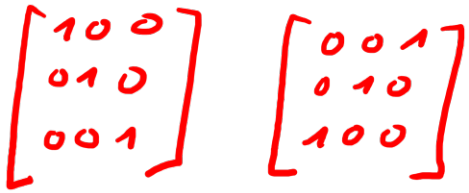


Ausgangssymbol

$$\begin{pmatrix} p(y_1) \\ p(y_2) \\ p(y_3) \end{pmatrix} = \begin{bmatrix} p(x_1)\cdot p(y_1|x_1) + p(x_2)\cdot p(y_1|x_2) + p(x_3)\cdot p(y_1|x_3) \\ p(x_1)\cdot p(y_2|x_1) + p(x_2)\cdot p(y_2|x_2) + p(x_3)\cdot p(y_2|x_3) \\ p(x_1)\cdot p(y_3|x_1) + p(x_2)\cdot p(y_3|x_2) + p(x_3)\cdot p(y_3|x_3) \end{bmatrix}$$

$$\begin{bmatrix} 0.4875 \\ 0.25625 \\ 0.25625 \end{bmatrix} = \begin{bmatrix} 0.5\cdot 0.95 + 0.25\cdot 0.025 + 0.25\cdot 0.025 \\ 0.5\cdot 0.025 + 0.25\cdot 0.95 + 0.25\cdot 0.025 \\ 0.5\cdot 0.025 + 0.25\cdot 0.025 + 0.25\cdot 0.95 \end{bmatrix}$$

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

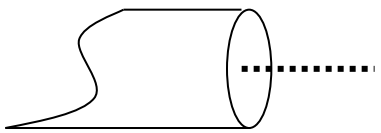


Fehlerwahrscheinlichkeit des Kanals.
Diese ist unabhängig von der Auftritts-wahrscheinlichkeit der Zeichen der Quelle.

Kanal symmetrisch

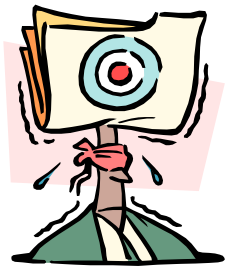
OST

Maximum-Likelihood-Verfahren,



Kanal

y_j



Entscheider

x_i

Kanaleigenschaften

$$p(Y|X) = \begin{bmatrix} 0.6 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.6 \\ 0.3 & 0.2 & 0.5 \end{bmatrix}$$

y_1 y_2 y_3

liefert Input

$y_1 \rightarrow x_1$

$y_2 \rightarrow x_3$

$y_3 \rightarrow x_2$

Wie gross ist die Restfehlerwahrscheinlichkeit und wovon hängt sie ab?

$r_1 = 0.30$

$r_2 = 0.09$

$r_3 = 0.01$

$y_1 \rightarrow x_1$

$y_2 \rightarrow x_3$

$y_3 \rightarrow x_2$

$$P_{korrekt} + P_{Fehler} = 1$$

$$P_{Fehler} = 1 - P_{korrekt}$$

$$P_K = \underbrace{p(x_1) \cdot p(y_1|x_1)} + \underbrace{p(x_2) \cdot p(y_3|x_2)} + \underbrace{p(x_3) \cdot p(y_2|x_3)}$$

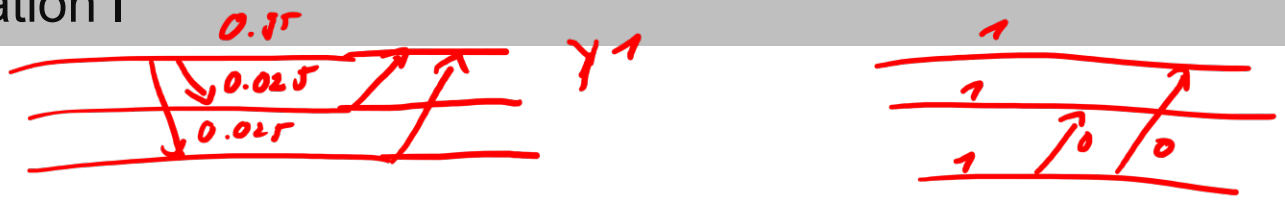
$$= p(x_1) \cdot 0.6 + p(x_2) \cdot 0.6 + p(x_3) \cdot 0.2$$

$$= \frac{1}{3} \cdot 0.6 + \frac{1}{3} \cdot 0.6 + \frac{1}{3} \cdot 0.2 = 0.466$$

$$P_F = 1 - 0.466 = 0.534$$

OST

Transinformation I



Eingangssymbol

$p(x_1)=0.5$

$p(x_2)=0.25$

$p(x_3)=0.25$

x_1 x_2 x_3

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

Ausgangssymbol

y_1 y_2 y_3

$$\begin{bmatrix} 0.4875 \\ 0.25625 \\ 0.25625 \end{bmatrix} = \begin{bmatrix} 0.5\cdot 0.95+0.25\cdot 0.025+0.25\cdot 0.025 \\ 0.5\cdot 0.025+0.25\cdot 0.95+0.25\cdot 0.025 \\ 0.5\cdot 0.025+0.25\cdot 0.025+0.25\cdot 0.95 \end{bmatrix}$$

$$H(X) = -\sum_{i=1}^{3} p(x_i)\cdot ld(p(x_i))$$
$$= -(0.5\cdot ld(0.5) + 2$$
$$= 0.5 + 1 = 1.5$$

$$H(Y) = -\sum_{i=1}^{3} p(y_i)\cdot ld(p(y_i))$$
$$= -(0.4875ld(0.4875) + 2\cdot 0.25625ld(0.25625))$$
$$= 0.5053 + 1.0067 = 1.512$$

$$H(X) \neq H(Y)$$

?

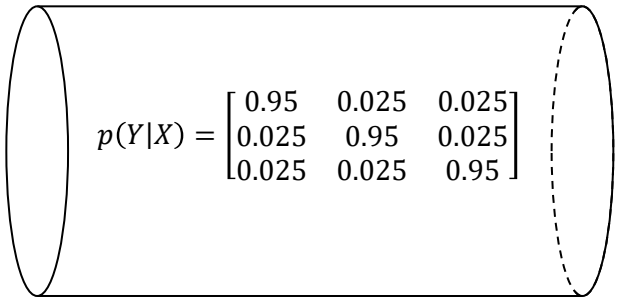
Transinformation I

Eingangssymbol

$p(x_1)=0.5$

$p(x_2)=0.25$

$p(x_3)=0.25$



Ausgangssymbol

$$\begin{bmatrix} 0.4875 \\ 0.25625 \\ 0.25625 \end{bmatrix} = \begin{bmatrix} 0.5 \cdot 0.95 + 0.25 \cdot 0.025 + 0.25 \cdot 0.025 \\ 0.5 \cdot 0.025 + 0.25 \cdot 0.95 + 0.25 \cdot 0.025 \\ 0.5 \cdot 0.025 + 0.25 \cdot 0.025 + 0.25 \cdot 0.95 \end{bmatrix}$$

Verbundentropie:

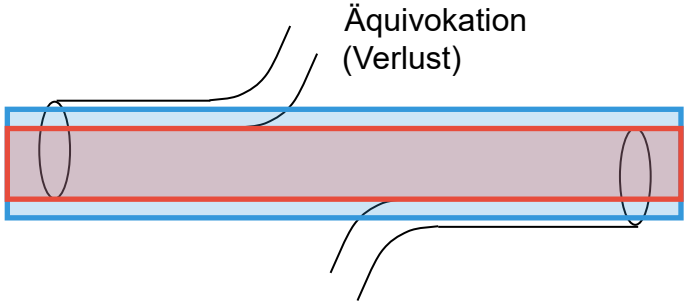
Das paarweise Auftreten aller möglichen Kombinationen am Kanaleingang und -ausgang

$$H(X,Y) = -\sum_{i}^{n}\sum_{j}^{n} p(x_i,y_j) * log_2(p(x_i,y_j))$$

OST

Äquivokation (Verlust)

$$H(X|Y) = -\sum_{i}^{n}\sum_{j}^{n} p(x_i, y_j) * log_2(p(x_i|y_j))$$



Vergleiche mit der Verbundentropie:

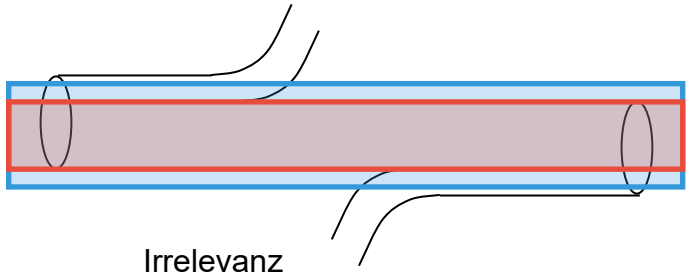
$$H(X,Y) = -\sum_{i}^{n}\sum_{j}^{n} p(x_i, y_j) * log_2(p(x_i, y_j))$$

- **Auch Rückschlussentropie genannt**
- **Ungewissheit über das gesendete Zeichen bei bekanntem Empfangszeichen**
- ***Merke*: Ist der Kanal fehlerfrei, so ist die Äquivokation gleich 0.**

OST

Irrelevanz (Rauschen)

$$H(Y|X) = -\sum_{i}^{n}\sum_{j}^{n} p(x_i, y_j) * log_2(p(y_j|x_i))$$



Irrelevanz
(Rauschen)

Vergleiche mit der Verbundentropie:

$$H(X,Y) = -\sum_{i}^{n}\sum_{j}^{n} p(x_i, y_j) * log_2(p(x_i, y_j))$$

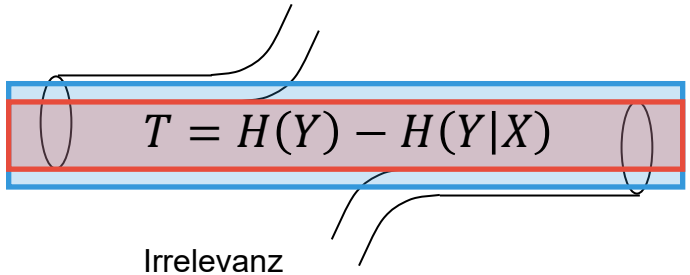
- **Auch Streuentropie genannt**
- **Ungewissheit der empfangenen Zeichen bei vorgegebenen Sendezeichen**

OST

Irrelevanz (Rauschen)

$$H(Y|X) = -\sum_{i}^{n}\sum_{j}^{n} p(x_i, y_j) * log_2(p(y_j|x_i))$$

$$H(Y|X) = -\sum_{i}^{n}\sum_{j}^{n} p(x_i)p(y_j|x_i) * log_2 p(y_j|x_i)$$

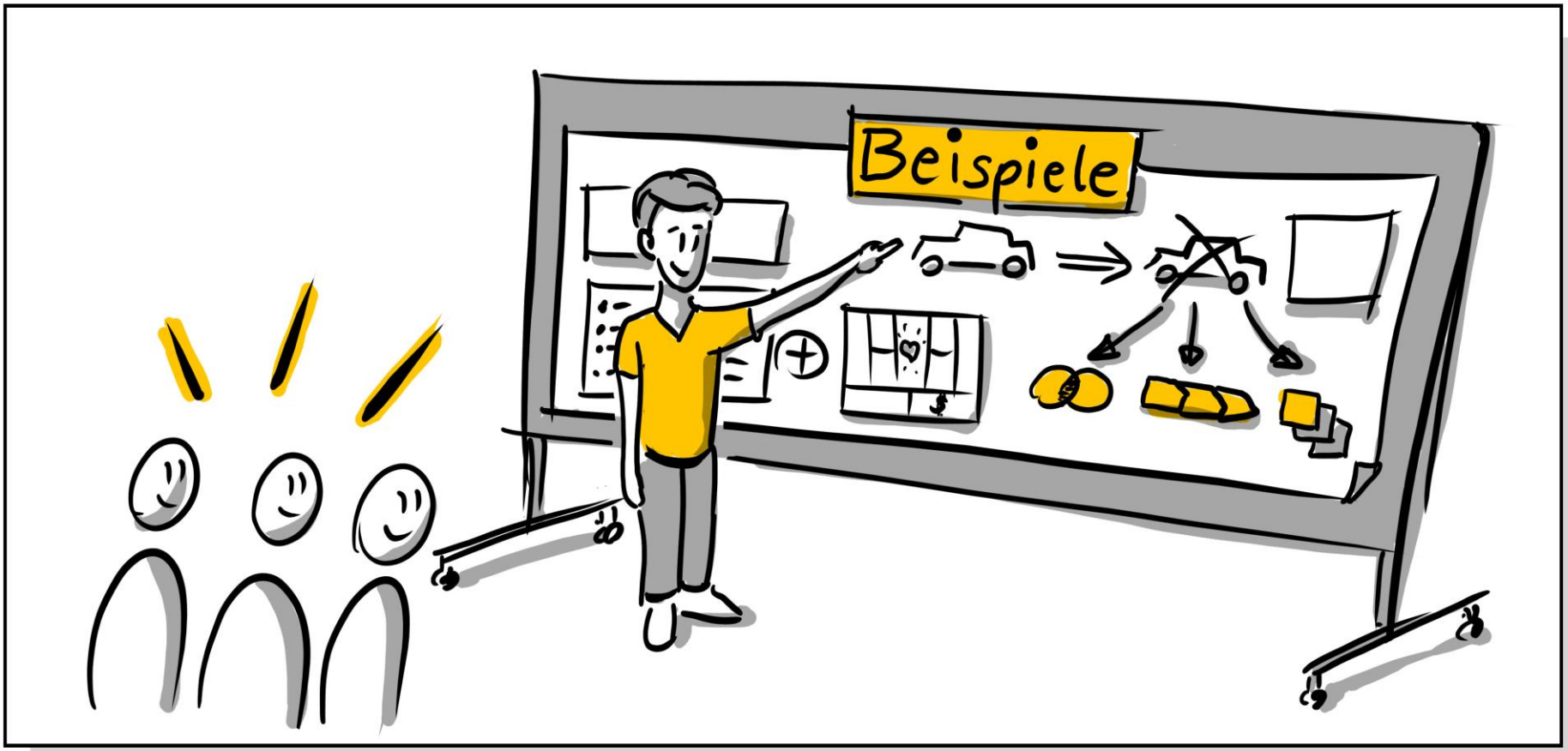


Zusammenfassung Begriffe

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

OST

Beispiele

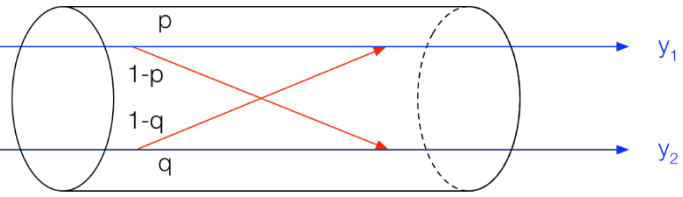


Transinformation Beispiel 1 (4)

Eingangssymbol

$p(x_1) = 0.5$ x_1

$p(x_2) = 0.5$ x_2



Ausgangssymbol

$$\begin{bmatrix} p(y_1) \\ p(y_2) \end{bmatrix} = \begin{bmatrix} p(x_1) * p(y_1|x_1) + p(x_2) * p(y_1|x_2) \\ p(x_1) * p(y_2|x_1) + p(x_2) * p(y_2|x_2) \end{bmatrix}$$

Transinformation?

