Inhaltsverzeichnis

1 Mathematische Grundlagen

1.1 Speichereinheiten

Kilobyte (KB)	10^3	Kibibyte (KiB)	2^{10}	1024
Megabyte (MB)	10^6	Mebibyte (MiB)	2^{20}	1'048'576
Gigabyte (GB)	10^9	Gibibyte (GiB)	2^{30}	1'073'741'824
Terabyte (TB)	10^{12}	Tebibyte (TiB)	2^{40}	
Petabyte (PB)	10^{15}	Pebibyte (PiB)	2^{50}	

Tabelle 1: Vergleich von Speichereinheiten in Dezimal und Binär

1.2 Interpretation einer Menge von Bits

- Dualzahl: 10101001
- Tupel: (1, 0, 1, 0, 1, 0, 1)
- Menge: {0, 1}
- Vektor: $\begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$
- Polynom: $1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$
- Zustand: z.B. Zustand eines Speicherregisters (Adressierung, Anweisung)

1.3 Dezimalzahl in Dualzahl umformen

Um Ganzzahlen in Binär zu konvertieren, wird der Rest von rechts nach links (LSB nach MSB) aufgeschrieben. Bei Kommazahlen wird der Rest von links nach rechts (MSB nach LSB) aufgeschrieben.

1.3.1 Ganzzahlen

$$\left.\begin{array}{rl} 11 : 2 \cdot 2 = 5 & \text{Rest :1} \\ 5 : 2 \cdot 2 = 2 & \text{Rest :1} \\ 2 : 2 \cdot 2 = 1 & \text{Rest :0} \\ 1 : 2 \cdot 2 = 0 & \text{Rest :1} \end{array}\right\} 1011$$

1.3.2 Kommazahlen

Möchten wir z.B. die Zahl 0.1 in Binär umrechnen, multiplizieren wir sie so lange, wie der definierte Datentyp Bits hat. D.h wenn wir 0.1 (dec) mit einem 8Bit Datentyp in Binär umrechnen, multiplizieren wir 8-mal. Dabei wird die Zahl vor dem Komma zum Resultat angehängt und mit der Zahl nach dem Komma wird weiter gerechnet.

$$\left.\begin{array}{ll} 0.625 \cdot 2 = 1.25 & \text{Rest :1} \\ 0.25 \cdot 2 = 0.5 & \text{Rest :0} \\ 0.5 \cdot 2 = 1 & \text{Rest :1} \end{array}\right\} 0.101$$

oder ein anderes Beispiel:

$$\left.\begin{array}{l} 0.1 \cdot 2 = 0.2 \\ 0.2 \cdot 2 = 0.4 \\ 0.4 \cdot 2 = 0.8 \\ 0.8 \cdot 2 = 1.6 \\ 0.6 \cdot 2 = 1.2 \\ 0.2 \cdot 2 = \ldots \end{array}\right\} 0.00011\ldots$$

1.4 Dualzahl in Dezimalzahl umformen

1.4.1 Ganzzahlen

$$\begin{array}{rcl} 1011 & \Rightarrow & 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \\ & \Rightarrow & 8 + 0 + 2 + 1 = 11 \end{array}$$

1.4.2 Kommazahlen

$$\begin{array}{rcl} 0.101 & \Rightarrow & 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} \\ & \Rightarrow & 0.5 + 0.125 = 0.625 \end{array}$$

1.5 Operationen mit Dualzahlen

1.5.1 Addition von Dualzahlen

$$\begin{array}{r} 111_2 \\ +101_2 \\ \hline 1100_2 \end{array}$$

1.5.2 Subtraktion von Dualzahlen

$$\begin{array}{r} 111_2 \\ -101_2 \\ \hline 0010_2 \end{array}$$

1.6 Subtraktion durch Addition

Beispiel: $75_{10} - 26_{10}$

$$100 - 26 = 74 \quad (1)$$
$$74 + 75 = 149 \quad (2)$$
$$149 - 100 = \underline{\underline{49}} \quad (3)$$

1 10er-Komplement einer Zahl wird berechnet, indem man die Differenz zur nächsten 10er-Potenz verwendet. (In diesem Beispiel ist 100 die nächst grössere Zehnerpotenz von 26) und am Ende +1 dazu addiert.

2 Nun die Addition: $74 + 75 = 149$

3 Da es einen Überlauf der Zahl 100 gibt: 149 - 100 = 49

Beispiel: $174_8 - 54_8$

$$777_8 - 54_8 + 1 = 724_8 \quad (4)$$
$$724_8 + 174_8 = 1120_8 \quad (5)$$
$$1120_8 - 1000_8 = \underline{\underline{120_8}} \quad (6)$$

1 Das 8er-Komplement einer Zahl wird berechnet, indem man die Differenz zur nächsten höheren Potenz von 7_8 bestimmt (In diesem Beispiel haben wir 174, somit ist 777 die nächst höhere Zahl) und am Ende +1 dazu addiert.

2 Nun die Addition: $724_8 + 174_8 = 1120_8$.

3 Da es einen Überlauf der Zahl 1000_8 gibt: $1120_8 - 1000_8 = 120_8$.

1.7 Fixkommazahl

Bei einer 10-Bit-Datentyp mit der eine Zahl in der Range [0;4[dargestellt werden soll, hat die 2 Bits vor dem Komma und 8 Bits nach dem Komma. D.h. die kleinste Zahl k > 0 ist $\frac{1}{2^8} = \frac{1}{256}$

Die grösste Zahl ist $3 + (1 - \frac{1}{256})$

1.8 Fliesskommazahlen

Bei der Fliesskommazahl-Darstellung (z. B. IEEE-754) wird eine Zahl durch Vorzeichen, Exponent und Mantisse dargestellt. Für eine vereinfachte 10-Bit-Darstellung mit 1 Bit Vorzeichen, 4 Bits Exponent (Bias 7) und 5 Bits Mantisse (inkl. hidden Bit) kann der Bereich angepasst werden.

Bias-Grösse:

- 16-Bit Half Precision: 15
- 32-Bit Single Precision: 127
- 64-Bit Double Precision: 1023

Allgemeine Formel:

$$\text{Wert} = (-1)^s \cdot 2^{e-\text{Bias}} \cdot (1 + m)$$

- s: Vorzeichenbit (0 oder 1),
- e: Exponent (aus 4 Bits, Bias = 7),
- m: Mantisse (Summe der 4 Mantissenbits als $\sum_{i=1}^{4} b_i \cdot 2^{-i}$).

Beispiel:
0|1000 0001|0100 0110 0110 0110 0110 011

- Vorzeichen: 0 (positiv),
- Exponent: $10000001_2 = 129_{10} - 127 = 2$,
- Mantisse: $\sum_{i=1}^{32_{\text{bit}}-\text{Vorzeichen}-\text{Exponent}} b_i \cdot 2^{-i}$,
- Wert: $(-1)^0 \cdot 2^2 \cdot (1 + 0.3999999761581421) \approx 5.199999904632568$.

$$
\begin{aligned}
m &= \sum_{i=1}^{23} b_i \cdot 2^{-i} \\
&= 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0 \cdot 2^{-4} + 0 \cdot 2^{-5} + 1 \cdot 2^{-6} + 1 \cdot 2^{-7} \\
&+ 0 \cdot 2^{-8} + 0 \cdot 2^{-9} + 1 \cdot 2^{-10} + 1 \cdot 2^{-11} + 0 \cdot 2^{-12} + 0 \cdot 2^{-13} \\
&+ 1 \cdot 2^{-14} + 1 \cdot 2^{-15} + 0 \cdot 2^{-16} + 0 \cdot 2^{-17} + 1 \cdot 2^{-18} + 1 \cdot 2^{-19} \\
&+ 0 \cdot 2^{-20} + 0 \cdot 2^{-21} + 1 \cdot 2^{-22} + 1 \cdot 2^{-23} \\
&= 0 + 0.25 + 0.125 + 0 + 0 + 0.015625 + 0.0078125 \\
&+ 0 + 0 + 0.0009765625 + 0.00048828125 + 0 + 0 \\
&+ 0.00006103515625 + 0.000030517578125 + 0 + 0 + 0.000003814697265625 + 0.0000019073486328125 \\
&+ 0 + 0 + 0.00000023841857910156 25 + 0.00000011920928955078125 \\
&\approx 0.3999999761581421
\end{aligned}
$$

1.9 Multiplizieren mit binären Vektoren

A(x) = (1 0 1 1 1)
B(x) = (0 1 0 1 1)
A(x) · B(x) = (1 0 1 1 1) · (0 1 0 1 1)

$$
\begin{aligned}
1(1^4) \cdot (1011) &= 10110000 \\
0(1^3) \cdot (1011) &= 0 \\
1(1^2) \cdot (1011) &= 101100 \\
1(1^1) \cdot (1011) &= 10110 \\
1(1^0) \cdot (1011) &= 1011
\end{aligned}
$$

Daraus entsteht dann die folgende (schriftliche) Addition:

$$
\begin{array}{r}
1011'0000 \\
+\ \ 10'1100 \\
+\ \ \ 1'0110 \\
+\ \ \ \ \ 1011 \\
\hline
\underline{\underline{1111'1101}}
\end{array}
$$

Wenn die Regel $\mathbb{Z}_2$ gilt, gibt es kein “carry-over” (1 + 1 = 0). Das heisst, dass das Resultat derselben Rechnung folgendermassen aussehen würde: $\underline{\underline{1000'0001}}$

1.10 Zahlenkomplemente

1.10.1 Zweierkomplement

MSB wird verwendet, um Zahl zu invertieren. z.B. wäre 1011 (4Bit) wäre $-1000 + 011 = -8 + 3 = -5$

Es gibt signed und unsigned Datentypen, die wie folgt definiert sind:

z.B. unsigned char (8 Bit): 0 . . . 255

signed char (8 Bit): -128 . . . +127

Berechnung:

1 **Bitweise invertierung (Einerkomplement)**: Z.B. 1100 -> 0011

2 **Addiere 1**: Z.B. 0011 $\rightarrow \underline{\underline{0100_b = -4_d}}$ (Es ist -4_d, da die Zahl 1100 eine führende **1** hat)

1.10.2 Einerkomplement ($(b-1)$-Komplement)

Das $(b-1)$-Komplement invertiert alle Ziffern einer Zahl. Im Binärsystem ($b=2$): 0 wird 1, 1 wird 0. MSB = 1 bedeutet negativ. Im Dezimalsystem ($b=10$): 9er-Komplement, z. B. 1234 wird $9999-1234=8765$.