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

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

Sei:

$p = q = 1$

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

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

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

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

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

$$= 0$$

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

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

$T = H(Y) = 1$

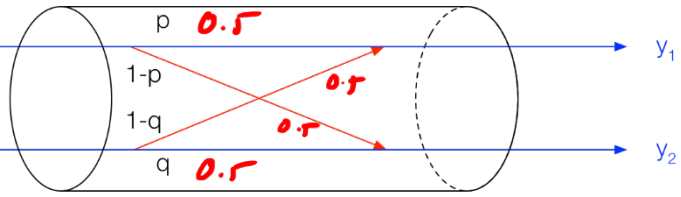
Transinformation Beispiel 2 (4)

$$\begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}$$

Eingangssymbol

$p(x_1) = 0.5$ x_1

$p(x_2) = 0.5$ x_2



Ausgangssymbol

$$\begin{bmatrix} p(y_1) \\ p(y_2) \end{bmatrix} = \begin{bmatrix} p(x_1) * p(y_1|x_1) + p(x_2) * p(y_1|x_2) \\ p(x_1) * p(y_2|x_1) + p(x_2) * p(y_2|x_2) \end{bmatrix}$$

Transinformation?

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

Sei:

$p = q = 0.5$

$\Rightarrow p(y_1) = 2 * 0.5^2 = 0.5$

$p(y_2) = 2 * 0.5^2 = 0.5$

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

$$H(Y|X) = -\sum_i^n \sum_j^n p(x_i, y_j) * log_2(p(y_j|x_i))$$
$$= -(4 * (0.25 * -(1)))$$
$$= 1$$

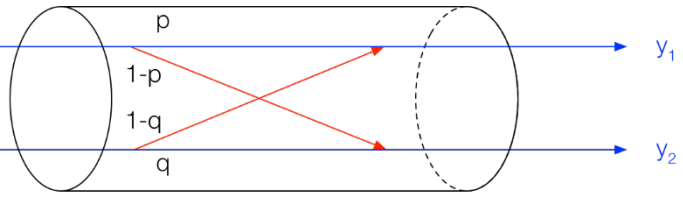
$$T = H(Y) - H(Y|X)$$
$$T = 0$$

Transinformation Beispiel 3 (4)

Eingangssymbol

$p(x_1) = 0.75$ x_1

$p(x_2) = 0.25$ x_2



Ausgangssymbol

$$\begin{bmatrix} p(y_1) \\ p(y_2) \end{bmatrix} = \begin{bmatrix} p(x_1) * p(y_1|x_1) + p(x_2) * p(y_1|x_2) \\ p(x_1) * p(y_2|x_1) + p(x_2) * p(y_2|x_2) \end{bmatrix}$$

Transinformation?

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

Sei:

$p = q = 1$

$\Rightarrow p(y_1) = p(x_1) = 0.75$

$p(y_2) = p(x_2) = 0.25$

$\Rightarrow H(Y) = H(X) \approx 0.811$

$$H(Y|X) = -\sum_i^n \sum_j^n p(x_i, y_j) * log_2(p(y_j|x_i))$$
$$= -\big(p(x_1) * p(y_1|x_1) * log_2(1) + p(x_2) * p(y_2|x_2) * log_2(1)\big)$$
$$= 0$$

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

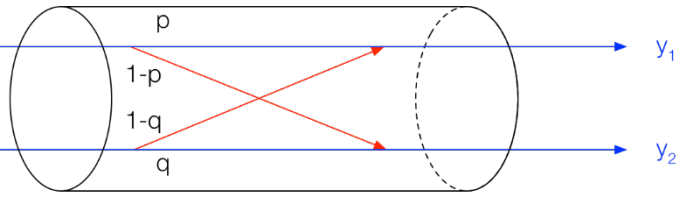
OST

Transinformation Beispiel 4 (4)

Eingangssymbol

$p(x_1) = 0.75$ x_1

$p(x_2) = 0.25$ x_2



Ausgangssymbol

$$\begin{bmatrix} p(y_1) \\ p(y_2) \end{bmatrix} = \begin{bmatrix} p(x_1) * p(y_1|x_1) + p(x_2) * p(y_1|x_2) \\ p(x_1) * p(y_2|x_1) + p(x_2) * p(y_2|x_2) \end{bmatrix}$$

Transinformation?

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

Sei:

$p = q = 0.5$

$\Rightarrow p(y_1) = 0.5 * (0.75 + 0.25) = 0.5$

$p(y_2) = 0.5 * (0.25 + 0.75) = 0.5$

$\Rightarrow H(Y) = 1;\ H(X) \approx 0.811$

$$H(Y|X) = -\big(2 * \big(0.75 * 0.5 * (-1)\big) + 2 * (0.25 * 0.5 * (-1))\big) = 1$$

$$T = H(Y) - H(Y|X)$$
$$T = 0$$

Fazit

- **Ein nicht gestörter Kanal (Einheitsmatrix) überträgt den mittleren Informationsfluss ohne weiteren Verlust, d.h. die Transinformation wird nur durch die Quelle bestimmt.**
- **Verändert sich die Entropie der Quelle, so verändert sich auch die Transinformation.**
- **Nimmt die Fehlerwahrscheinlichkeit zu, so verringert sich die Transinformation.** wenn über 50% Wahrscheinlichkeit kann Kehrwert genommen werden und T wird grösser...
- **Sind alle Positionen der Kanalmatrix gleich besetzt, so wird die Transinformation $T = 0$, d.h. $H(Y) = H(Y|X) = 1$ und zwar unabhängig von der Entropie am Kanaleingang.**
- **Die Transinformation gibt den maximalen und somit fehlerfreien Informationsfluss über einen gestörten Kanal an.**
  Leider haben wir noch keine Aussage, wie dies zu erreichen ist!
- **Idee ?**

OST

Teil 1: Mathematische Grundlagen zu Zahlen und Algebra

1. **Mathematische Grundlagen: Das Stellenwertsystem**
2. **Binärzahlen: Praktisch angeschaut**
3. **Eine (nicht) mathematische Einführung in die Gruppentheorie**
   - Was ist Gruppe, Ring, Körper?
   - Modulo Rechnung
   - Interpretation eines Datenworts als
     - Tupel, Zahl, Vektor, Polynom
   - Was ist eine zyklische Gruppe
   - Bool`sche Alegra
4. **Erste Schritte in den Wahrscheinlichkeitsbegriff und die Kombinatorik**

Teil 2: Codierungs- und Informationstheorie Grundlagen

1. **Einführung in die Informationstheorie**
2. **Quellencodierung**
   - Komprimierung
   - Verschlüsselungsverfahren
3. **Kanalmodell**
4. **Kanalcodierung**
   - Blockcodes
   - Faltungscodes

OST

Repetition

- **Was ist Entropie? Was hat das mit dem Informationsgehalt zu tun?**
- **Was beschreibt die Kanalmatrix?**

- **Erklären Sie in eigenen Worten den Begriff «Transinformation» (ohne Formeln)**
- **Was verstehen wir unter Äquivokation bzw. Irrelevanz?**

Struktur

Hamming und Abramson sind konkrete Codefamilien,
CRC ist ein allgemeines Prüfverfahren auf Basis von Polynomen.

Blockcodes

- **Coderaum**
- **Hamming Blockcode**
- **zyklischer Hammingcode**
- **Abramson Code**

Hammingcode
- Typ: Linearer Blockcode (Fehlererkennung und -korrektur)
- Verwendung: Korrigiert 1-Bit-Fehler, erkennt 2-Bit-Fehler
- Funktionsweise: Fügt Paritätsbits an die Daten an,

sodass mit einer einfachen Logik Fehlerpositionen bestimmt werden können.
- Generatorpolynom: Wird bei zyklischen Hammingcodes verwendet,

z.B. (x^3 + x^2 + 1) für (7,4)-Code.

CRC-Code (Cyclic Redundancy Check)

Typ: Prüfsummenverfahren (Fehlererkennung, keine Korrektur)
Verwendung: Häufig bei Datenübertragung (Netzwerk, Speicher, USB)
Funktionsweise: Die Daten werden als Polynom interpretiert und durch ein Generatorpolynom dividiert (Division modulo 2). Der Rest ist der CRC-Code.
Generatorpolynom: Das zentrale Element beim Berechnen und Prüfen des CRC-Codes.

Abramson code
Typ: Zyklischer Code, Spezialfall des CRC
Verwendung: Fehlererkennung (ähnlich wie CRC, oft als "Ein-Bit-Fehler-erkennender Code" bezeichnet)
Funktionsweise: Nutzt ein Generatorpolynom mit mindestens drei Termen (z.B. (x^r + x + 1)), um insbesondere Einzelbitfehler zu erkennen.
Generatorpolynom: Bestimmt die Fehlererkennungseigenschaften.

OST

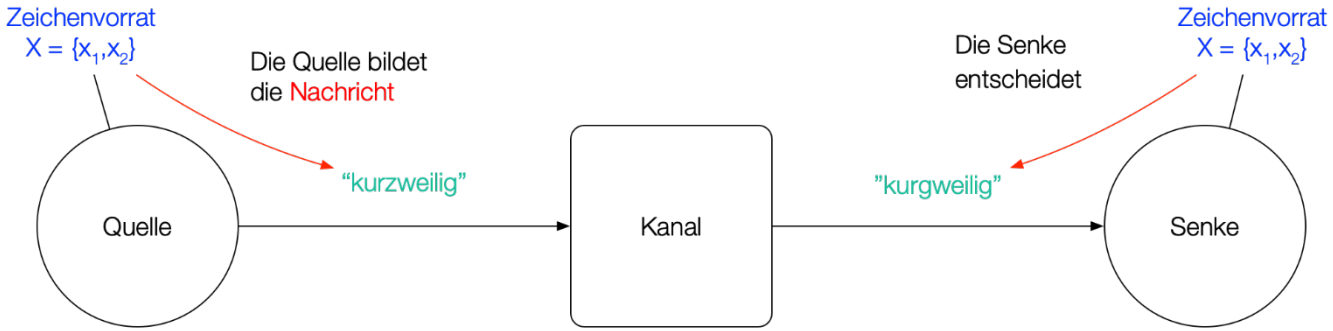
“Die Modulo-Funktion bei Polynomen” aus dem Skript:

Mathematische Hilfsmittel zur Codierungstheorie – ohne Ballast

Kanalcodierung

Wir wissen ja nur, das was falsch ist – aber wieviele Bits wurden nun gestört?

Im 3-d-Coderaum: die grau hinterlegten CW wären fehlerhaften CW. D.h. wenn 1-Bit-Fehler,
dann erkennen wir den Fehler, wenn wir 2-bit-drehen, können wir den Fehler nicht mehr erkennen.
Wir müssen uns nun überlegen, wie wir das erweitern können (d.h. die einzelnen CW möglichst weit auseinanderziehen



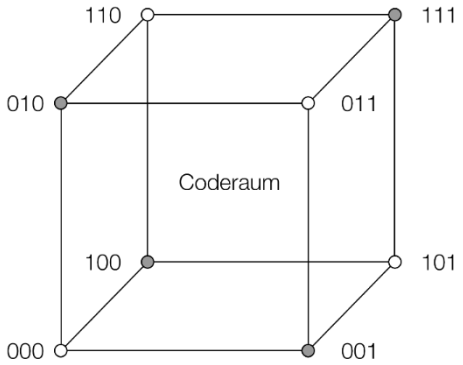
Kurgweilig
Wie korrigieren??
kurzweilig
oder
langweilig
?

Idee

Hinzufügen von Redundanz,
so dass sich der zur Verfügung
stehende Coderaum in gültige
und ungültige Codeworte aufteilt.

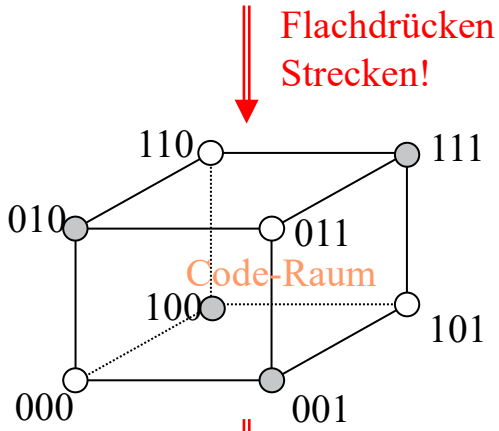
(**CW: Codewort**)

technisches
Beispiel

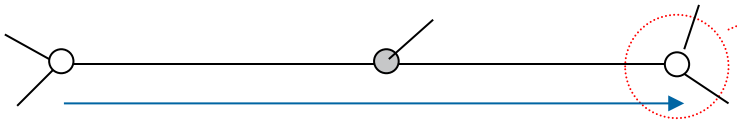


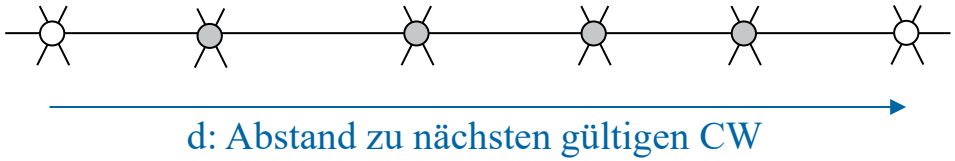
Grundidee: wir fügen nun so Redundanz hinzu, damit der Abstand zwischen allen gültigen CW maximal bzw. möglichst gross wird. Denn grosser Abstand = viele Fehler erkennen und ggf. korrigieren.

Der n-dimensionale Coderaum



den drei-dimensionalen, flachgedrückten Würfel



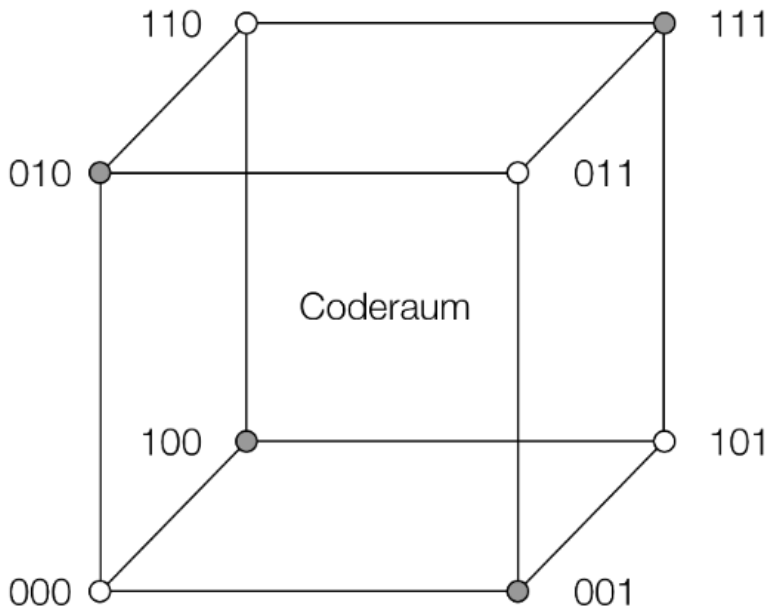


der sechs-dimensionalen, flachgedrückten Würfel

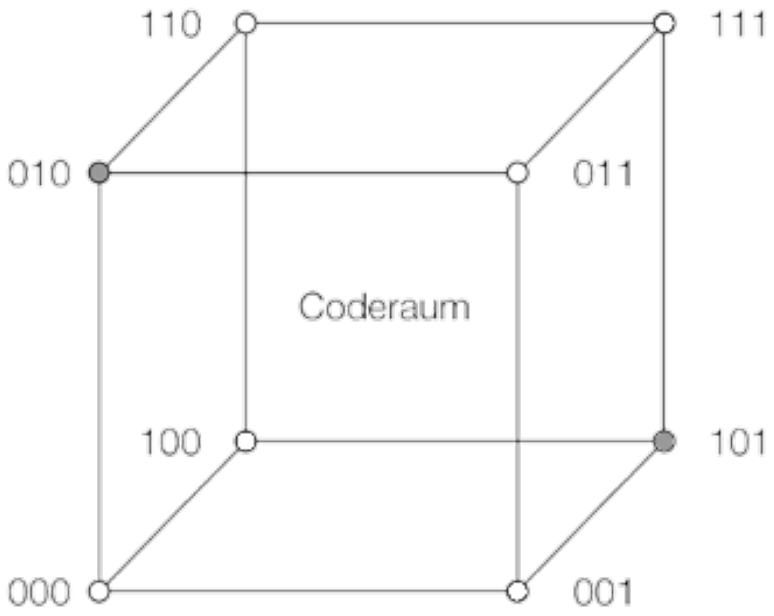
Rest (n) folgt durch Induktion ;)

Wieviele Fehler erkennbar und korrigierbar?

die grau hinterlegten CW wären fehlerhaften CW



1 erkennbar, 0 korrigierbar -> Hammingabstand von 2



2 erkennbar, 1 korrigierbar -> Hammingabstand von 3

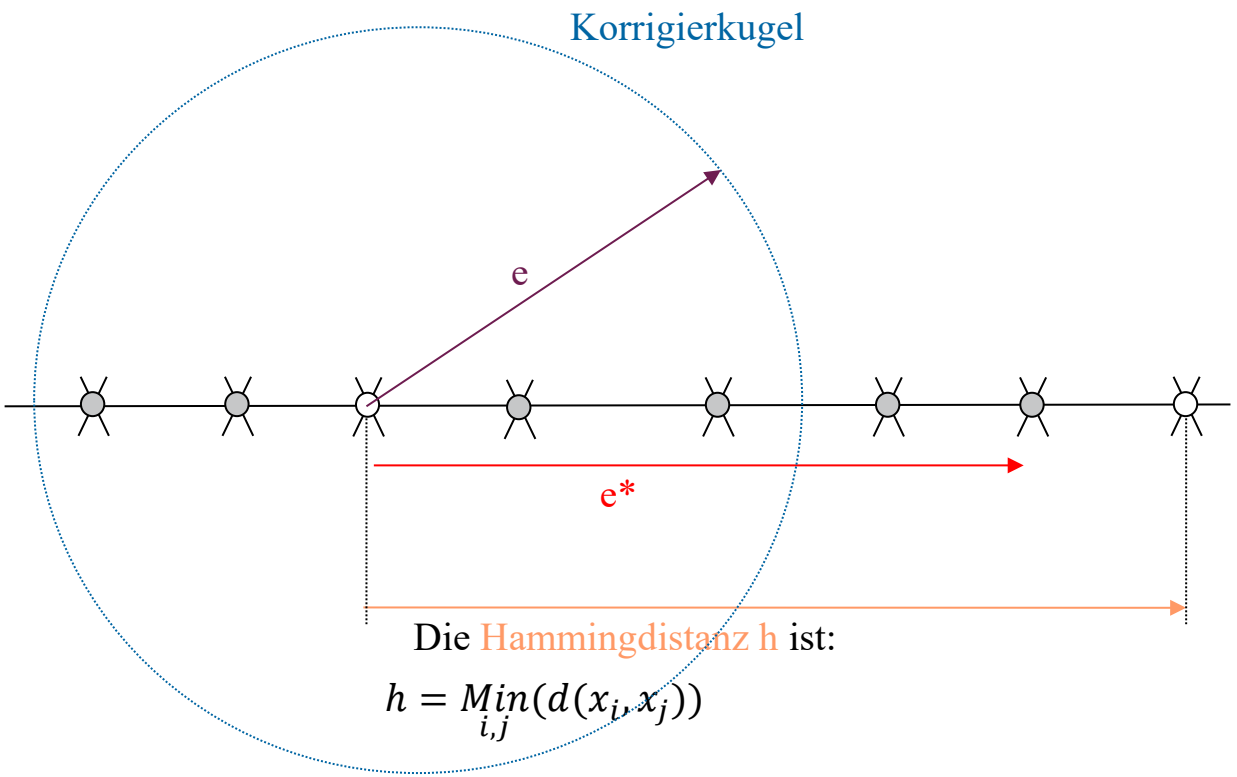
OST

Coderaum: Definitionen

Hammingdistanz = Mindestdistanz zwischen allen gültigen Codeworten

Korrigierkugel: die Korrigierkugeln der einzelnen korrekten CW dürfen sich nicht überschneiden – sonst würden wir falsch korrigieren.
In einer Korrigierkugel gibt es immer nur 1 gültiges Codewort

Wenn alle CW innerhalb einer Korrigierkugel = Code ist dichtgepackt



Hier ist h = 5 (=ungerade)
e* = 4
e = abgerundet((h-1)/2) = 2

Frage: bei einer Hammingdistanz von 3, können dann 4 Fehler noch erkannt werden?
Antwort: vielleicht aber nicht mehr sicher.

Anzahl der sicher erkennbaren Fehler

$$e^* = h - 1$$

Anzahl der sicher korrigierbaren Fehler

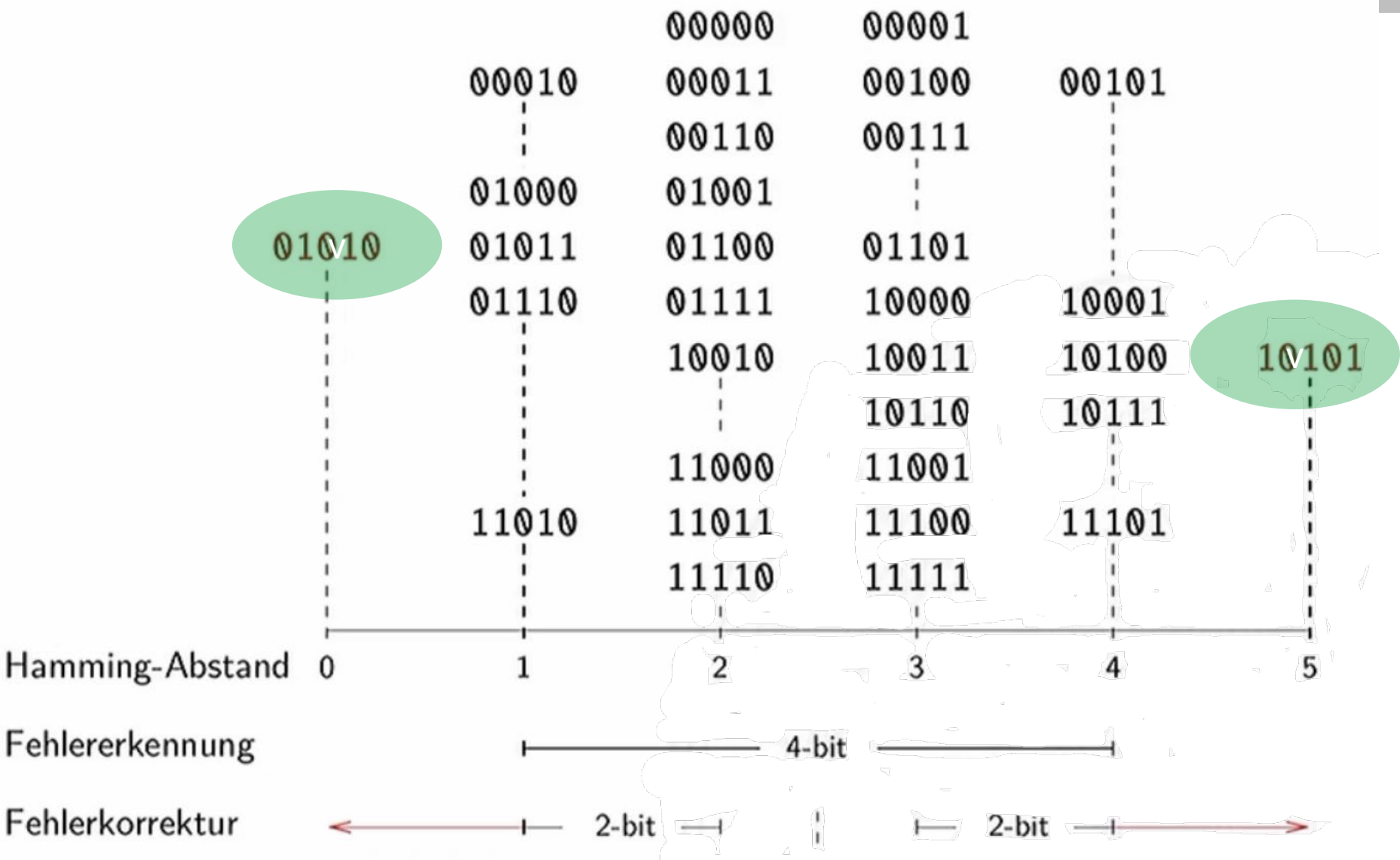
- **h gerade:**

$$h = 2e + 2 \Rightarrow$$
$$e = \frac{h-2}{2}$$

- **h ungerade:**

$$h = 2e + 1 \Rightarrow$$
$$e = \frac{h-1}{2}$$

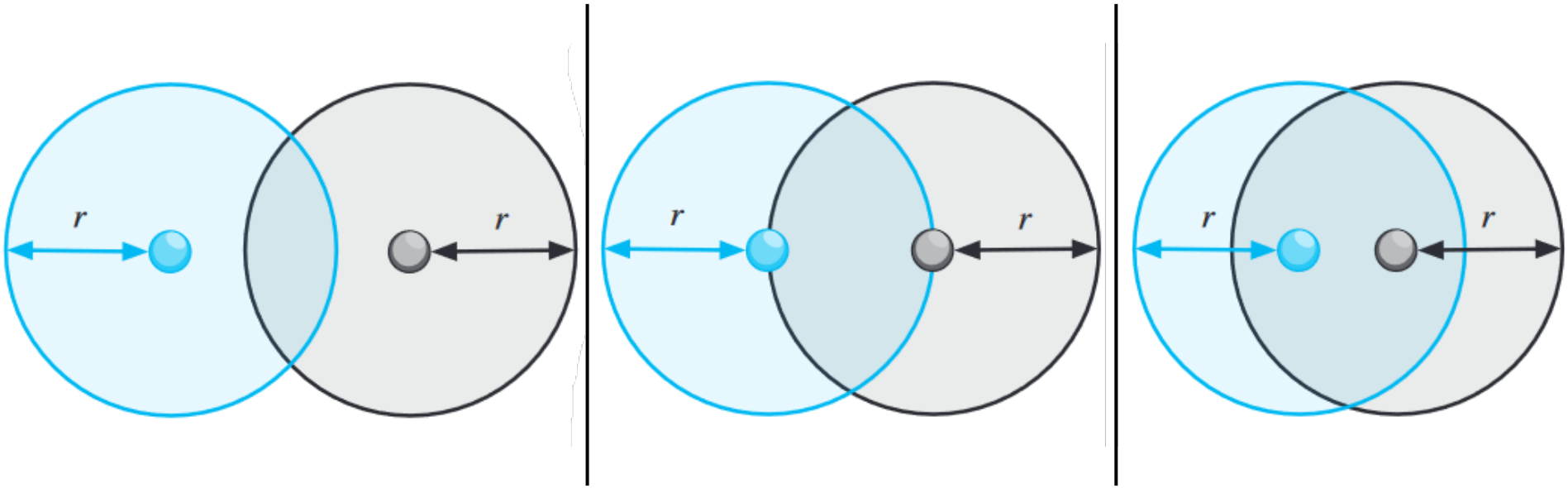
Fehlererkennung und Korrektur



Coderaum: Fehlererkennung

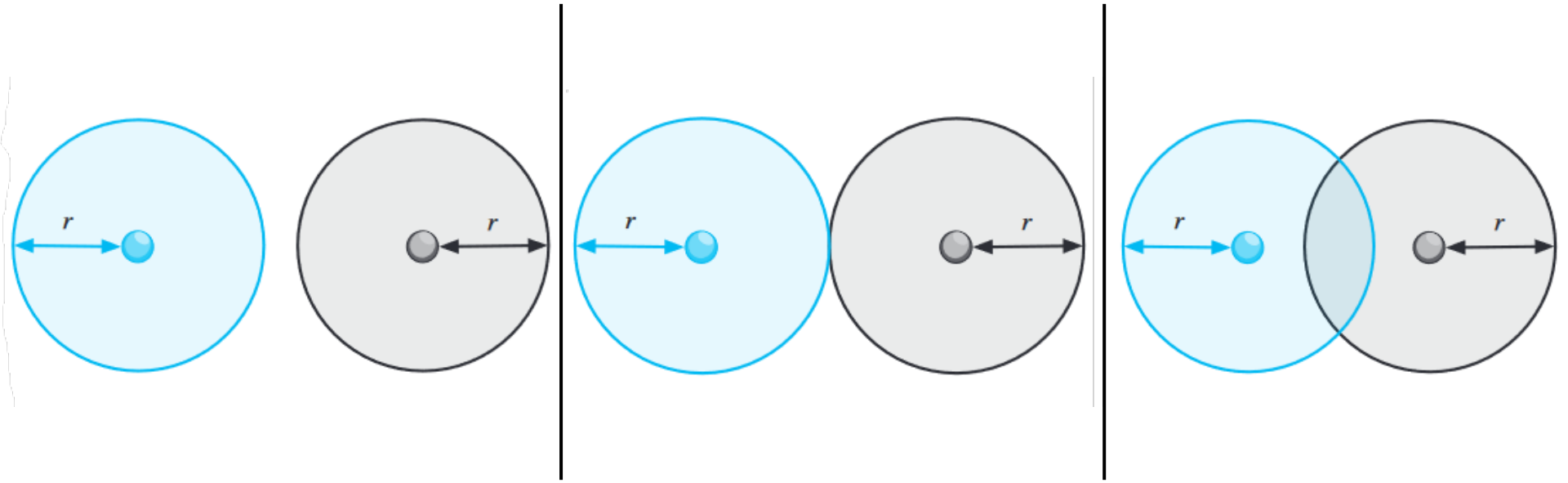
Erst wenn die soweit auseinander liegen, dass sie eindeutig innerhalb einer Kugel befinden, können sie e* (=r) Fehler erkennen.