(Das Komplement wird nur für die negative Zahl berechnet)

Binärbeispiele (3 Bit):

- $010_2 = 2_{10}$: MSB = 0, positiv, Wert = 2.
- $101_2 = -2_{10}$: MSB = 1, negativ, invertiere $01_2 = 1_{10}$, Wert = -2_{10}.

Dezimalbeispiel: $-1234+5000$ (4 Ziffern):

1 $5000-1234$: 9er-Komplement von 1234 ist $9999-1234=8765$.

2 Addiere: $5000+8765=13765$.

3 Überlauf: $13765-10000=3765$, plus Carry 1: $3765+1=3766$.

Ergebnis: $\underline{\underline{3766}}$.

1.10.3 Exzess-Darstellung

In der Exzess-Darstellung wird eine Zahl n durch Addition eines festgelegten Bias 2^{k-1} (für k Bits) codiert, um positive und negative Werte binär darzustellen. Zum Dekodieren wird der Bias subtrahiert.

Binär	Exzess-4 (3 Bit)	2er-Komplement
000	-4	0
001	-3	1
010	-2	2
011	-1	3
100	0	-4
101	1	-3
110	2	-2
111	3	-1

Tabelle 2: Vergleich Exzess-4 und 2er-Komplement (3 Bit)

Dezimal zu Binär im Exzess-Code
Zur Kodierung einer Dezimalzahl n addiert man den Bias 2^{k-1} und konvertiert das Ergebnis in eine k-Bit-Binärzahl.

Beispiel: Konvertierung von -5 im Exzess-16 (5 Bit):

$$-5+16=11_{10}=01011_2$$

Beispiel: Konvertierung von 34 im Exzess-128 (8 Bit):

$$34+128=162_{10}=10100010_2$$

Binär zu Dezimal im Exzess-Code
Zur Dekodierung konvertiert man die Binärzahl in eine Dezimalzahl und subtrahiert den Bias 2^{k-1}.

Beispiel: Konvertierung von 01011_2 im Exzess-16:

$$11_{10}-16=-5$$

Beispiel: Konvertierung von 10100010_2 im Exzess-128:

$$162_{10}-128=34$$

1.11 Vektoren

1.11.1 Vektor addition

Vektoraddition in $\mathbb{Z}_2$

$$\begin{bmatrix}1\\1\\0\end{bmatrix} + \begin{bmatrix}0\\0\\1\end{bmatrix} + \begin{bmatrix}1\\1\\1\end{bmatrix} + \begin{bmatrix}1\\0\\1\end{bmatrix} + \begin{bmatrix}0\\0\\1\end{bmatrix} = \begin{bmatrix}3\\2\\4\end{bmatrix} \rightarrow \begin{bmatrix}1\\0\\0\end{bmatrix} \text{ (wegen mod 2)}$$

1.11.2 Vektor multiplikation

Beispiel: Kreuzprodukt von $\vec{a} = (a_1, a_2, a_3)$ und $\vec{b} = (b_1, b_2, b_3)$:

$$\vec{a} \times \vec{b} = \begin{bmatrix}a_1\\a_2\\a_3\end{bmatrix} \times \begin{bmatrix}b_1\\b_2\\b_3\end{bmatrix} = \begin{bmatrix}a_2 \cdot b_3 - a_3 \cdot b_2\\a_3 \cdot b_1 - a_1 \cdot b_3\\a_1 \cdot b_2 - a_2 \cdot b_1\end{bmatrix}$$

1.12 Zahlen als Polynom schreiben

1.12.1 Dezimalzahl

$$\begin{aligned} 345 &\Rightarrow 3 \cdot 10^2 + 4 \cdot 10^1 + 5 \cdot 10^0 \\ 701 &\Rightarrow 7 \cdot 10^2 + 1 \cdot 10^0 \end{aligned}$$

1.12.2 Dualzahlen

$$\begin{aligned} 0010 &\Rightarrow 1 \cdot 2^1 &&\Rightarrow 2_{\text{d}} \\ 1100 &\Rightarrow 1 \cdot 2^3 + 1 \cdot 2^2 &&\Rightarrow 8 + 4 = 12_{\text{d}} \end{aligned}$$

1.12.3 Hexadezimalzahl

$$\begin{aligned} \text{0x123} &\Rightarrow 1 \cdot 16^2 + 2 \cdot 16^1 + 3 \cdot 16^0 = 352_{10} \\ \text{0x2B3} &\Rightarrow 2 \cdot 16^2 + 11 \cdot 16^1 + 3 \cdot 16^0 = 691_{10} \end{aligned}$$

1.13 Codierung und Decodierung von UTF-8

1.13.1 Codierung in UTF-8

UTF-8 ist eine Codierungsmethode, die es ermöglicht, alle Unicode-Zeichen durch 1 bis 4 Bytes zu repräsentieren. Für ASCII-Zeichen werden nur 1 Byte benötigt, während für Zeichen mit höheren Codepunkten mehr Bytes verwendet werden.

Beispiel für die Codierung:

- Das Zeichen C (U+0043) wird als `0100 0011` codiert (1 Byte).
- Das Zeichen é (U+00E9) wird als `1100 0011 1010 1001` codiert (2 Bytes).
- Das Emoji “ (U+1F60A) wird als `1111 0000 1001 1111 1001 1000 1010 1000` codiert (4 Bytes).

1.13.2 Decodierung von UTF-8

Die Decodierung von UTF-8 basiert darauf, die Anzahl der führenden Einsen im ersten Byte zu lesen, um die Länge des gesamten Zeichens zu bestimmen:

Erklärung der Byte-Strukturen in UTF-8:

- Ein 1-Byte Zeichen beginnt mit `0xxx xxxx`, steht für ein ASCII-Zeichen. Dieses Byte ist das gesamte Zeichen.
- Ein 2-Byte Zeichen beginnt mit `110x xxxx`, gefolgt von einem Fortsetzungsbyte, das mit `10xx xxxx` beginnt.
- Ein 3-Byte Zeichen beginnt mit `1110 xxxx`, gefolgt von zwei Fortsetzungsbytes, die mit `10xx xxxx` beginnen.
- Ein 4-Byte Zeichen beginnt mit `1111 0xxx`, gefolgt von drei Fortsetzungsbytes, die mit `10xx xxxx` beginnen.

Beispiel für die Decodierung:

- `0100 0011` (1 Byte) wird als ASCII-Zeichen H decodiert.
- `1100 0011 1010 1001` (2 Bytes) wird als é decodiert.
- `1111 0000 1001 1111 1001 1000 1010 1000` (4 Bytes) wird als Emoji “ decodiert.

Prozess der Decodierung:

1. Erkenne die Anzahl der führenden Einsen im ersten Byte. Dies gibt die Byte-Anzahl des gesamten Zeichens an.
2. Kombiniere die Bits der folgenden Bytes entsprechend, um den Unicode-Codepunkt zu rekonstruieren.
3. Übersetze den Codepunkt zurück in das entsprechende Zeichen.

2 Schaltalgebra

2.1 Gesetze und Rechenregeln

Bedeutung	Beispiel	Beispiel'
Kommutativgesetze	a ∨ b = b ∨ a	a ∧ b = b ∧ a
Assoziativgesetze	(a ∨ b) ∨ c = a ∨ (b ∨ c)	(a ∧ b) ∧ c = a ∧ (b ∧ c)
Absorptionsgesetze	a ∨ (a ∧ b) = a	a ∧ (a ∨ b) = a
Distributivgesetze	a ∨ (b ∧ c) = (a ∨ b) ∧ (a ∨ c)	a ∧ (b ∨ c) = (a ∧ b) ∨ (a ∧ c)
Komplementärgesetze	a ∨ ¬a = 1	a ∧ ¬a = 0
Neutralitätsgesetze	a ∨ 0 = a	a ∧ 1 = a
Extremalgesetze	a ∨ 1 = 1	a ∧ 0 = 0
Dualitätsgesetze	¬1 = 0	¬0 = 1
Idempotenzgesetze	a ∨ a = a	a ∧ a = a
Involutionsgesetz	¬(¬a) = a	
De Morgansche Gesetze	¬(a ∨ b) = ¬a ∧ ¬b	¬(a ∧ b) = ¬a ∨ ¬b

2.2 NAND / NOR Konvertierung

$$\begin{aligned}
\text{NAND}(A,B) &= \neg(A \wedge B) \\
&= \overline{A \wedge B} \\
&= A \uparrow B \\
&= A|B \\
\text{NOR}(A,B) &= \neg(A \vee B) \\
&= \overline{A \vee B} \\
&= A \downarrow B
\end{aligned}$$

2.3 Konjunktive und Disjunktive Normalform

Konjunktive Normalform (KNF):
Eine logische Formel ist in KNF, wenn sie als Konjunktion von Klauseln ist, wobei jede Klausel eine Disjunktion von Literalen ist.