Sind zwei CW so nahe beisammen, dass sie auf dem Rand oder innerhalb einer anderen Kugel liegen,
kann es passieren, dass CW durch die Veränderung von r oder weniger Bits zu einem anderen CW wird.
Somit können nicht mehr e* (=r) Fehler erkannt werden.



Quelle: Dirk W. Hoffmann, Einführung in die Informations- und Codierungstheorie, Vieweg 2023

Coderaum: Fehlerkorrektur



Erst wenn die Distanz zu einem anderen Codewort den Wert 2r (=e) übersteigt ist eine Korrektur von e Fehler möglich

Sind die CW so nahe beisammen, dass sich die Kugeln berühren oder schneiden,
kann ein CW durch die Veränderung von r oder weniger Bits nicht mehr eindeutig zugeordnet werden.

Quelle: Dirk W. Hoffmann, Einführung in die Informations- und Codierungstheorie, Vieweg 2023

Coderaum: *Dichtgepackt* oder nicht, das ist hier die Frage.

Wenn <= bedeutet dies, es sind nicht alle CW in einer Korrigierkugel,
wenn = bedeutet dies, alle CW sind in einer Korrigierkugel.

Ist ein Code nicht dichtgepackt, gibt es CW die sich nicht in einer Korrigierkugel befinden.
Dann können diese nicht erkennt und korrigiert werden.

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

- n die Dimension des Code (Anzahl aller CW = 2^n),
- m die Dimension der Nachrichten (Anzahl aller gültigen CW = 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
Für jedes gültige CW eine Korrigierkugel

Anzahl der CW pro Korrigierkugel
einschliesslich das gültige CW

Anzahl aller CW

Gilt:

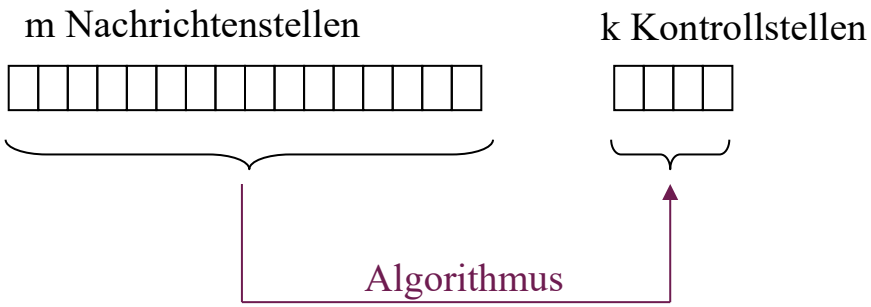
$$2^m \cdot \sum_{w=0}^{e} \binom{n}{w} = 2^n$$

So ist der Code dichtgepackt!

=wieviele Möglichkeiten gibt es, w Fehler in einem Codewort mit n Bit zu haben. (Anzahl der Möglichkeiten, w Eleme

OST

Blockcodes: Einführung



Beispiel: Quersummencode

m=2 k=1

x_1	x_2	x_3
0	0	0
0	1	1
1	0	1
1	1	0
0	0	1
0	1	0
1	0	0
1	1	1

Gültige Codeworte,
sie erfüllen den Algorithmus

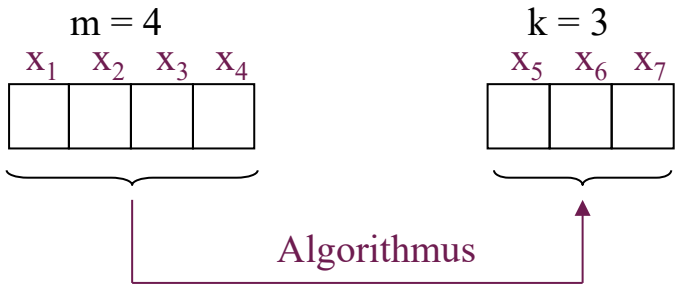
Ungültige Codeworte,
sie erfüllen den Algorithmus nicht,
d.h. sie liefern ein *Fehlermuster*

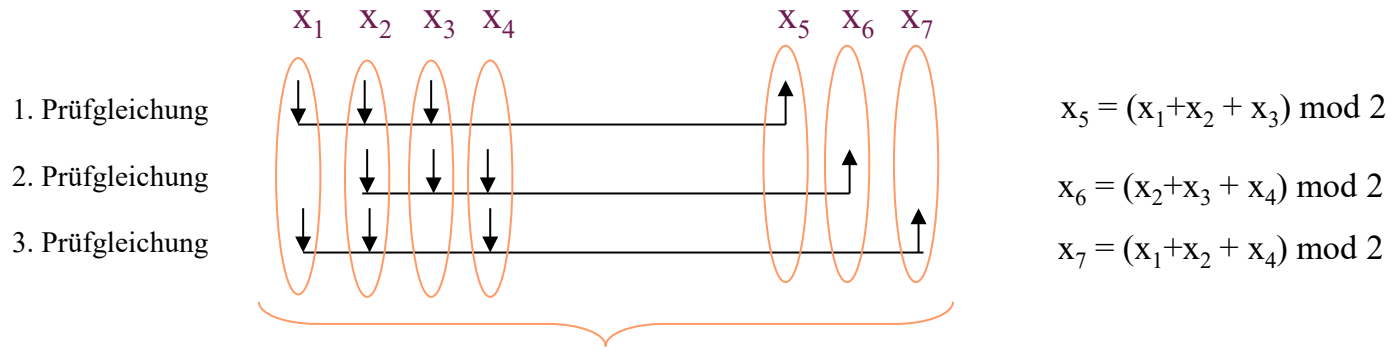
Algorithmus zur Berechnung der Kontrollstellen

$$x_3 = (x_1 + x_2) \bmod 2$$

Blockcodes: Hamming-Code I

Hammingcode hat die Standard-Hammingdistanz von 3





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

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

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

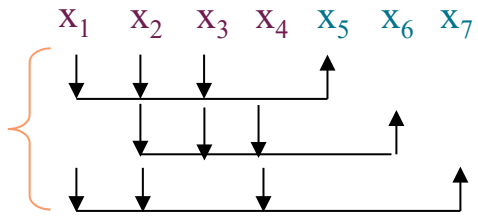
Alle Vektoren paarweise verschieden und ungleich 0 !

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.

Frage: wie viele Fehler können nicht mehr erkannt werden? 3

Blockcodes: Hamming-Code II

Wenn wir die Einheitsmatrix für die Kontrollstellen haben, haben wir einen systematischen Code



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

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

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

Generatormatrix

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

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

Einheitsmatrix markiert die Anzahl der Kontrollstellen k

Hieraus folgt die Codebedingung: $\sum_i x_i \cdot \vec{P}_i \equiv \vec{0} \bmod 2$

OST

Blockcodes: Hamming-Code III

x_1	x_2	x_3	x_4	x_5	x_6	x_7
0	0	0	0			
0	0	0	1			
0	0	1	0			
0	0	1	1			
0	1	0	0			
0	1	0	1			
0	1	1	0			
0	1	1	1			
1	0	0	0			
1	0	0	1			
1	0	1	0			
1	0	1	1			
1	1	0	0			
1	1	0	1			
1	1	1	0			
1	1	1	1			

Nachrichtenstellen Kontrollstellen

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

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

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

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

Blockcodes: Hamming-Code III

Formel: an jeder stelle xi multiplizieren wir den Vektor aus der Generatormatrix und erhalten in der Summe den Null-Vektor.

x_1	x_2	x_3	x_4	x_5	x_6	x_7
0	0	0	0	0	0	0
0	0	0	1	0	1	1
0	0	1	0	1	1	0
0	0	1	1	1	0	1
0	1	0	0	1	1	1
0	1	0	1	1	0	0
0	1	1	0	0	0	1
0	1	1	1	0	1	0
1	0	0	0	1	0	1
1	0	0	1	1	1	0
1	0	1	0	0	1	1
1	0	1	1	0	0	0
1	1	0	0	0	1	0
1	1	0	1	0	0	1
1	1	1	0	1	0	0
1	1	1	1	1	1	1

Nachrichtenstellen | Kontrollstellen

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

P = Prüfvektor

Die Codebedingung:

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

Wird für alle gültigen Codeworte (Tabelle) erfüllt.

Was ergibt die Berechnung der Codebedingung bei einem Bitfehler?

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

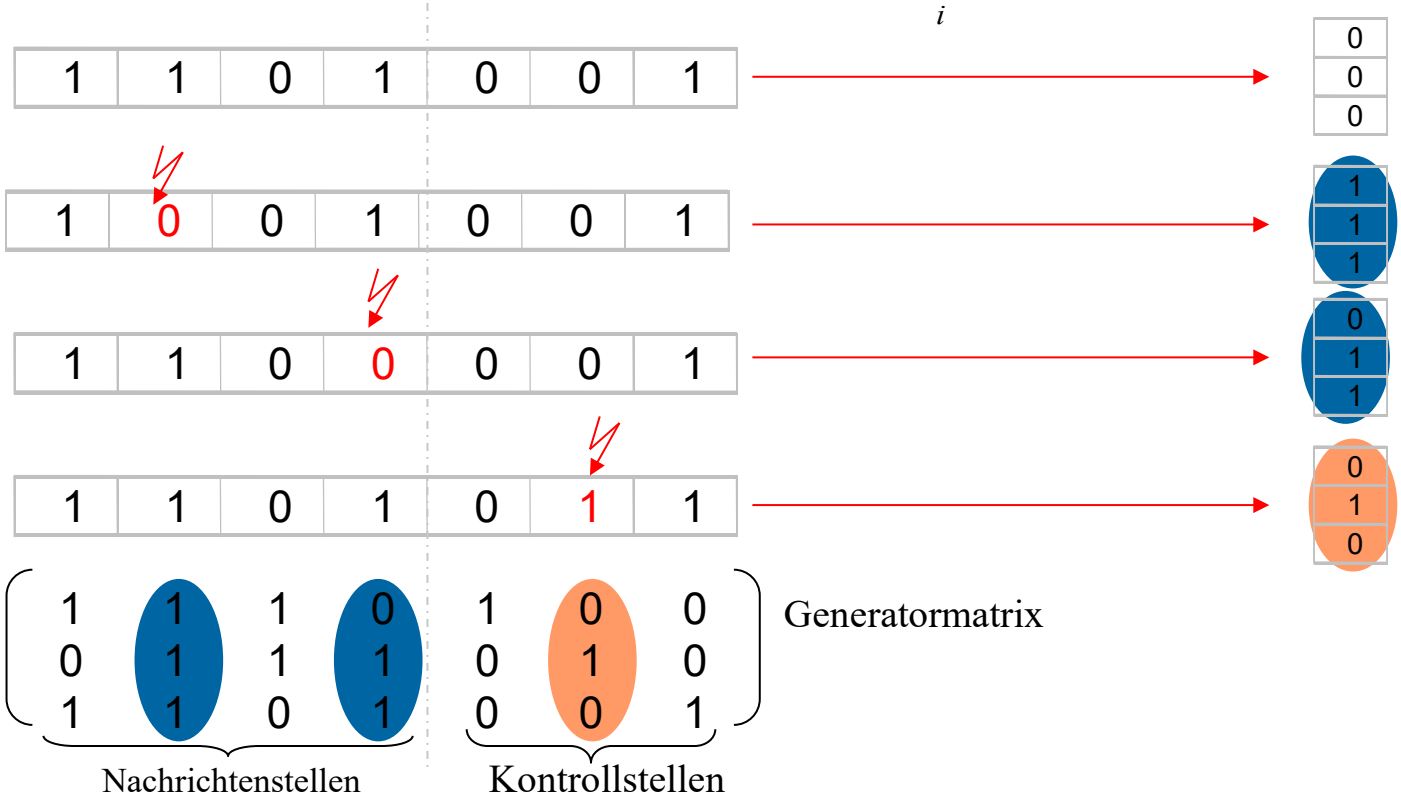
$x_6 = (x_2 + x_3 + x_4) \bmod 2$

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

OST

Das Syndrom Z:

$$\vec{Z} = \sum_i x_i \cdot \vec{P}_i \bmod 2$$



Hamming-Code: Das Fehlersyndrom

+ ist XOR weil es in mod 2 ist.

Bewertung des linearen Blockkodes
Die Codierung
+
ist gut verständlich
die Konstruktion der Geratormatrix ist einfach
-
die Implementierung einer schnellen Auswertung des Syndroms jedoch schwierig bzw. zeitaufwendig
Begründung:
Das ganze CW muss erst eingelesen werden, bevor die Auswertung beginnen kann.
Die CW sind in der Praxis sehr lang, d.h. auch die Prüfmatrix wäre bei einem 16 Bit- Prüfwort sehr gross

Gesendetes Codewort $X = [x_1, x_2, x_3, .., x_n]$

Überlagert durch das Fehlermuster $F = [f_1, f_2, f_3, .., f_n]$

Empfangenes Wort

$$X' = X + F$$
$$= [x_1 + f_1, x_2 + f_2, x_3 + f_3, .., x_n + f_n] \bmod 2$$
$$= [x'_1, x'_2, x'_3, .., x'_n]$$

Wenn kein Fehler besteht F aus [0,0,0,0,…0], wenn wir einen Fehler haben ist eines dieser f = 1
Fehlermuster ist natürlich zufällig

Z = Fehlersyndrom

Aus der Codebedingung folgt das Syndrom

$$\vec{Z} = \sum_i x'_i \cdot \vec{P_i} = \sum_i (x_i + f_i) \cdot \vec{P_i}$$
$$= \sum_i x_i \cdot \vec{P_i} + \sum_i f_i \cdot \vec{P_i}$$

Codebedingung = 0

$$\Rightarrow \vec{Z} = \sum_i f_i \cdot \vec{P_i}$$

Das heisst, bei genau einem Fehler markiert die Prüfspalte den Fehlerort.

Generatorpolynom holt man aus Bücher: das muss prim/primitiv sein.
Anzahl Kontrollstellen = Grad des Generator-Polynoms

Idee: Generatormatrix kann durch Generatorpolynom beschrieben werden!

Ziel: Vereinfachte Berechung der Kontrollstellen durch rückgekoppelte Schieberegister.