Beispiel:
$(x \vee \neg y) \wedge (\neg x \vee z)$.

Disjunktive Normalform (DNF / KDNF):
Eine logische Formel ist in DNF, wenn sie als Disjunktion von Minternen ist, wobei jedes Mintern eine Konjunktion von Literalen ist.

Beispiel:
$(x \wedge y) \vee (\neg x \wedge z)$.

Umformung:
Von KNF zu DNF (z. B. $(x \vee \neg y) \wedge (\neg x \vee z)$):

- Wende die Distributivgesetze an: $(x \vee \neg y) \wedge (\neg x \vee z) = (x \wedge \neg x \vee x \wedge z) \vee (\neg y \wedge \neg x \vee \neg y \wedge z)$.
- Vereinfache: $(x \wedge z) \vee (\neg y \wedge \neg x) \vee (\neg y \wedge z)$, dann weiter zu $(x \wedge z) \vee (\neg x \wedge \neg y)$.

Von DNF zu KNF (z. B. $(x \wedge y) \vee (\neg x \wedge z)$):

- Wende die Distributivgesetze an: $(x \wedge y) \vee (\neg x \wedge z) = (x \vee \neg x) \wedge (x \vee z) \wedge (y \vee \neg x) \wedge (y \vee z)$.
- Vereinfache: $(x \vee z) \wedge (y \vee \neg x)$.

2.4 Gatter aus NAND / NOR

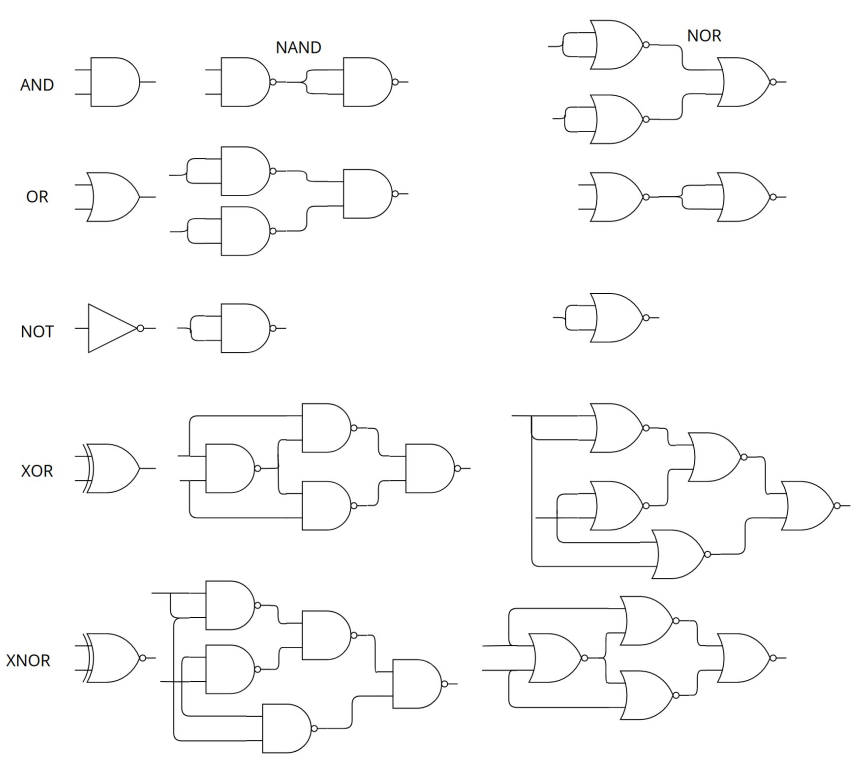
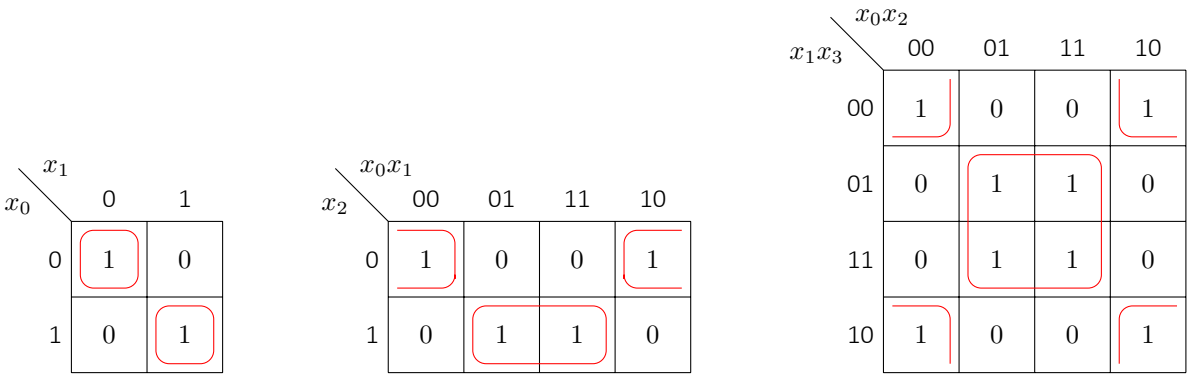
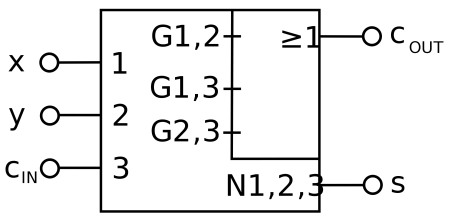


Abbildung 1: Gatter aus NAND und NOR

2.5 Karnaugh-Veitch-Diagramm



2.6 Paralleladdierer



x, y	Inputs
c_{IN}	Carry
s	Summe
c_{out}	Carry-Out

Abbildung 2: Symbol eines Volladdierer

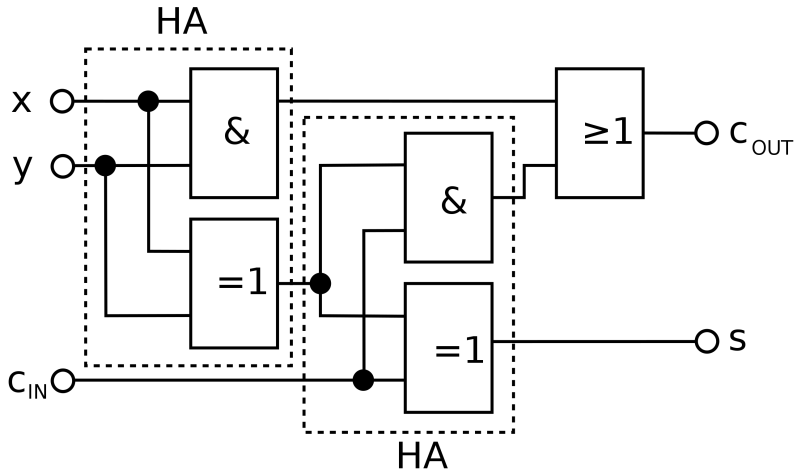


Abbildung 3: Aufbau eines Volladdierer

- Die Durchlaufzeit ist die Schaltzeit · Anzahl der Bits (Anzahl der Addierer)
- Um mehrstellige binäre Zahlen zu addieren, kann man Volladdierer in Serie schalten. Dabei entsteht ein "Parrallel-addierer"
- Der Summen-Output s ist lediglich das Ergebnis des jeweiligen Volladdierer (VA) und entspricht nicht der totalen Summe
- Der c_{out} muss auf den c_{IN} des nächsten VA verbunden werden, um einen Übertrag zu zeigen. z.B. $8 + 7 = 15$, also >9 $\rightarrow$ Übertrag
- Resultat: Das Resultat besteht aus der Summe aller s Outputs + dem c_{OUT} des höchsten (MSB) VA
- Nachteile: Die Geschwindigkeit ist begrenzt und durch die Verzögerung der Carry-Propagation können zeitkritische Applikationen Probleme haben.