Generatorpolynom G(u)

$$G(u) = \sum_{i=0}^{k} g_i \cdot u^i$$

Codewortpolynom X(u)

$$X(u) = \sum_{i=0}^{n} g_i \cdot u^i$$

Codebedingung →

Das Codewortpolynom ist ohne Rest durch das Generatorpolynom teilbar (in mod-2-Rechnung)

$$X(u) \div G(u) \equiv Q(u) \bmod 2$$
$$X(u) \equiv Q(u) \cdot G(u) \bmod 2$$

$g_i = \{0,1\}$ mit $g_0 = g_k = 1$

Grad **k** entspricht der Anzahl der Prüfstellen.
Grad **n** entspricht der Anzahl der Codewortstellen.
Die Zahl der Nachrichtenstellen ist **m**
⇒ **n = m +k**

OST

Zyklische Codes: Ermittlung der Kontrollstellen durch Polynomdivision

(7,4) Hammingcode. 4 bits werden mit + 3 kontrollstellen zu 7 bits codiert.
Hammingdistanz ist standardmässig 3 bei Hamming-Code

Sei: $m = 4, k = 3, n = 7$

Nachricht: $(x_1,x_2,x_3,x_4,) = (1\ 0\ 0\ 0)$

Generator: $G(u) = u^3 + u + 1 \Rightarrow (g_3\ g_2\ g_1\ g_0) = (1\ 0\ 1\ 1)$

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

1 0 0 0 1 0 1 : 1 0 1 1 ≡ 1 0 1 1 **mod 2**

```
1 0 1 1
% 0 1 1
  0 0 0 0
% 1 1 0
  1 0 1 1
% 1 1 1
  1 0 1 1
% 1 0 1
```

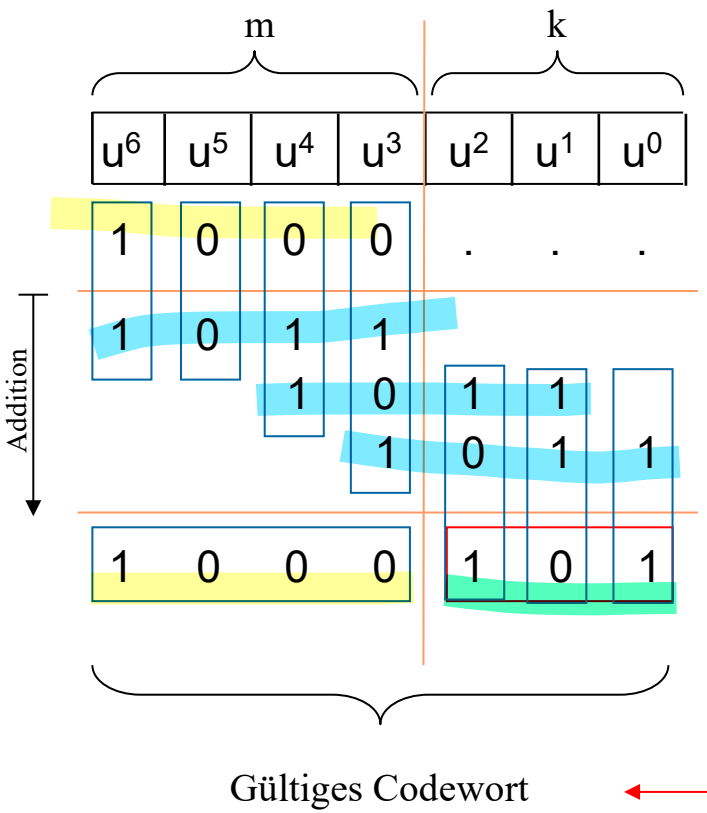
101 sind die gesuchten Kontrollstellen, die die Codebedingung erfüllen.

Zyklische Codes: Ermittlung der Kontrollstellen durch Mehrfachaddition

Polynomdivison durch Mehrfachaddition vereinfacht

Weil: 9:3 könnte man ersetzen mit: wieviele Male passt die 3 in die 9
oder 3+3+3=9

Nur Nachrichtenwort – nicht Codewort (s.h. ohne Kontrollstellen)
Wir addieren soviele Male, bis wir den Nachrichten-Teil des Codewortes erhalten: 1000



Sei: $m = 4, k = 3, n = 7$

Nachricht: $(x_1,x_2,x_3,x_4,) = (1\ 0\ 0\ 0)$

Generator: $G(u) = u^3 + u + 1 \Rightarrow (g_3\ g_2\ g_1\ g_0) = (1\ 0\ 1\ 1)$

Idee: X(u) ist durch G(u) mod2 teilbar, also muss X(u) durch Addition von G(u) mod2 erzeugbar sein!

Addition des ersten Terms G(u) erzeugt die Stellen u^6, u^5 von X(u)

Addition des zweiten Terms G(u) erzeugt die Stelle u^4 von X(u)

Addition des dritten Terms G(u) erzeugt die Stelle u^3 von X(u)

Der Rest muss nach Codebedingung die Kontrollstellen bilden!

Zyklische Codes: Prüfen der Codebedingung

Empfangenes Codewort: 1 0 0 0 1 0 1

Generator: 1 0 1 1

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

Code- Bedingung erfüllt!

Idee:
Durch die Codebedingung muss die fortgesetzte Addition (mod 2) des Generators zum empfangenen CW Das Nullwort ergeben.

$$X(u) \div G(u) \equiv Q(u) \bmod 2$$
$$X(u) \equiv Q(u) \cdot G(u) \bmod 2$$

Empfangenes Codewort: 1 0 0 1 1 0 1

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

Code- Bedingung nicht erfüllt!

Fehlersyndrom

OST

Zyklischer Hamming- Code und Generatormatrix ?

Das sind die Fehlersyndrome zu dem Codewort

Wir nehmen ein gültiges Codewort (das von vorherigem Slide) und fügen an jeder Stelle einen Fehler ein.
So erhalten wir für jeden Fehler das Fehlersyndrom.

Jedes Fehlersyndrom ist nun eine Spalte in der Generatormatrix.
Ort ist eigentlich egal – es müssen nur alle linear unabhängig sein.
ABER: um die Position des Fehlers zu bestimmen, fügen wir das Fehlersyndrom
in die Matrix an der Position des Bitfehlers, welcher das Syndrom generiert hat, ein.

Gültiges Codewort: **1 0 0 0 1 0 1**

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

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

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

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

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

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

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

Generatormatrix →

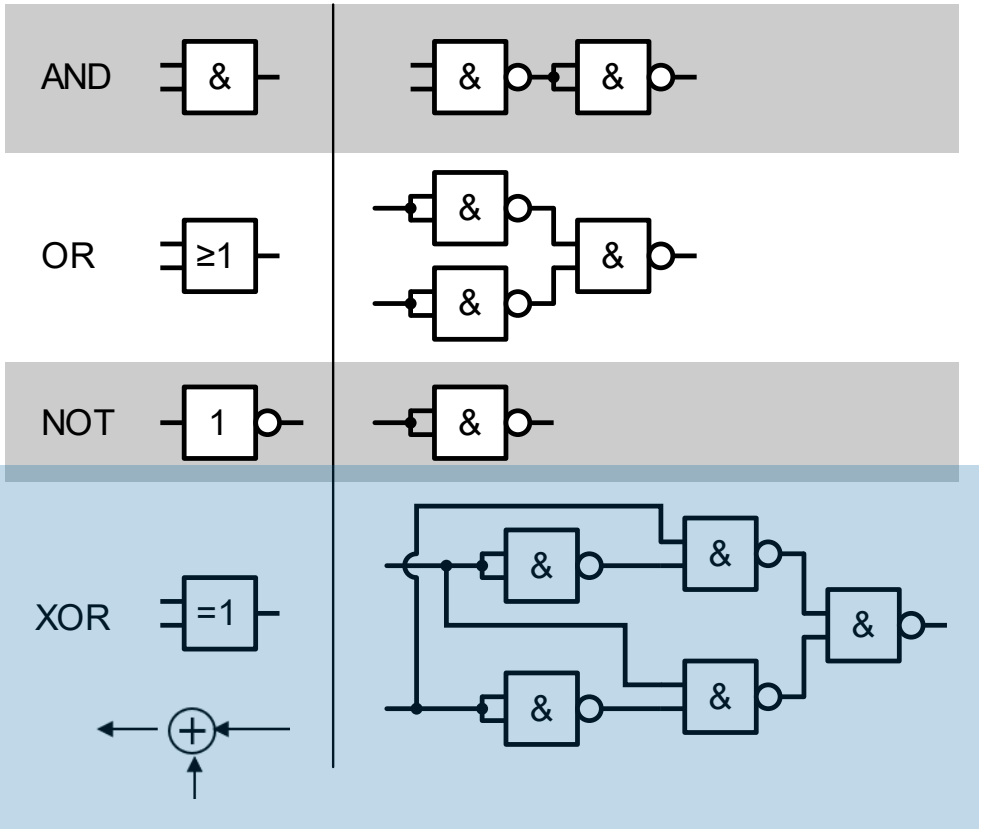
$$\begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

OST

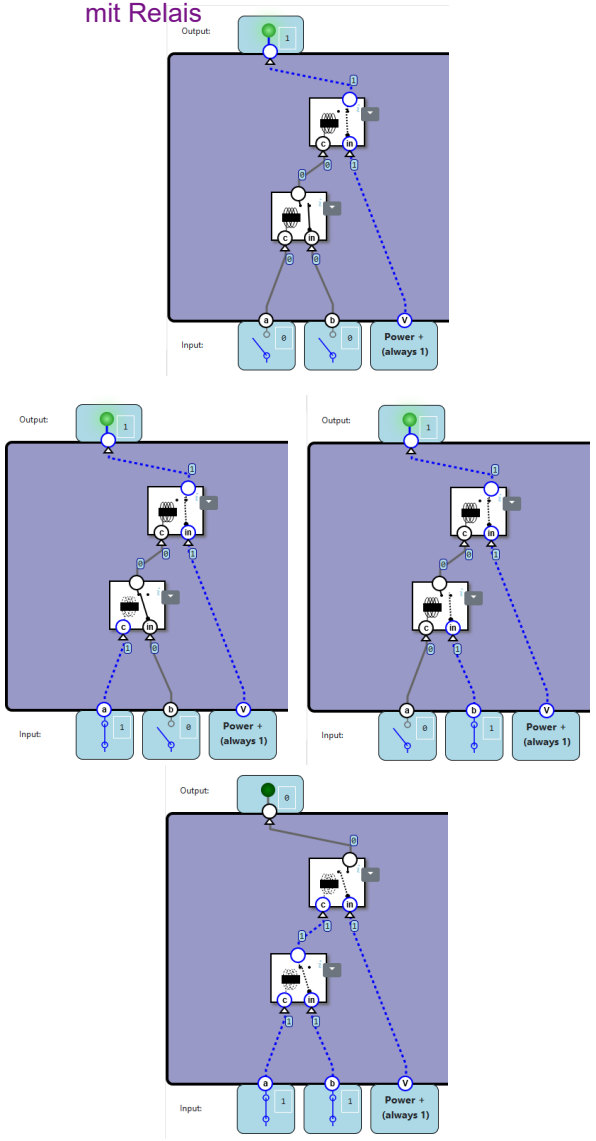
Zyklische Codes: Implementierung in Hardware

Rechts: NAND mit Relais (aus NAND-Game)
Links: wie können nun ein XOR aus NAND gebaut werden.
Diese Folie zeigt nur, wie XOR «einfach» in Hardware umgesetzt werden kann -> als Basis für Schieberegister in nächster Folie

- **Polynomdivision lässt sich elegant in Hardware realisieren**
  - Mittels Schieberegister und XOR
- **Boolsche Logik: alle Gatter lassen sich aus NAND aufbauen**



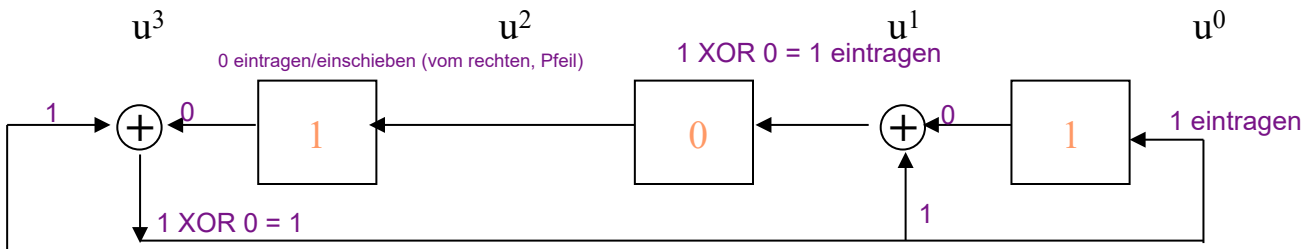
mit Relais



Zyklische Codes: Ermittlung der Kontrollstellen durch rückgekoppeltes Schieberegister

das ist ein Schieberegister

1. alle mit 0 initialisieren (die 3 Kästchen und Codewort)
2. x1 =1 nehmen und Pfeil folgen, das + ist ein XOR mit jeweiliger Zahl aus dem Schieberegister (1 XOR 0 = 1)



Sei: m = 4, k = 3, n = 7
Nachricht: $(x_1,x_2,x_3,x_4,) = (1\ 0\ 0\ 0)$
Generator: $G(u) = u^3 + u + 1$

Codewort:

	0	0	0	Vorbelegung
$x_1=1$	0	1	1	Nach Übernahme x_1
$x_2=0$	1	1	0	Nach Übernahme x_2
$x_3=0$	1	1	1	Nach Übernahme x_3
$x_4=0$	1	0	1	Nach Übernahme x_4

Ermittelten Kontrollstellen

Takt gegeben _|-|_|-|_|-|_

Rückkopplung ist der Zyklus (also das es gespeichert wird und zurück gegeben wird)

⊕ Modulo 2 Addierer (XOR)

höchster Grad vom Polynom, entscheidet codeblocklänge und Kontrollstellen

Zyklische Hamming-Codes:
Hammingdistanz h=3
Diese werden gebildet durch sogenannte primitive Polynome p(x) = g(x):

$p(x) = 1+x+x^3$
$p(x) = 1+x+x^4$
$p(x) = 1+x^2+x^5$
$p(x) = 1+x+x^6$
$p(x) = 1+x^3+x^7$
$p(x) = 1+x^2+x^3+x^4+x^5+x^6+x^7$
$p(x) = 1+x^2+x^3+x^4+x^5+x^8$
$p(x) = 1+x^4+x^9$
$p(x) = 1+x^3+x^{10}$
$p(x) = 1+x^2+x^{11}$
$p(x) = 1+x+x^4+x^6+x^{12}$
$p(x) = 1+x+x^3+x^4+x^{13}$
$p(x) = 1+x^2+x^6+x^{10}+x^{14}$
$p(x) = 1+x+x^{15}$
$p(x) = 1+x^5+x^{23}$
$p(x) = 1+x+x^2+x^4+x^5+x^7+x^8+x^{10}+x^{11}+x^{12}+x^{16}+x^{22}+x^{23}+x26^{6}+x^{32}$

zyklische Polynome von Galois

Zyklische Abramson-Codes bzw. CRC-Codes:
Hammingdistanz h=4
Diese werden gebildet durch die Multiplikation eines primitven Polynoms mit dem Term (1+x)

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

Polynom mit 1+x multiplizieren, dann ist es nicht mehr primitiv.