2.6.1 Beispiel Addition

Addition von 0111 + 1011 A = x, B = y, c_{IN} = Ü, s (Summe) = $\sum$, c_{OUT} = C

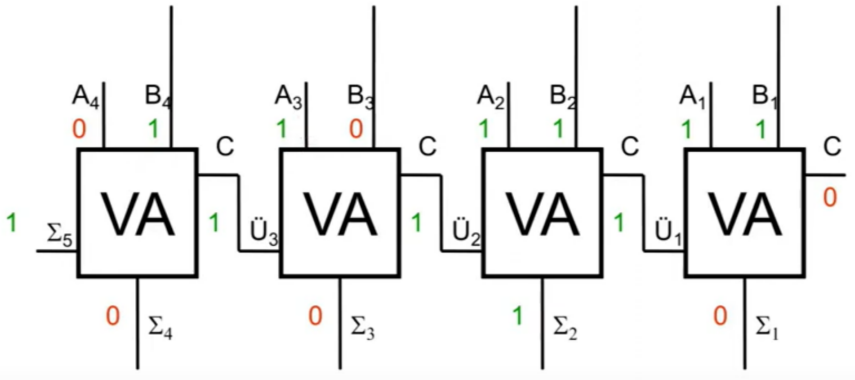


Abbildung 4: Beispiel Berechnung Volladdierer

Die Bits werden Paarweise basierend auf ihrem Stellenwert 2^x auf die Addierer zugeteilt. Für die Addition: 0111 + 1011 entstehen folgende Paare: (0,1), (1,0), (1,1), (1,1)

3 Polynomdivision

3.1 Rechnen mit Polynomen

3.1.1 Addition

$$\begin{aligned} 101 + 110 &= (2^2 + 2^0) + (2^2 + 2^1) \\ &= 2^2 + 2^2 + 2^1 + 2^0 \\ &= 2 \cdot 2^2 + 2^1 + 2^0 \\ &= 8 + 2 + 1 = 11 \end{aligned}$$

$A \oplus B$ in $\mathbb{Z}_2$

$$\begin{aligned} A(x) \oplus B(x) &= (x^4 + x^2 + x + 1) \oplus (x^3 + x + 1) \\ &= x^4 + x^3 + x^2 + 2x + 2 \\ &\quad \text{mod } 2 \\ &= \underline{\underline{1'1100}} \end{aligned}$$

3.1.2 Subtraktion

$$\begin{aligned} 110 - 011 &= (2^2 + 2^1) - (2^1 + 2^0) \\ &= 2^2 + 2^1 - 2^1 - 2^0 \\ &= 2^2 - 2^0 \\ &= 4 - 1 = 3 \end{aligned}$$

3.1.3 Multiplikation

$$\begin{aligned} 101 \times 111 &= (2^2 + 2^0) \times (2^2 + 2^1 + 2^0) \\ &= 2^2 \times (2^2 + 2^1 + 2^0) + 2^0 \times (2^2 + 2^1 + 2^0) \\ &= (2^4 + 2^3 + 2^2) + (2^2 + 2^1 + 2^0) \\ &= 2^4 + 2^3 + 2 \cdot 2^2 + 2^1 + 2^0 \\ &= 16 + 8 + 8 + 2 + 1 = 35 \end{aligned}$$

$A \otimes B$ in $\mathbb{Z}_2$

$$\begin{aligned} A(x) \otimes B(x) &= (x^4 + x^2 + x^1 + x^0) \cdot (x^3 + x^1 + x^0) \\ &= x^7 + x^5 + x^4 + x^5 + x^3 + x^2 + x^4 + x^2 + x^1 + x^3 + x^1 + x^0 \\ &= x^7 + 2x^5 + 2x^4 + 2x^3 + 2x^2 + 2x^1 + x^0 \\ &\quad \text{mod } 2 \\ &= \underline{\underline{1000'0001}} \end{aligned}$$

3.2 Hexadezimalzahl als Polynom

$$\begin{aligned} 2B3_h \quad &\Rightarrow \quad 3 \cdot 16^0 + 11 \cdot 16^1 + 2 \cdot 16^2 \\ &\Rightarrow \quad 3 + 176 + 512 = 691 \end{aligned}$$

3.3 Reduzible Polynome

Ein Polynom bezeichnet man als reduzibel, wenn es als Produkt zweier Polynome mit jeweils niedrigeren Grad dargestellt werden kann:

$$\text{z.B.} \quad x^2 + x \quad \Rightarrow \quad x(x+1)$$

Die Polynomdivision darf keinen Rest haben. Folgendes Beispiel würde nicht funktionieren:

$$\text{z.B.} \quad x^2 + x + 1 \qquad \rightarrow \text{Es gibt einen Rest, somit ist es kein reduzibles Polynom}$$

3.4 Polynomdivision

Bei Polynomdivisionen gehen wir nach folgenden Prinzip vor:

1 dividieren

2 multiplizieren

3 subtrahieren

Beispiel ohne Rest:

$$\begin{array}{l}
\big(\quad x^3 - 6x^2 + 9x - 4\big) \div \big(x - 1\big) = x^2 - 5x + 4 \\
\underline{-x^3 \;\; + x^2} \\
\qquad -5x^2 + 9x \\
\qquad \underline{5x^2 - 5x} \\
\qquad\qquad 4x - 4 \\
\qquad\qquad \underline{-4x + 4} \\
\qquad\qquad\quad 0
\end{array}$$

Beispiel mit Rest:

$$\begin{array}{l}
\big(\quad x^6\big) \div \big(x^3 + x + 1\big) = x^3 - x - 1 + \dfrac{x^2 + 2x + 1}{x^3 + x + 1} \\
\underline{-x^6 - x^4 - x^3} \\
\qquad -x^4 - x^3 \\
\qquad \underline{x^4 \qquad + x^2 \;\; + x} \\
\qquad\quad -x^3 + x^2 \;\; + x \\
\qquad\quad \underline{x^3 \qquad\quad + x + 1} \\
\qquad\qquad\quad x^2 + 2x + 1
\end{array}$$

Das Resultat dieser Division ist: $x^3 - x - 1 + \dfrac{x^2 + 2x + 1}{x^3 + x + 1}$ oder mit Rest geschrieben: $x^3 - x - 1$, Rest: $x^2 + 2x + 1$

3.5 Polynomdivision zurückrechnen

Das Resultat einer Polynomdivision ist: $(x^4 + 1)$, Rest$(x + 1)$

Berechnet sich die ursprüngliche Division:

$$\begin{aligned}
x^4 + 1, \text{ Rest: } x + 1 &= (x^4 + 1) + \frac{x + 1}{x^4 + 1} \\
&= \frac{(x^4 + 1)^2 + x + 1}{x^4 + 1} \\
&= \frac{x^8 + 2x^4 + x + 2}{x^4 + 1}
\end{aligned}$$

4 Wahrscheinlichkeit / Zufallsvorgänge

4.1 Wahrscheinlichkeiten

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

Beispiel: Ziehung der Lottozahlen, Roulette, Münzwurf . . .

4.2 Ergebnismenge

- Die Menge aller möglichen Ausgänge/Ergebnisse eines Zufallsvorgangs heisst Ergebnismenge und wird mit Ω bezeichnet.
- Ein einzelnes Element $\omega \in \Omega$ heisst Ergebnis.
- Die Anzahl aller Ergebnisse wird $|\Omega|$ notiert.
  - **Beispiel**: Zufallsvorgang: "Werfen einer Münze": $\Omega = \{1, 2, 3, 4, 5, 6\}$

4.3 Auftrittswahrscheinlichkeit / Laplacesche Wahrscheinlichkeitsdefinition

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

Beispiel:
Bei der wiederholten Durchführung eines Experimentes treten die Ergebnisse A1 bis A4 (exklusiv) mit den unten angegebenen Häufigkeiten auf.

Ereignis	A1	A2	A3	A4
Anzahl	5550	3567	2234	4443

Tabelle 3: Beispiel Auftrittswahrscheinlichkeit

Die Wahrscheinlichkeit, dass A1 auftritt, ist:

$$P(A1) = \frac{5500}{15744\text{(Summe aller Ergebnisses)}} = 0.353 = 35.3\%$$

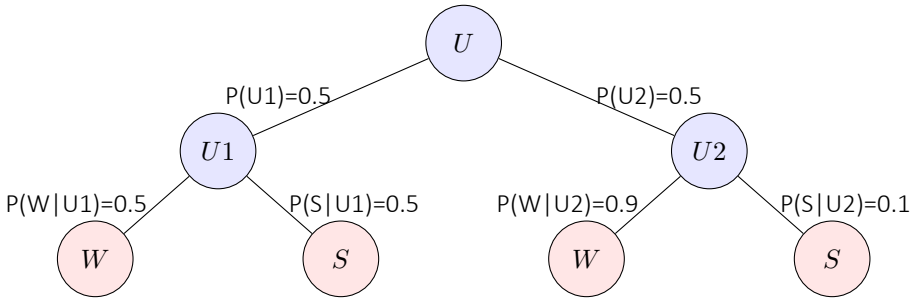
Die Wahrscheinlichkeit, dass A2 oder A4 auftritt, ist:

$$P(A2 \vee A4) = P(A2) + P(A4) = \frac{3567 + 4443}{15744} = 0.509 = 50.9\%$$

Die Wahrscheinlichkeit, dass zuerst A1 und dann A3 auftritt:

$$P(A1 \wedge A3) = P(A1) \cdot P(A3) = 0.05 = 5\%$$

Weiteres **Beispiel**:
Es gibt zwei Urnen, U1 und U2. Urne U1 enthält 5 schwarze und 5 weisse Kugeln und Urne U2 enthält 9 schwarze und 1 weisse Kugel. Die Wahrscheinlichkeit eine Kugel aus U1 oder U2 zu ziehen, ist gleich gross (50/50). Wie gross ist nun die Wahrscheinlichkeit eine weisse Kugel zu ziehen?



$$\begin{aligned} P(\mathsf{W}|U1) \wedge P(U1) \vee P(\mathsf{W}|U2) \wedge P(U2) &= P(\mathsf{W} \mid \mathsf{U1}) \cdot P(U1) + P(\mathsf{W} \mid \mathsf{U2}) \cdot P(U2) \\ &= 0.5 \cdot 0.5 + 0.5 \cdot 0.1 \\ &= 0.25 + 0.05 \\ &= \underline{\underline{0.3}} \end{aligned}$$

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

4.4 Kombinatorik

4.4.1 Permutation

$$n!$$

Die Anzahl der Möglichkeiten, n Elemente in einer bestimmten Reihenfolge anzuordnen

4.4.2 Geordnete Proben mit Wiederholung

$$n^k$$

n Die Anzahl der Möglichkeiten
k Anzahl der Durchführungen

Beispiel:

Ein Würfel wird 3-mal gewürfelt. Dabei entstehen $6^3 = 216$ Möglichkeiten (5, 5, 6), (2, 3, 2), ...

4.4.3 Geordnete Proben ohne Wiederholung

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

n Die Anzahl der Möglichkeiten
k Anzahl der Elemente $\in$ n (Jedes Element kann nur einmal verwendet werden)

Beispiel:

Gesucht ist die Anzahl der unterschiedlichen Elemente, die aus einem Würfel (1-6) gebildet werden können. z.B. (1, 2), (1, 3), (2, 3), ...

$$\frac{6!}{(6-2)!} = \frac{720}{24} = 30$$

4.4.4 Ungeordnete Proben ohne Wiederholung und ohne Beachtung der Reihenfolge

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

n Die Anzahl der Möglichkeiten
k Anzahl der Elemente $\in$ n

Die Formel gibt die Anzahl Möglichkeiten an, k Elemente aus einer Menge n Elementen, wobei die Reihenfolge keine Rolle spielt und keine Wiederholungen erlaubt sind.

Beispiel: Wie viele Möglichkeiten gibt es, um 6 zufällige Zahlen aus 49 Zahlen zu ziehen? (Lotto 6 aus 49)

$\rightarrow$ Alle möglichen Kombinationen mit 6 Zahlen: $\binom{49}{6} = \frac{49!}{6! \cdot (49-6)!} \approx 14 \cdot 10^6$

4.5 Hypergeometrische Verteilung

Die hypergeometrische Verteilung ist ein Modell, das verwendet wird, um die Wahrscheinlichkeit von k Erfolgen (gewünschten Ergebnissen) in n Ziehungen zu berechnen.

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

$P(k)$	Die Wahrscheinlichkeit, k Erfolge zu erzielen
K	Anzahl der möglichen Erfolge (z.B. Anzahl der Gewinnzahlen im Lotto)
k	Anzahl der Erfolge, die gezogen werden (z.B. Anzahl der richtig getippten Zahlen)
N	Die Gesamtzahl der Objekte (z.B. Gesamtanzahl der Lottozahlen)
n	Anzahl der Ziehungen (z.B. Werden 6 Kugeln beim Lotto gezogen)

Siehe 11.4 TI-Nspire, für eintippen der Formel beim TI-Nspire.

4.6 Bitfehler

Ein Bitfehler bezeichnet die inkorrekte Wiedergabe eines Bits während der Datenübertragung oder -speicherung. Das bedeutet, dass ein Bit von 0 zu 1 oder von 1 zu 0 fälschlicherweise geändert wird. Die Wahrscheinlichkeit eines solchen Fehlers wird als Bitfehlerrate (BER) ausgedrückt und ist ein Mass dafür, wie oft solche Fehler, im Verhältnis zur Gesamtzahl der übertragenen Bits, auftreten.

Die Wahrscheinlichkeit, dass ein Datenblock der Grösse n Bits fehlerhaft ist, kann mit folgender Formel berechnet werden:

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

BER	Bitfehlerrate
n	Die Grösse des Datenblocks in Bits

Beispiel

Gegeben sei eine Bitfehlerrate von 10^{-3}. Wie gross ist die Wahrscheinlichkeit, dass ein Datenblock mit einer Grösse von 150 kbit fehlerhaft ist?

Unter Verwendung der obigen Formel:

$$P(\text{fehlerhaft}) = 1 - (1 - 10^{-3})^{150 \cdot 10^3} = 1$$

4.6.1 Berechnung der Fehlerwahrscheinlichkeit

Wahrscheinlichkeit für genau k Fehler in einem Datenblock mit n Bits bei Bitfehlerrate BER:

Binomialverteilung:

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

4.7 Binomialverteilung

Die Binomialverteilung bestimmt, wie wahrscheinlich es ist, eine bestimmte Anzahl von Erfolgen zu erzielen, wenn wir etwas mehrmals versuchen und jeder Versuch kann entweder gelingen oder misslingen.

Die Wahrscheinlichkeitsfunktion der Binomialverteilung ist gegeben durch:

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

$P(k; n, p)$	Die Wahrscheinlichkeit, genau k Erfolge in n Versuchen zu erzielen
n	Anzahl der Versuche oder Wiederholungen
k	Anzahl der erzielten Erfolge
p	Wahrscheinlichkeit eines Erfolgs bei jedem Versuch

Beispiel "Fehlerkorrekturverfahren":
Mit einer Vorwärtskorrektur kann man bis zu 3 Fehler richtig korrigieren. Die Grösse eines Datenblocks ist 150 bit. Wie gross ist die Wahrscheinlichkeit, dass höchstens 3 Fehler auftreten?:

$$\begin{aligned} P(\leq 3 \text{ Fehler}) &= P(0 \text{ Fehler}) + P(1 \text{ Fehler}) + P(2 \text{ Fehler}) + P(3 \text{ Fehler}) \\ &= \binom{150}{0} \cdot 0.01^0 \cdot 0.99^{150} + \binom{150}{1} \cdot 0.01^1 \cdot 0.99^{149} + \binom{150}{2} \cdot 0.01^2 \cdot 0.99^{148} + \binom{150}{3} \cdot 0.01^3 \cdot 0.99^{147} \\ &= 0.221 + 0.336 + 0.252 + 0.126 \\ &= 0.935 \end{aligned}$$

4.7.1 Restfehlerwahrscheinlichkeit

Die Restfehlerwahrscheinlichkeit gibt an, wie gross die Wahrscheinlichkeit ist, dass nach einem Fehlerkorrekturverfahren noch Fehler im Datenblock vorhanden sind. Wenn ein Verfahren bis zu einer bestimmten Anzahl von Fehlern korrigieren kann, ist die Restfehlerwahrscheinlichkeit die Wahrscheinlichkeit, dass die Anzahl der Fehler diese Grenze überschreitet.

Beispiel:
Ein Fehlerkorrekturverfahren ist in der Lage, bis zu 3 Bitfehler in einem Datenblock von 150 Bit zu korrigieren.

$$P(\text{Restfehler}) = 1\text{–}P(\leq 3 \text{ Fehler}) = 1\text{–}0.935 = 0.065 = 6.5\%$$

Diese Wahrscheinlichkeit zeigt, dass in etwa 6,5% der Fälle mehr als 3 Fehler in einem 150-Bit-Datenblock auftreten werden, selbst wenn ein Fehlerkorrekturverfahren angewendet wird.

2. Beispiel:
Man bestimme die Wahrscheinlichkeit dafür, dass bei zwei Würfen eines unverfälschten Würfels wenigstens einmal die 4 erscheint.

Die Wahrscheinlichkeit, dass keine 4 gewürfelt wird, liegt bei $\frac{5}{6}$. Daraus ergibt sich die folgende Gleichung:

$$P(\text{mind. einmal 4}) = 1 - \frac{5}{6} \cdot \frac{5}{6} = 1 - \frac{25}{36} = \frac{11}{36} = 0.306$$

5 Quellencodierung und Komprimierung

5.1 Huffman-Codierung

5.1.1 Redundanz der Quelle

$$
\begin{aligned}
I(x_i) &= -\log_2(P(x_i)) \\
H(X) &= \sum_{i=1}^{n} P(x_i) \cdot -\log_2(P(x_i)) = \sum_{i=1}^{n} P(x_i) \cdot I(x_i) \\
H_0 &= \log_2(n) \\
R_Q &= H_0 - H(X) \\
L &= \sum_{i=1}^{n} P(x_i) \cdot L(x_i)
\end{aligned}
$$

x_i	Das i-te Ereignis
n	Gesamtzahl der Zeichen
$I(x_i)$	Informationsmenge (in Bit)
$P(x_i)$	Auftrittswahrscheinlichkeit
$H(X)$	Erwarteter Informationsgehalt (in Bit) der Quelle
H_O	Grösster möglicher Wert von Informationsmengen, wenn alle Möglichkeiten gleich häufig sind
R_Q	Redundanz der Quelle (in Bit)

Beispiel: Berechne die Redundanz der Quelle
Gegeben sind die Wahrscheinlichkeiten der Zeichen A, B, C, D, und E:

- $P(A) = 0.5$
- $P(B) = 0.25$
- $P(C) = 0.1$
- $P(D) = 0.1$
- $P(E) = 0.05$

Die Berechnung von $H(X)$ ist wie folgt:

$$
\begin{aligned}
H(X) &= (-0.5 \cdot log_2(0.5)) + (-0.25 \cdot log_2(0.25)) + (-0.1 \cdot log_2(0.1)) + (-0.1 \cdot log_2(0.1)) + (-0.05 \cdot log_2(0.05)) \\
&= 1.88048 \text{ bit}
\end{aligned}
$$