Bsp.:

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

höchste Potenz hat sich um 1 erhöhtZyklus hat sich nicht verändert.
gleiche Hammingdistanz
Anzahl Nachrichtenstellen -1, Kontrollstellen +1

Aus: Martin Werner, Information und Codierung, vieweg 2002

OST

Teil 1: Mathematische Grundlagen zu Zahlen und Algebra

1. **Mathematische Grundlagen: Das Stellenwertsystem**
2. **Binärzahlen: Praktisch angeschaut**
3. **Eine (nicht) mathematische Einführung in die Gruppentheorie**
   - Was ist Gruppe, Ring, Körper?
   - Modulo Rechnung
   - Interpretation eines Datenworts als
     - Tupel, Zahl, Vektor, Polynom
   - Was ist eine zyklische Gruppe
   - Bool`sche Alegra
4. **Erste Schritte in den Wahrscheinlichkeitsbegriff und die Kombinatorik**

Teil 2: Codierungs- und Informationstheorie Grundlagen

1. **Einführung in die Informationstheorie**
2. **Quellencodierung**
   - Komprimierung
   - Verschlüsselungsverfahren
3. **Kanalmodell**
4. **Kanalcodierung**
   - Blockcodes
   - Faltungscodes

Der Faltungscode ist ein Stromcode mit Gedächtnis, kann komplexere Fehler erkennen/korrigieren und wird für kontinuierliche Datenströme verwendet.

Faltungscodes

- **Einführung**
- **Definition**
- **Encoderschaltung**
- **Zustandsdarstellung**
- **Übertragungsfunktion**
- **Optimale Codes**
- **Trellisdiagramm (Decodierung)**

Faltungscode
Struktur: Keine feste Blocklänge, sondern beliebig lange Bitströme werden mit Schieberegistern und Generatorpolynomen codiert.
Eigenschaft: Die Codierung hängt nicht nur von den aktuellen, sondern auch von vorherigen Eingabebits ab (Gedächtnis!).
Fehlerbehandlung: Kann mehrere Fehler erkennen und korrigieren, hängt von Codeparametern ab.
Codierung: Mit Schieberegister und mehreren Generatorpolynomen; jedes Eingabebit beeinflusst mehrere Ausgabebits über mehrere Zeitpunkte hinweg.
Anwendung: Vor allem in der Satellitenkommunikation, Mobilfunk, digitalen TV, wo lange oder kontinuierliche Datenströme codiert werden.
Decodierung: Häufig mit dem Viterbi-Algorithmus.

OST

Die ersten Arbeiten zu Block- und Faltungscodes gehen auf die 50er Jahre zurück.

- **Blockcodes wurden schnell zur Sicherung gegen Übertragungsfehler eingesetzt, nicht zuletzt wegen ihrer einfachen Implementierung.**
- **Eigenschaften:**
  - Einfache Implementierung der Encoder und Decoder durch Schieberegister
  - Hohe Fehlererkennungsmächtigkeit (Bündelfehlererkennung bei zyklischen Codes)
  - **Blockbildung der zu codierenden Daten notwendig - entsprechend keine fortlaufende Codierung möglich!**
  - **Eine fortlaufende Codierung ist aber mit Faltungscodes möglich!**

10-15 Jahre später erst

Für Faltungscodes wurde erst 1967 ein effizienter Algorithmus zur Dekodierung (Viterbi-Algorithmus) gefunden (Einsatz in den Mobilfunkstandards wie GSM, UMTS/3G, 4G/LTE,)

- **Eigenschaften:**
  - Faltungscodes erlauben fortlaufende Codierung eines kontinuierlichen Datenstroms (keine Blockbildung erforderlich)
  - Decodierung von Faltungscodes benötigt keine Blocksynchronisation
  - Gute Faltungscodes werden durch Rechnersimulation [sic!] gefunden

bei Faltungscodes geht es um Polynome

Vergangenheit von vorherigem Code wird mitberücksichtigt.

Der Viterbi-Algorithmus ist ein Maximum-Likelihood-Decodierverfahren für Faltungscodes.
Ein Faltungscode wird durch seine Generatorpolynome beschrieben.
Diese Polynome bestimmen, wie die Eingabebits verschaltet werden, um die Ausgabebits zu erzeugen.

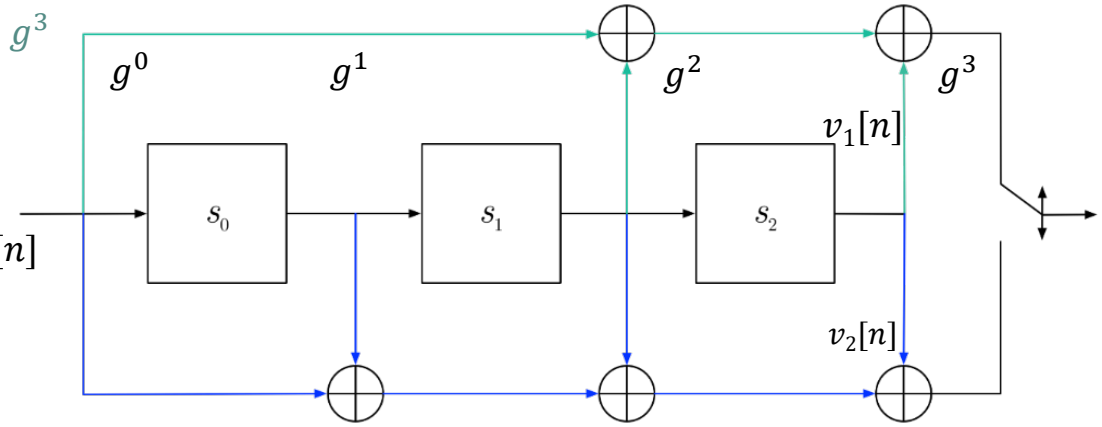
Coderate
Rate 1/2: Ein Eingabebit → Zwei Ausgabebits
Rate 1/3: Ein Eingabebit → Drei Ausgabebits

(Ausgänge , Anzahl Eingänge ,Speicher)

Encoderschaltung des (2,1,3) Encoders
Idee: die Vergangenheit berücksichtigen, ein Beispiel I

2 Polynome
g1 und g2?

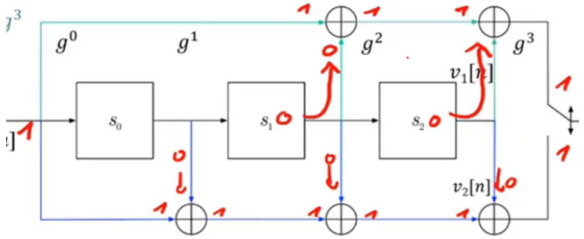
$\{g_1\} = \{1\ 0\ 1\ 1\}$
oder
$G_1(x) = 1 + g^2 + g^3$



Nachrichtenfolge $u[n]$

Codefolge $v[n_2]$

$\{G_2\} = \{1\ 1\ 1\ 1\}$
oder
$G_2(x) = 1 + g + g^2 + g^3$



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

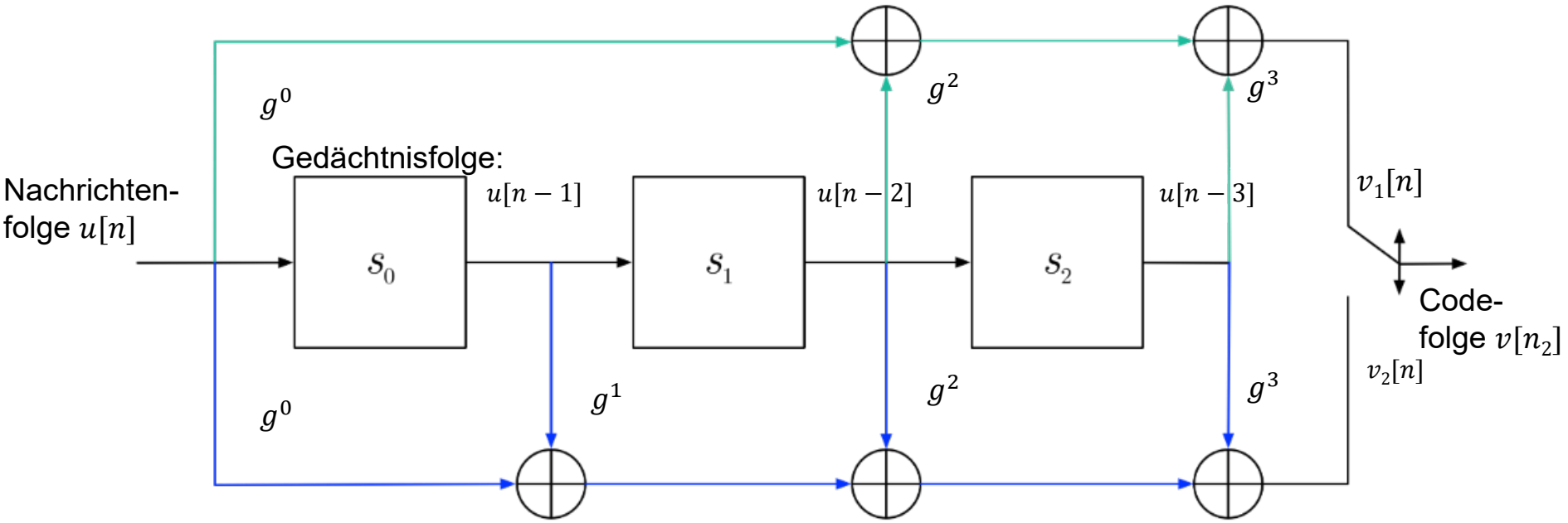
Die Speicherplätze s_0, s_1, s_2 sind mit 0 vorbelegt.

$u[n]$	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

OST

Encoderschaltung des (2,1,3) Encoders
Idee: die Vergangenheit berücksichtigen, ein Beispiel II



Nachrichtenfolge: $\{u[n]\} = \{u_1, u_2, \dots, u_n\}$

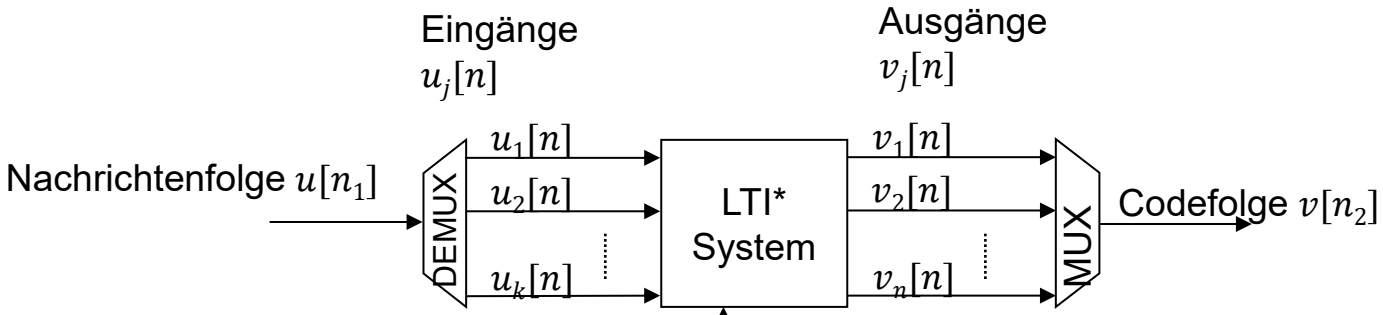
Ausgangsfolgen: $\{v[n]\} = \{[\, g^0 u_{[n]} + g^1 u_{[n-1]} + g^2 u_{[n-2]} + g^3 u_{[n-3]}], [g^0 u_{[n+1]} + g^1 u_{[n+1-1]} + \cdots, \dots, \}$

$$\{v[n]\} = \sum_{m=0}^{M} g^m \, u_{[n-m]}$$

OST

Binärer Faltungsencoder: Verallgemeinerung

Die Ausgangsfolge wird durch die Faltung der Eingangsfolgen erzeugt (zur Vertiefung siehe auch Z-Transformation)



$$v_j[n] = \sum_{i=1}^{k} g_{ji}[n] \cdot u_i[n] = \sum_{i=1}^{k} \sum_{m=0}^{Mi} g_{ji}[m] \cdot u_i[n-m]$$

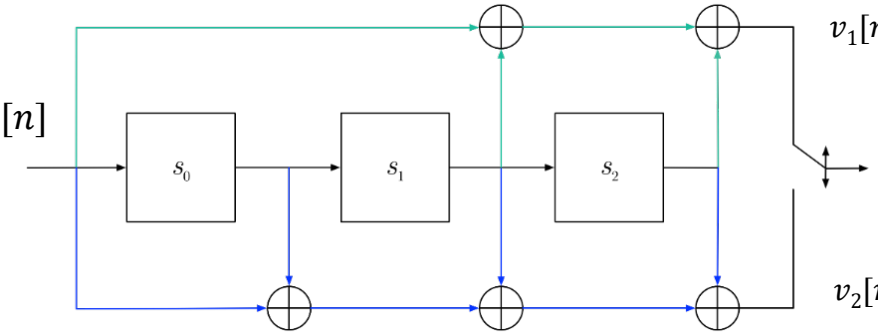
Faltung, der Begriff: mathematisches Thema, Faltungsalgebra. Das ist die Faltung. Irgendeine Fläche der Polynome...