Da wir 5 Zeichen haben, ist $n = 5$ und somit ist H_0 berechnet durch:

$$H_0 = \log_2(5) = 2.32193$$

Die Redundanz R_Q ergibt sich zu:

$$R_Q = H_0 - H(X) = 2.32193 - 1.88048 = 0.442 \text{ bit}$$

5.1.2 Redundanz der Quelle minimieren (Übertragung eines binären Codes nach Huffman)

$$L = \sum_{i=1}^{n} P(x_i) \cdot L(x_i)$$
$$R = L - H(X)$$
$$R_Q = \lceil \log_2(n) \rceil - H(X)$$
$$\text{Verbesserung (in \%)} = \frac{R_Q - R}{R_Q} \cdot 100$$

x_i	Das i-te Ereignis
n	Gesamtzahl der Zeichen
$P(x_i)$	Auftrittswahrscheinlichkeit
L	(mittlere) Codewortlänge (in bit)
R_Q	Redunanz der nicht komprimierten Quelle (in bit)
R	Verbesserte Redundanz (in bit)
Verbesserung	Prozentuale Verbesserung der Redundanz durch Huffman-Codierung

Um die Codewortlänge zu bestimmen, muss man einen Baum zeichnen, dabei gilt folgendes Vorgehen:

1 Schreibe die Zeichen mit abnehmender Wahrscheinlichkeit auf (Element mit höchster Wahrscheinlichkeit zuoberst)

2 Wähle die zwei Zeichen mit der geringsten Wahrscheinlichkeit und erstelle einen Knoten

3 Wiederhole diesen Schritt, bis alle Zeichen in dem Baum verbunden sind

4 Weise jedem linken Ast den Wert «0» und jedem rechten Ast den Wert «1» zu

5 Verfolge den Pfad vom Element zum Ziel, notiere die Sequenz von 0 und 1

Beispiel: Um wie viel Prozent kann die Redundanz des Beispiels zuvor minimiert werden

Berechne die (mittlere) Codewortlänge L wie folgt:

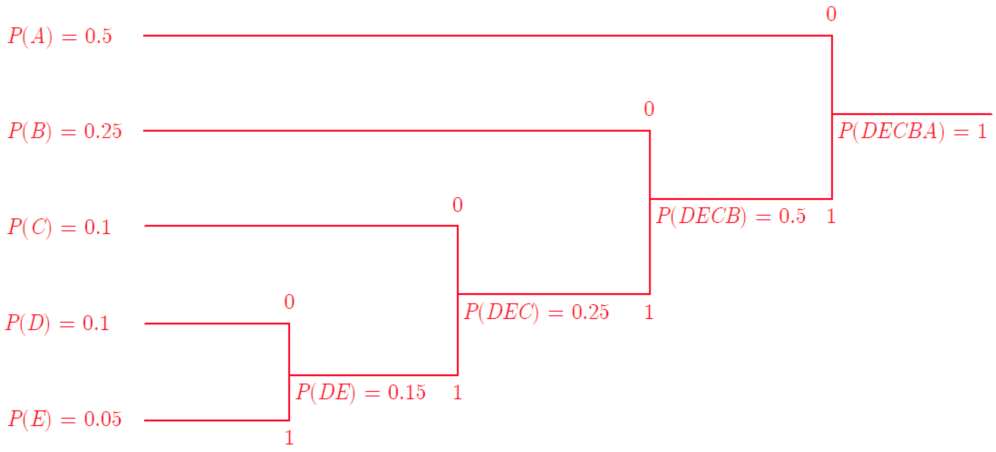


Abbildung 5: Huffman tree

Daraus resultieren die folgenden Binärcodes:

$P(A) = 0 = 1$ 1 = Anzahl der Bits, 1110 wäre also eine Länge von 4 bits
$P(B) = 10 = 2$
$P(C) = 110 = 3$
$P(D) = 1110 = 4$
$P(E) = 1111 = 4$

Somit haben wir: $L_{x_A} = 0, \ldots, L_{x_E} = 4$

Nun ergibt sich die Länge L

$$L = 0.5 \cdot 1 + 0.25 \cdot 2 + 0.1 \cdot 3 + 0.1 \cdot 4 + 0.05 \cdot 4 = 1.9 \text{ bit}$$

Nun müssen wir R_Q bestimmen mit dem aus 5.1.1 berechneten $H(X)$. Für L verwenden wir 3 Bits, da dies die nicht komprimierte Codewortlänge ist. Dies kann bestummen werden, indem wir den zuvor berechnet Wert H_0 aufrunden (2.23 -> 3) $H(X)$

$$R_Q = L_{\text{not compressed}} - H(X) = 3 - 1.88 = 1.12 \text{ bit}$$
$$R = L - H(X) = 1.9 - 1.88 = 0.02 \text{ bit}$$

Das entspricht einer Verbesserung von:

$$\frac{R_Q - R}{R_Q} \cdot 100 = \frac{1.12 - 0.02}{1.12} \cdot 100 = 98.21\%$$

5.2 Lauflängencodierung

5.2.1 Definition und Anwendung

Die Lauflängencodierung ist ein einfaches verlustfreies Datenkompressionsverfahren, das besonders bei Daten mit vielen gleichen, aufeinanderfolgenden Zeichen (wie z.B. in Bildern) effektiv ist. Sie ersetzt Sequenzen von gleichen Datenwerten durch das Auftreten des Wertes und die Anzahl seiner Wiederholungen.

5.2.2 Prozess der Lauflängencodierung

$$C(s) = \sum (l_i \cdot s_i) \quad \text{für alle } i \text{ mit } 1 \leq i \leq n$$
$$l_i = \text{Anzahl der aufeinanderfolgenden Wiederholungen von } s_i$$
$$s_i = \text{Das Symbol, das wiederholt wird}$$
$$n = \text{Anzahl der unterschiedlichen Sequenzen in der Eingabe}$$

Beispiel: DNA-Sequenz Codierung
Um die Lauflängencodierung auf eine DNA-Sequenz anzuwenden, folgen Sie diesen Schritten:

1. Zähle die Anzahl der aufeinanderfolgenden Wiederholungen jedes Nukleotids.
2. Codiere jede Gruppe von Wiederholungen durch die Anzahl gefolgt von dem Nukleotid.

Gegeben sei die DNA-Sequenz:

aaaaagggccctttttaaaaag

Die lauflängencodierte Sequenz wäre:

5a3g3c5t5ag

5.2.3 Berechnung der Kompressionsrate

Die Kompressionsrate R wird berechnet durch:

$$R(\%) = \frac{L_{codiert}}{L_{original}} \cdot 100$$
$$L_{original} = \text{Länge der ursprünglichen Sequenz}$$
$$L_{codiert} = \text{Länge der codierten Sequenz}$$

Beispiel:
Angenommen, die ursprüngliche DNA-Sequenz ist 50 Zeichen lang und die lauflängencodierte Sequenz ist 30 Zeichen lang, dann wäre die Kompressionsrate:

$$R = \frac{30}{50} \cdot 100 = 60\%$$

Dies bedeutet, dass die ursprüngliche Sequenz um 60% komprimiert wurde.

5.3 Lempel-Ziv-Welch (LZW)-Komprimierung

5.3.1 Prinzip der LZW-Komprimierung

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

5.3.2 Funktionsweise der LZW-Komprimierung

1 Beginne mit einem Wörterbuch, das bereits alle möglichen Einzelzeichen enthält.

2 Suche im Wörterbuch die längste Sequenz, die mit den nächsten Zeichen (n + 1) der Eingabe übereinstimmt.

3 Speichere den Index des gefundenen Wörterbucheintrags.

4 Erstelle einen neuen Eintrag im Wörterbuch mit der gefundenen Sequenz, gefolgt vom nächsten Zeichen der Eingabe.

5 Verschiebe das Eingabefenster um die Anzahl der codierten Zeichen.

6 Wiederhole den Prozess, bis alle Zeichen codiert sind.

Beispiel: Komprimierung einer Ziffernfolge (LZW-Komprimierung)
Gegeben Gegeben sei eine einfache LWZ-Komprimierung, welche nur Ziffern 0-9 komprimiert. Die LZW Komprimierung startet mit einem Wörterbuch, welches bereits für Ziffern 0 - 9 einen Eintrag hat. Des Weiteren ist die folgende Ziffernfolge zu kodieren: «123123123123».