*LTI: Linear Time Invariant

OST

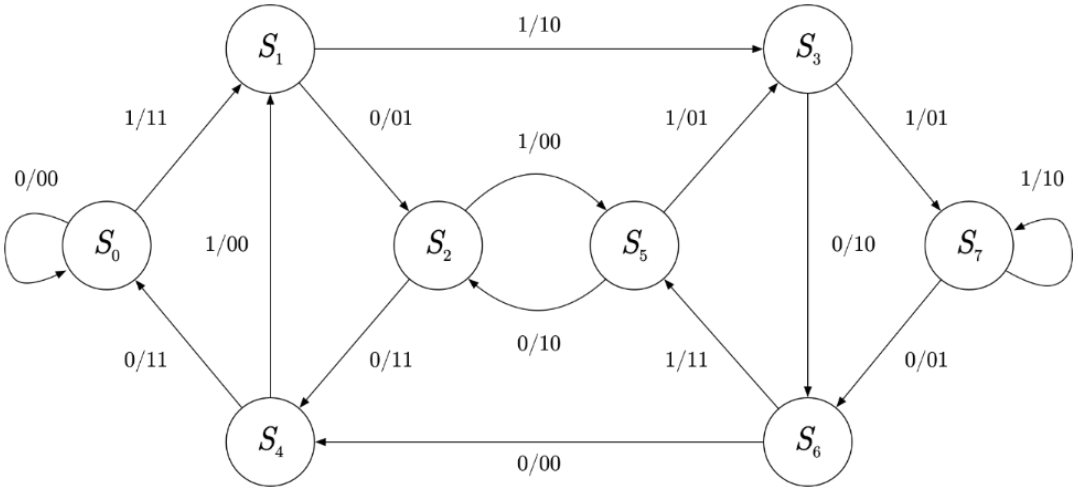
Zustandsdarstellung des (2,1,3) Encoders

Nachrichtenfolge $u[n]$



Codefolge $v[n_2]$

als Zustands-Diagramm aufgezeichnet:

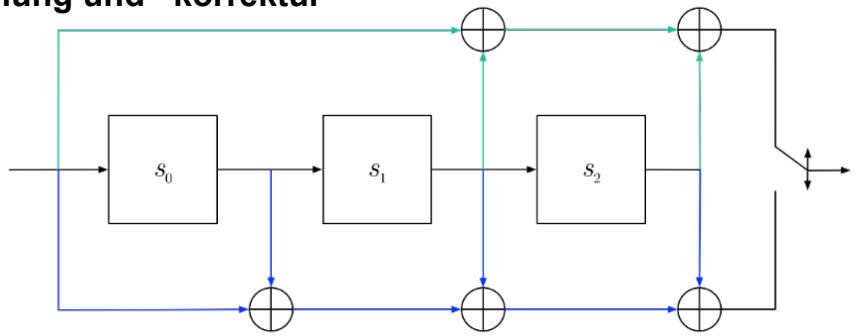


Zustands-grösse			Zustand S_i
s_0	s_1	s_2	$(i = s_0 \cdot 2^0 + s_1 \cdot 2^1 + s_2 \cdot 2^2)$
0	0	0	0
1	0	0	1
0	1	0	2
1	1	0	3
0	0	1	4
1	0	1	5
0	1	1	6
1	1	1	7

OST

Impulsantwort eines Faltungscodierers

- **Die Impulsantwort beschreibt, wie ein Faltungscodierer auf einen einzelnen Impuls in der Eingabefolge reagiert**
- **Sie zeigt, wie ein einzelner Eins-Impuls in eine Sequenz von Ausgangsbits umgewandelt wird**
- **Wichtige Eigenschaft zur Analyse der Fehlererkennung und –korrektur**



- **Beispiel**
  - Eingangsimpuls: 1, 0, 0, 0, …
  - Impulsantwort: 11, 01, 11, 00
- **Erkenntnisse**
  - **Länge der Impulsantwort**: Der Codierer verteilt die Information über vier Zeitschritte hinweg.
  - **Fehlererkennung und -korrektur**: Ein Codierer mit einer längeren und komplexeren Impulsantwort hat tendenziell bessere Fehlerkorrektureigenschaften, da er mehr Informationen über die Datenverteilung über die Zeit hinweg speichert.
- **Praktische Bedeutung**
  - **Erinnerungskapazität**: Dieser Codierer hat eine Erinnerung von vier Zeitschritten. Nach vier Zeitschritten ist der Einfluss eines Eingangsimpulses nicht mehr in der Ausgabe sichtbar.
  - **Redundanz**: Verdeutlicht die Redundanz, die der Codierer in die Ausgabe einfügt. Diese ist entscheidend für die Fehlerkorrektur.
  - **Analyse der Codierleistung**: Durch das Untersuchen der Impulsantwort kann die Leistungsfähigkeit des Codierers in Bezug auf Fehlererkennung und -korrektur bewerten werden.

OST

Aufgabe

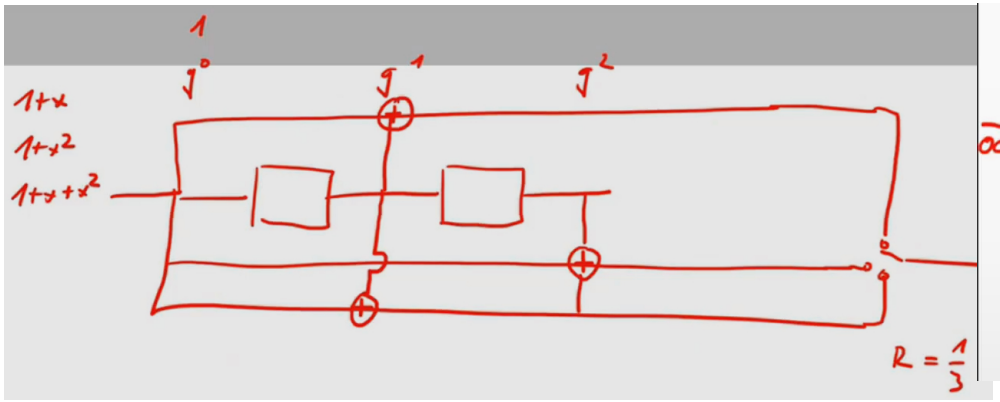
Coderate R = 1 / anzahl Ausgänge

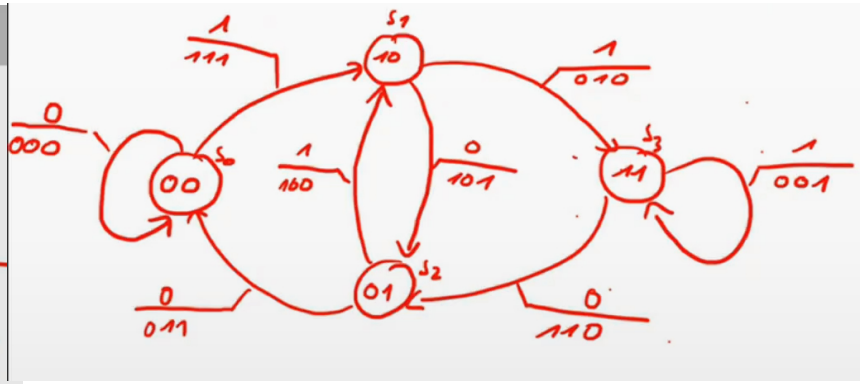
Gegeben ist ein (3,1,2) Faltungscode mit

- $g_1(x) = 1 + x$
- $g_2(x) = 1 + x^2$
- $g_3(x) = 1 + x + x^2$

bei 3 Ausgängen wie hier gibt es 3 Polynome

Ermitteln Sie den Codierer und das Zustandsdiagramm!

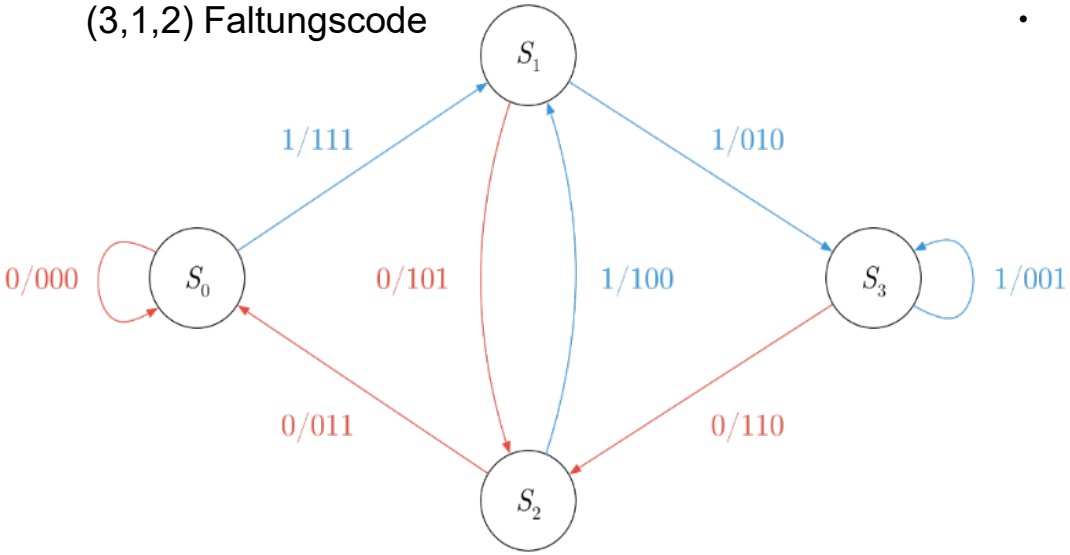




OST

Struktur: Fundamentalweg und Gewichte

Beispiel:
(3,1,2) Faltungscode

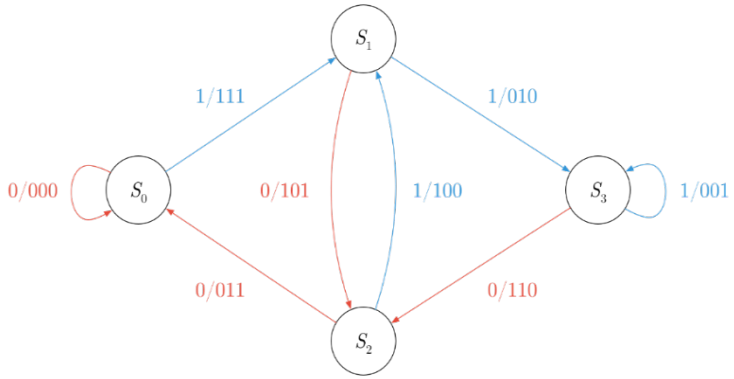


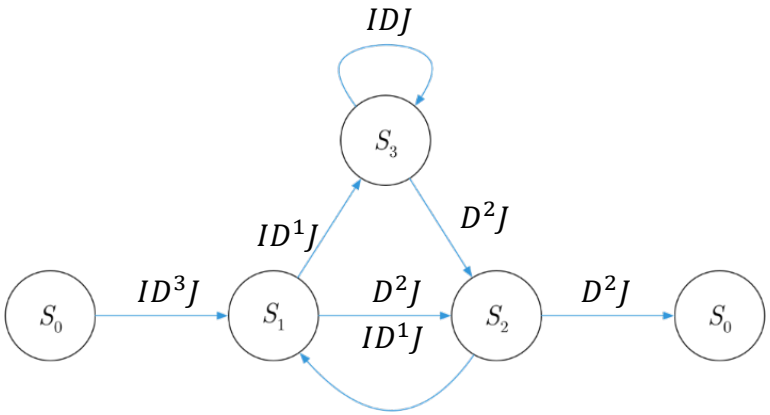
Definitionen

- *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.

OST

Struktur: Fundamentalweg und Gewichte





Definitionen

- *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: bezeichne den Zustandsübergang, der durch «1» ausgelöst wird.
  - D^l: bezeichne die Anzahl der durch den Übergang zur Codefolge hinzukommenden «1» Bitstellen (Gewichtszunahme).
  - J: sei eine Zählvariable, die die Anzahl der Übergänge zählt.
  - Jede Kante eines Fundamentalweges lässt sich durch das Triplet ($I\ D^l\ J$) beschreiben.
    → *Kantengewicht*

Beispiel:

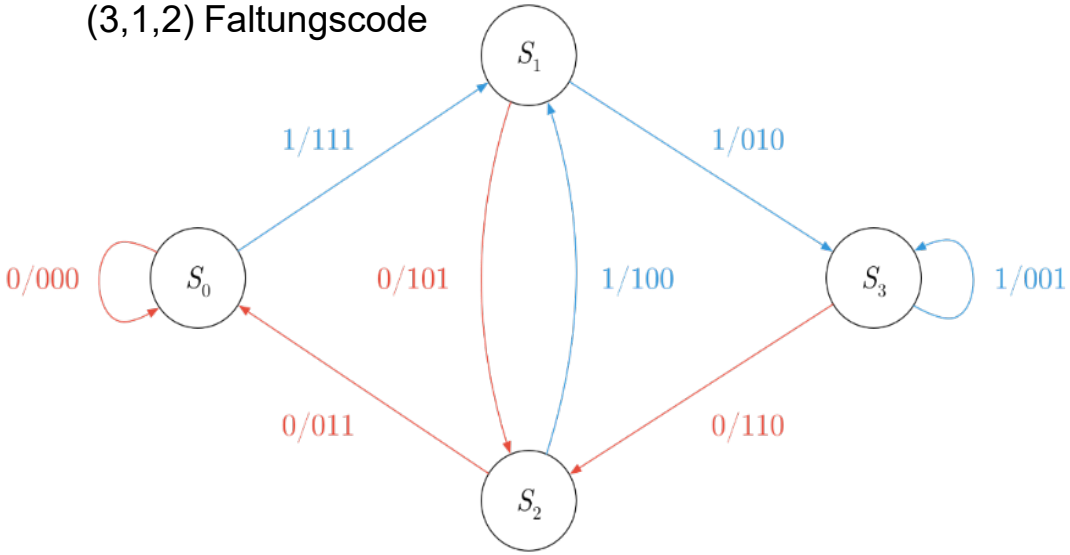
$\{g[n]\} = \{1\ 0\ 0\} \Rightarrow ID^3J \cdot D^2J \cdot D^2J = ID^7J^3$

: Gesamtgewicht der Folge

OST

Impulsantwort → Generatorpolynome

Beispiel:
(3,1,2) Faltungscode

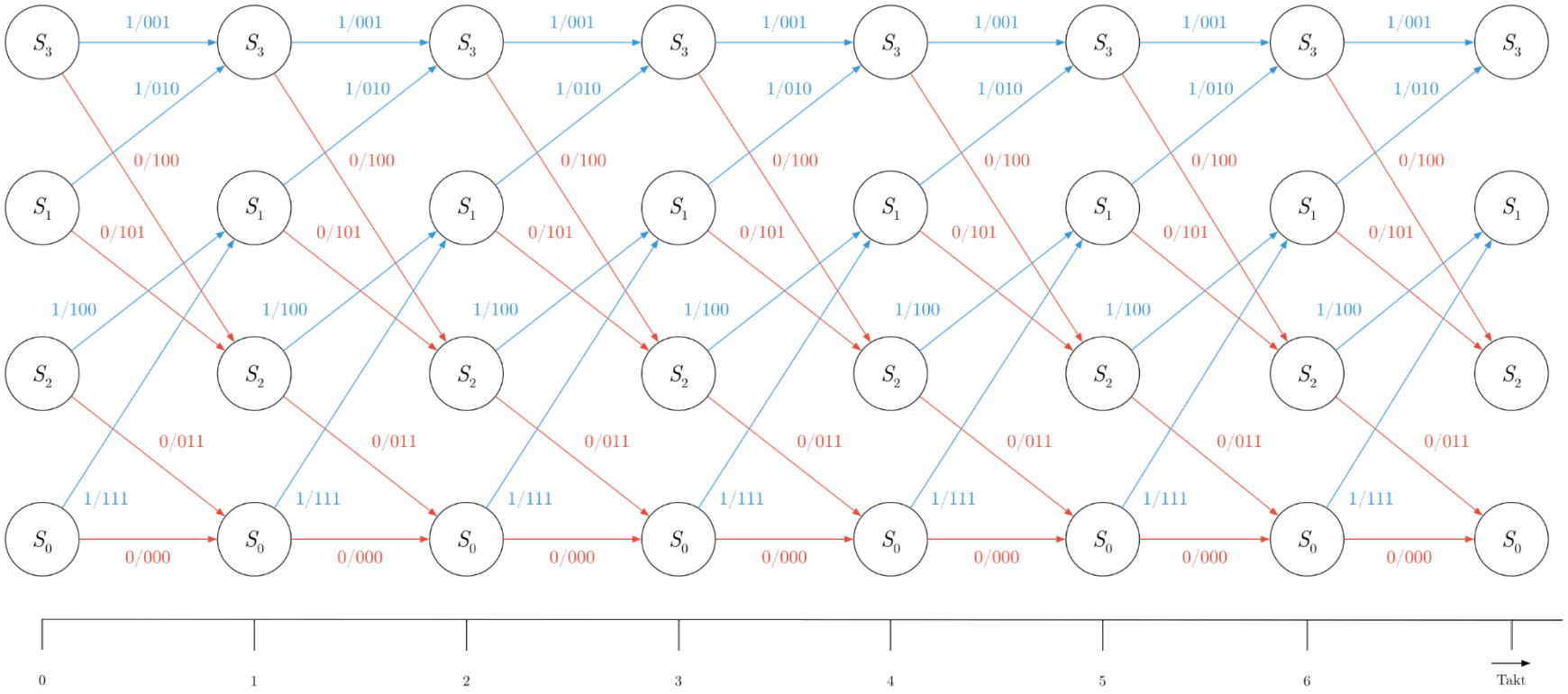


Generatorpolynom:
- Typ: Mathematisches Werkzeug
- Verwendung: Definiert die Struktur von zyklischen Codes (z.B. CRC, zyklische Hammingcodes, BCH, Reed-Solomon, Faltungscodes)
- Funktionsweise: Gibt an, wie Paritäts- oder Redundanzbits berechnet werden.
  - Im CRC teilt man die Nachricht durch das Generatorpolynom;
  - bei Faltungscodes beschreibt es die Verschaltung der Speicherstellen.