Die bereits gegebenen/vorhandenen Einträge:

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

Tabelle 4: Gegebenes Wörterbuch der LZW-Komprimierung

1 Da 0-9 bereits einen Eintrag haben, beginnt der Index im Wörterbuch bei 10:_

2 Das erste Muster ist 12, da 12 noch nicht im Wörterbuch steht, wird 12 mit dem nächst freien Index (10) im Wörterbuch hinzugefügt

3 Nun wird 23 überprüft, da diese Kombination auch noch nicht im Wörterbuch vorhanden ist, wird es ebenfalls hinzugefügt

4 Das selbe mit 31 . . .

5 Nun kommt wieder die Zahlenfolge 12, welche bereits einen Eintrag hat. Jetzt wird dieser Eintrag 12 mit der nächsten Zahl verglichen, also 123

6 Die nächste Zahlenfolge ist 312, da 3 der letzte Index ist, der zuvor überprüften Zahlenfolge

7 Das selbe mit 231 . . .

8 Jetzt kommt wieder 123, welches aber bereits einen Eintrag hat und somit geskipt wird

Daraus ergibt sich das nachfolgende Tabelle:

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

Tabelle 5: LZW-Komprimierungsschritte

Daraus resultiert die folgende codierte Nachricht (Indexe der erkannten Zeichenfolgen):

1 2 3 10 12 11 13

. . . und das folgende Wörterbuch:

Index	Eintrag
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
9	9
10	12
11	23
12	31
13	123
14	312
15	231

Tabelle 6: Wörterbuch der LZW-Komprimierung nach Anwendung der gegebenen Ziffernfolge

5.3.3 Berechnung der Kompressionsrate

Die Kompressionsrate wird berechnet durch:

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

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

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

Beispiel
Möchten wir die Kompressionsrate der Aufgabe aus dem Beispiel 5.3.2 berechnen, sehen wir, dass:

- Der grösste Index < 16 ist
- Die grösste Eingangsziffer 3 und somit < 16 ist

Das heisst, dass wir das ganze mit 4 Bits realisieren können.

Die Ziffernfolge «123123123123» hat 12 Zeichen, somit eine Bitlänge von: 12 * 4 Bits = 48 Bits

$$L_{\text{original}} = \text{Ziffernfolge} \cdot \text{Bitgrösse}$$

Die komprimierte Länge entspricht der Länge der codierten Nachricht · der Bitlänge: 7 * 4 Bits = 28 Bits

$$L_{\text{komprimiert}} = \text{codierte Nachricht} \cdot \text{Bitgrösse}$$

Das entspricht einer Kompressionsrate R von:

$$R(\%) = \frac{L_{komprimiert}}{L_{original}} \cdot 100 = \frac{28}{48} \cdot 100 = 58.33\%$$

5.3.4 Dekomprimierung mit LZW

Zur Dekomprimierung einer LZW-codierten Nachricht wird das gleiche Wörterbuch verwendet, das bei der Komprimierung erstellt wurde. Das Wörterbuch wird durchlaufen, um die entsprechenden Sequenzen zu ersetzen und die ursprüngliche Nachricht wiederherzustellen.

1 Lies den nächsten Code aus der komprimierten Nachricht.

2 Finde den entsprechenden Eintrag im Wörterbuch und dekodiere ihn.

3 Füge den dekodierten Text zur Nachricht hinzu und aktualisiere das Wörterbuch, wenn nötig.

4 Wiederhole die Schritte, bis die gesamte Nachricht dekomprimiert ist.

Beispiel:
Möchten wir die codierte Nachricht «7 5 10 11 12» dekomprimieren, wissen wir, dass die ersten Einträge des Wörterbuchs die Grundzeichen des Eingabestroms sind, in diesem Fall 0-9.

Nun schreiben wir die codierte Nachricht in die Tabelle und decodieren sie über die Indexe:

1 Der erste Eintrag ist 7, 7 hat den Index 7

2 . . . 5 gibt 5 zurück

3 Die Zahl 10 muss ”reverse-engineered“ werden. Da 7 und 5 die ersten beiden Zahlen sind, die man einliest, ist das erste Wort, das man dem Wörterbuch hinzufügt «75». Somit hat der Index 10 den Wert 75

4 Da wir nun das Muster «7575» haben, sind die nächsten beiden Zahlen 57, somit hat der Index 11 den Wert 57

5 Die nächste Zahlenfolge ist wieder 75, da wir 75 bereits im Wörterbuch haben, kommt die nächste Zahl hinzu. Somit Index 12 = 755

Nachricht	Decodiert	Eintrag
7 5 10 11 12	7	
7 5 10 11 12	75	→ 10: 75
7 5 10 11 12	7575	→ 11: 57
7 5 10 11 12	7575 7	→ 12: 755
7 5 10 11 12	7575 7 755	→ 13: 577

Tabelle 7: LZW-Decodierungsprozess

. . . somit erhalten wir die Nachricht «757557755» ⇒ (7 5 75 57 755)

5.4 Kombination von Komprimierungen

Kann der LZW-Algorithmus und die Huffman-Codierung sinnvoll kombiniert werden?
Ja. Zuerst werden die Daten LZW-komprimiert, anschliessend das Wörterbuch Huffman-codiert. D.h. die gefundenen Phrasen werden entsprechend ihrer Häufigkeit codiert.

6 Quellencodierung und Verschlüsselung

6.1 Caesar-Chiffre

Die Caesar-Chiffre ist eine der ältesten bekannten Verschlüsselungstechniken. Sie ist ein Typ der Substitutionschiffre, bei der jeder Buchstabe im Klartext durch einen Buchstaben ersetzt wird, der sich um eine festgelegte Anzahl von Stellen weiter im Alphabet befindet.

6.1.1 Verschlüsselung

Um eine Nachricht mit der Caesar-Chiffre zu verschlüsseln, wählt man eine Verschiebungsanzahl n und verschiebt dann jeden Buchstaben der Nachricht um n Stellen im Alphabet. Wenn das Ende des Alphabets erreicht wird, fängt man wieder am Anfang an. Mathematisch kann die Verschlüsselung eines Buchstabens x durch folgende Formel ausgedrückt werden:

$$E_n(x) = (x + n) \mod 26$$

wobei x die Position des Buchstabens im Alphabet ist (beginnend mit 0) und n die Verschiebungsanzahl.

6.1.2 Entschlüsselung

Die Entschlüsselung ist einfach der umgekehrte Prozess der Verschlüsselung. Jeder Buchstabe des Chiffretextes wird um die gleiche Anzahl von Stellen zurück im Alphabet verschoben. Die Formel für die Entschlüsselung eines Buchstabens y lautet:

$$D_n(y) = (y - n) \mod 26$$

⇒ Wichtig dabei: Der y-Wert stammt aus der Originalen Tabelle. Die zu konvertierende Zahl muss immer aus der Originalen Tabelle ausgelesen werden)

Beispiel
Nehmen wir an, die Verschiebungsanzahl n ist 3 und wir möchten die Nachricht "HALLO" verschlüsseln. "H" wird zu "K", "A" zu "D", "L" zu "O", und so weiter. Die verschlüsselte Nachricht lautet somit "KDOOR".

0	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
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

Tabelle 8: Die Stellen des Alphabets in der Caesar-Chiffre (ohne Verschiebung)

6.2 RSA-Verschlüsselung

Die RSA-Verschlüsselung ist ein asymmetrisches kryptographisches Verfahren, das auf der Schwierigkeit beruht, grosse Zahlen in ihre Primfaktoren zu zerlegen. Das Verfahren nutzt ein Schlüsselpaar: einen öffentlichen und einen privaten Schlüssel.

6.2.1 Schlüsselerstellung

Die Erzeugung der Schlüssel erfolgt in mehreren Schritten:

1 Wähle zwei unterschiedliche grosse Primzahlen p und q.

2 Berechne $n = p \cdot q$ und $\phi(n) = (p - 1) \cdot (q - 1)$.

3 Wähle eine ganze Zahl e, die zu $\phi(n)$ teilerfremd ist.

4 Bestimme d so, dass $d \cdot e \equiv 1 \mod \phi(n)$ gilt.

Der öffentliche Schlüssel ist das Paar (n, e), und der private Schlüssel ist das Paar (n, d).

6.2.2 Verschlüsselung

Um eine Nachricht m zu verschlüsseln, verwendet man den öffentlichen Schlüssel und berechnet:

$$c = m^e \mod n$$

Dabei ist c der verschlüsselte Text.

6.2.3 Entschlüsselung

Die Entschlüsselung erfolgt mit dem privaten Schlüssel durch Berechnung von:

$$m = c^d \mod n$$

Das Ergebnis m ist der ursprüngliche Klartext.

6.2.4 Beispiel für eine Schlüsselerzeugung und Verschlüsselung

Gegeben seien die Primzahlen $p = 11$ und $q = 7$.

Schlüsselerzeugung:

$$n = p \cdot q = 11 \cdot 7 = 77$$
$$\phi(n) = (p-1) \cdot (q-1) = 10 \cdot 6 = 60$$

Für den öffentlichen Schlüssel wählen wir $e = 7$, welches teilerfremd zu $\phi(n)$ ist. Der private Schlüssel d wird berechnet als das multiplikative Inverse von e modulo $\phi(n)$. Die rot markierten Zahlen für x und y sind die Initialwerte, das grau markierte Feld ist das Inverse.

Berechnung von d:

Schritt i	a ($\phi(n)$)	b	Quotient q	Rest r	x ($x = y_{i+1}$)	y ($y = x_{i+1} - q \cdot y_{i+1}$)
1	60	7	8	4	2	-17
2	7	4	1	3	-1	2
3	4	3	1	1	1	-1
4	3	1	3	0	0	1

Tabelle 9: Berechnung des privaten Schlüssels d mit dem erweiterten Euklidischen Algorithmus

$$d = -17 \mod 60 = 43$$

Verschlüsselung einer Nachricht $m = 2$
(Um zu beweisen, dass das Verfahren funktioniert):

$$c = m^e \mod n = 2^7 \mod 77 = 128 \mod 77 = 51$$

Entschlüsselung des Chiffrats $c = 51$:

$$m = c^d \mod n = 51^{43} \mod 77$$

Die Entschlüsselung liefert das ursprüngliche m.