OST

Netz-/Trellisdiagramm des (3,1,2) Faltungscodes

Zustandsdiagramm für Generatorpolynom



OST

Codierung des (3,1,2) Faltungscodes

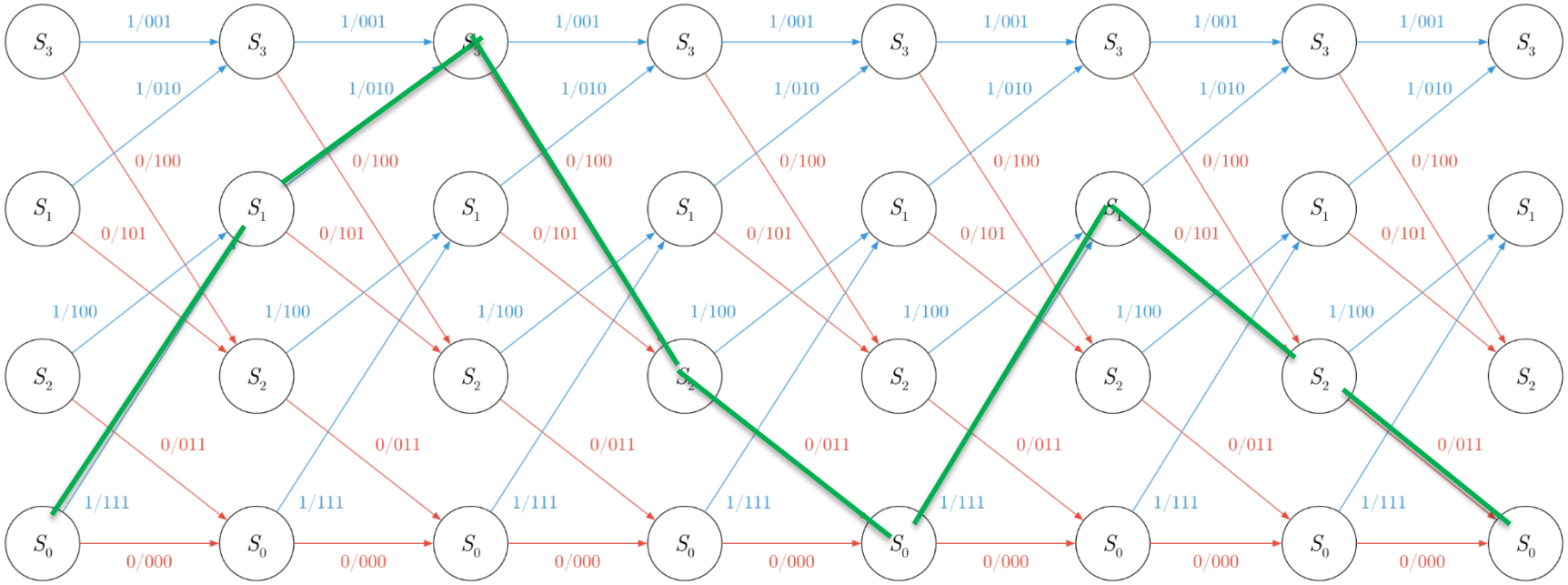
Jede Spalte ist gleich, bei Zeittakt ein bit einlesen von aktuellem Zustand aus.

Am Schluss bei Zustand 0 landen, damit es ein Fundamentalweg (siehe oben) ist.

(ist wie traveling salesman/handelsreisender Problem)

Trellis Diagramm Beispiel

ganz einfach, ablesen und zahlen neben bit aufschreiben.
Keine Kosten berechnen!



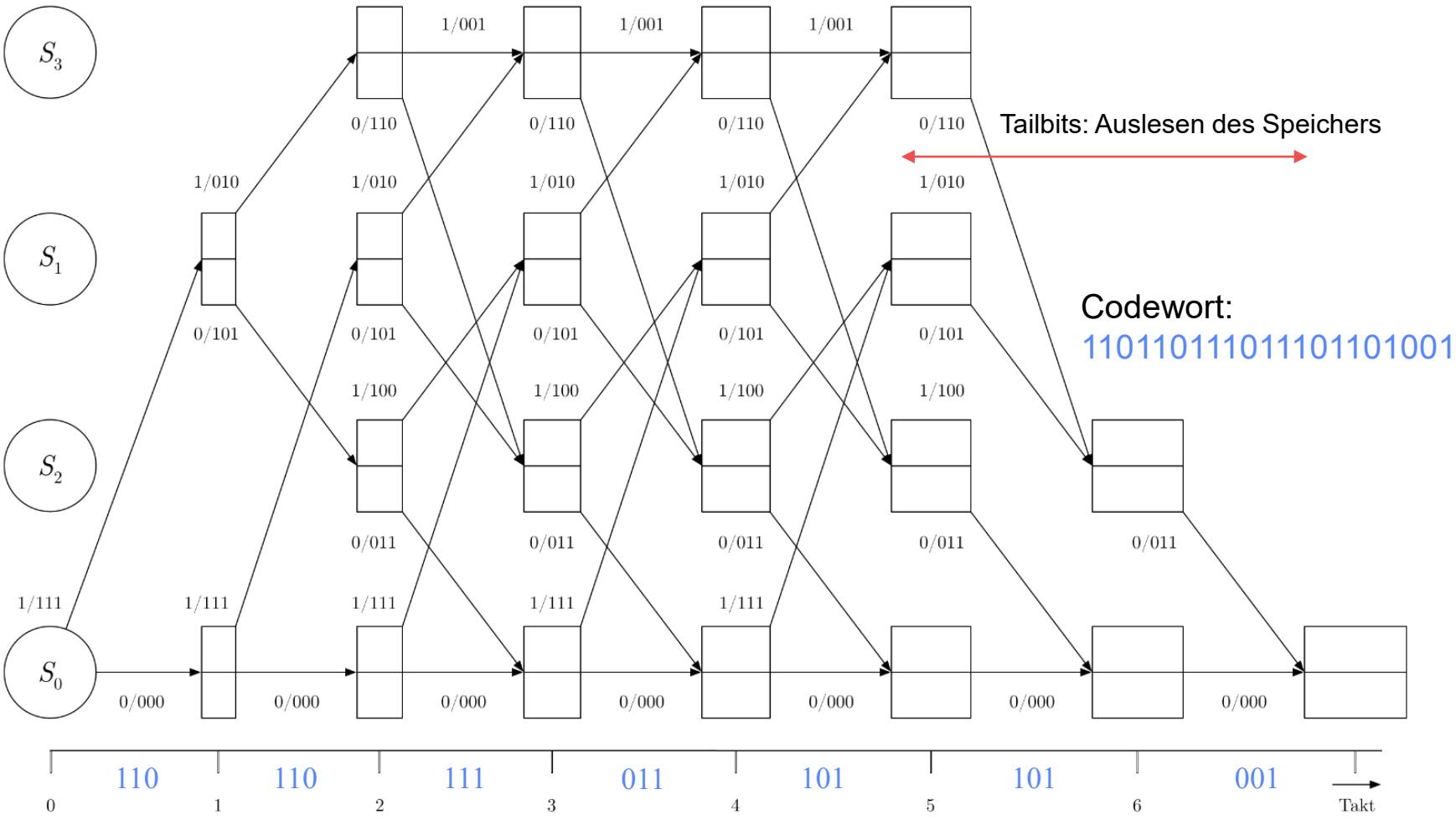
nachrichtenwort:

gibt Codewort (ablesen)
111 010 100 011 111 101 011

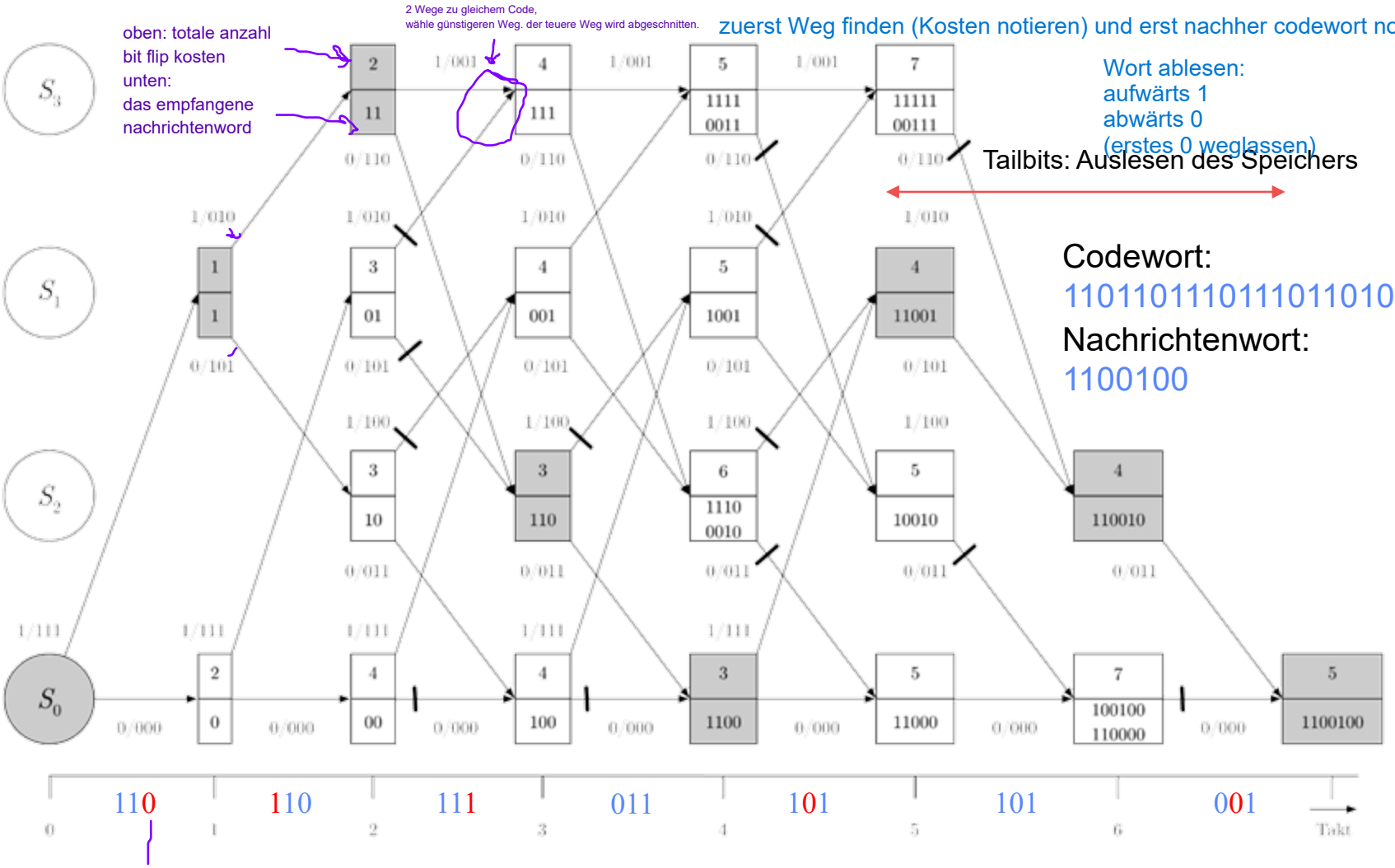
OST

Decodierung des (3,1,2) Faltungscodes

Das ist der Viterbi Algorithmus!



Decodierung des (3,1,2) Faltungscodes



hier ist ein Fehler passiert,
weil nur 111 oder 000 sind möglich.
Zum korrigieren wird der Weg
mit den geringsten Kosten gesucht.

Als 0 korrigieren kostet 2 bit flips, als 1 korrigieren kostet 1 bit flip.
Diese Kosten werden in Kasten notiert.

OST

Generatorpolynome für optimale Faltungscodes

Coderate R = 1 / anzahl Ausgänge

(Ausgänge , Anzahl Eingänge ,Speicher)

Beispiel (2,1,4)-Code = m=4, R=1/2
g1 = 23oct = 010 011 -> Generatorpolynom ist x^4 + x^3 + 1
g2 = 35oct = 011 101 -> Generatorpolynom x^4 + x^2 + x + 1
df = 7 ist wie Hammingdistanz ein Wert zum wissen, wieviele Fehler erkennt werden

Tabelle lesen in Zusammenfassung!

Coderate
1 Eingang, 2 Ausgänge (pro Takt?)

1 Eingang, 3 Ausgänge

1 Eingang, 4 Ausgänge

(Gedächtnis-)Speicher 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 (2,1,4)-Code	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

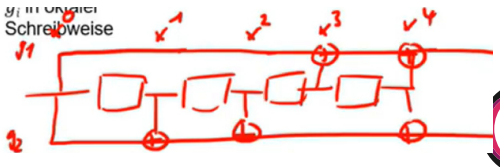
oktal schreibweise = jede Stelle ist 3 bit.

g_i in oktaler Schreibweise

[Martin Werner, Information und Codierung, vieweg]

Generatorpolynom gibt immer 8x, und 1 ist ja auch x^0 (?)

Maximale Potenz bei Generatorpolynom ist x^m, also im Beispiel x^4
- Speicher = höchster Grad des Polynoms



OST