6.3 Euklidischer Algorithmus

6.3.1 ggT

Um den grössten gemeinsamen Teiler (ggT) von zwei Zahlen zu berechnen, kann man den Euklidischen Algorithmus verwenden:

Schritt i	a	b	Quotient q	Rest r
1	2307	1099	2	109
2	1099	109	10	9
3	109	9	12	1
4	9	1	9	0

Tabelle 10: Berechnung des ggT mithilfe des Euklidischen Algorithmus

Der ggT von 2037 und 1099 ist die rote Zahl 1, wie in der dritten Zeile der Tabelle hervorgehoben.

6.3.2 kgV

Das kleinste gemeinsame Vielfache (kgV) lässt sich über den ggT wie folgt berechnen:

$$\text{kgV}(a,b) = \frac{a \cdot b}{\text{ggT}(a,b)}$$

Beispiel: Berechnung des kgV von 297 und 63:

$$297 \cdot 63 = \text{ggT}(297,63) \cdot \text{kgV}(297,63)$$

Nach der Bestimmung des ggT(297, 63) = 9, wie oben gezeigt, folgt für das kgV:

$$\text{kgV}(297,63) = \frac{297 \cdot 63}{\text{ggT}(297,63)} = \frac{297 \cdot 63}{9} = 2079$$

6.4 Probleme asymmetrischer Verfahren

- **Performance:** Asymmetrische Algorithmen sind sehr langsam (ca. 10'000x langsamer als symmetrische Algorithmen).
- **Infrastruktur:** Der öffentliche Schlüssel muss über eine vertrauenswürdige Stelle abrufbar sein. Die Sicherstellung der Vertrauenswürdigkeit und die Bestimmung der Herkunft des öffentlichen Schlüssels sind mit grossem Aufwand verbunden.
- **Mehrere Empfänger:** Bei mehreren Empfänger muss die Nachricht mit dem öffentlichen Schlüssel jedes einzelnen Empfängers verschlüsselt werden.
- **Man-In-The-Middle**

7 Informationstheorie

7.1 Diskrete Quelle ohne Gedächtnis

Diskrete Quellen ohne Gedächtnis haben Symbole, die unabhängig von vorherigen Symbolen auftreten. Jedes Symbol x_k aus einem endlichen Set tritt mit einer Wahrscheinlichkeit $p(x_k)$ auf.

7.1.1 Entscheidungsgehalt

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

$$H_0 = \log_2(N)$$

H_0	Entscheidungsgehalt der Quelle
N	Anzahl unterschiedlicher Symbole

Beispiel:
Betrachten wir eine Quelle mit 5 unterschiedlichen Symbolen:

$$H_0 = \log_2(5) \approx 2.32 \text{ bit}$$

7.1.2 Informationsgehalt

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

$$I(x_k) = -\log_2(p(x_k))$$

$I(x_k)$	Informationsgehalt des Symbols x_k in bit
$p(x_k)$	Wahrscheinlichkeit des Auftretens von x_k

Beispiel:
Für die Symbole a, b, c, d, e mit den Wahrscheinlichkeiten 0.3, 0.1, 0.1, 0.2, 0.3 ergibt sich:

Zeichen	p	I
a	0.3	1.74
b	0.1	3.32
c	0.1	3.32
d	0.2	2.32
e	0.3	1.74

7.1.3 Entropie

Die Entropie $H(x)$ einer Quelle misst die durchschnittliche Unsicherheit oder den durchschnittlichen Informationsgehalt pro Symbol.

$$H(x) = \sum_{k=1}^{N} p(x_k) \cdot I(x_k)$$

$H(x)$	Entropie der Quelle
$p(x_k)$	Wahrscheinlichkeit des Symbols x_k
$I(x_k)$	Informationsgehalt des Symbols x_k
N	Anzahl der Symbole

Beispiel:

Mit den oben genannten Wahrscheinlichkeiten und Informationsgehalten:

$$H(x) = 0.3 \cdot 1.74 + 0.1 \cdot 3.32 + 0.1 \cdot 3.32 + 0.2 \cdot 2.32 + 0.3 \cdot 1.74 \approx 2.16 \text{ bit/Zeichen}$$

7.1.4 Redundanz der Quelle

Die Redundanz R_q misst den Unterschied zwischen dem maximal möglichen Entscheidungsgehalt und der tatsächlichen Entropie der Quelle.

$$R_q = H_0 - H(x)$$

R_q	Redundanz der Quelle
H_0	Entscheidungsgehalt der Quelle
$H(x)$	Entropie der Quelle

Beispiel:

$$R_q = 2.32 - 2.16 = 0.16 \text{ bit}$$

7.2 Codierung der Zeichen

7.2.1 Mittlere Codewortlänge

Die mittlere Codewortlänge L ist definiert als der gewichtete Durchschnitt der Längen der Codewörter, wobei jedes Gewicht der Auftretenswahrscheinlichkeit des entsprechenden Symbols gleich ist.

$$L = \sum_{k=1}^{N} p(x_k) \cdot L(x_k)$$

$L(x_k)$	Länge des Codeworts für das Symbol x_k
$p(x_k)$	Wahrscheinlichkeit des Auftretens von x_k

Beispiel:
Zunächst präsentieren wir die Codewörter jedes Zeichens und ihre entsprechenden Wahrscheinlichkeiten:

Zeichen	Codewort	Wahrscheinlichkeit p
a	0	0.3
b	110	0.1
c	1111	0.1
d	1110	0.2
e	10	0.3

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

$$L = \sum_{k=1}^{N} p(x_k) \cdot L(x_k)$$

wobei $L(x_k)$ die Länge des Codeworts für das Symbol x_k ist:

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

7.2.2 Redundanz des Codes

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

$$R_c = L - H(x)$$

R_c	Redundanz des Codes
L	Mittlere Codewortlänge
$H(x)$	Entropie der Quelle

Beispiel:

$$R_c = 2.4 - 2.16 = 0.24 \text{ bit}$$

7.2.3 Verbesserung der Codierung

Um eine Codierung mit geringerer Redundanz zu erreichen, sollten häufiger auftretende Symbole kürzere Codewörter erhalten. Eine Neuzuordnung der Codewörter kann die mittlere Codewortlänge reduzieren.

7.3 Maximierung der Entropie

Die Entropie ist maximal, wenn alle Zeichen gleichwahrscheinlich auftreten.

7.3.1 Binäre Quelle ohne Gedächtnis

Für eine binäre Quelle ohne Gedächtnis, bei der jedes Symbol x mit einer Wahrscheinlichkeit p auftritt, wird die Entropie H maximiert, wenn die Symbole gleich wahrscheinlich sind. Die Entropieformel für eine binäre Quelle ist:

$$H = -p \log_2(p) - (1-p) \log_2(1-p)$$

Um das Maximum zu finden, leiten wir H nach p ab und setzen die Ableitung gleich Null:

$$\frac{dH}{dp} = -\log_2(p) + \log_2(1-p) = 0$$
$$\implies p = \frac{1}{2}$$

Das Ergebnis zeigt, dass die Entropie maximal ist, wenn $p = 0.5$, also beide Symbole gleich wahrscheinlich sind.

Beispiel:
Wenn $p = 0.5$, dann ist die Entropie:

$$H = -0.5 \log_2(0.5) - 0.5 \log_2(0.5) = 1 \text{ bit}$$

7.3.2 Quelle mit drei Zeichen

Bei einer Quelle mit drei unterschiedlichen Symbolen, die mit den Wahrscheinlichkeiten p_1, p_2, und $1-p_1-p_2$ auftreten, ist die Entropieformel:

$$H = -p_1 \log_2(p_1) - p_2 \log_2(p_2) - (1 - p_1 - p_2) \log_2(1 - p_1 - p_2)$$

Um das Maximum der Entropie zu finden, setzen wir die partiellen Ableitungen nach p_1 und p_2 gleich Null. Dies führt zu den Bedingungen:

$$\begin{aligned} \ln(p_1) &= \ln(1 - p_1 - p_2) \\ \ln(p_2) &= \ln(1 - p_1 - p_2) \end{aligned}$$

Dies impliziert, dass $p_1 = p_2 = \frac{1}{3}$ und $1 - p_1 - p_2 = \frac{1}{3}$ die Entropie maximiert. Somit ist jedes Symbol gleich wahrscheinlich.

Beispiel:
Wenn $p_1 = p_2 = \frac{1}{3}$, dann ist die Entropie:

$$H = 3 \times \left(-\frac{1}{3} \log_2 \left(\frac{1}{3} \right) \right) \approx 1.585 \text{ bits}$$

7.4 Diskrete Quelle mit Gedächtnis

Eine diskrete Quelle mit Gedächtnis kann als Markov-Quelle erster Ordnung modelliert werden, bei der das Auftreten jedes Symbols von dem unmittelbar vorhergehenden Symbol abhängt. Die Übergangswahrscheinlichkeiten zwischen den Zuständen dieser Quelle sind durch ein Markov-Diagramm gegeben.

7.4.1 Berechnung der Zustandswahrscheinlichkeiten

Für die Markov-Quelle erster Ordnung mit den Zuständen x_1, x_2, und x_3 und den entsprechenden Übergangswahrscheinlichkeiten kann das stationäre Verhalten durch Lösen der folgenden Gleichungen bestimmt werden:

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

Tabelle 11: Bedingte Wahrscheinlichkeiten Markov-Diagramm

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

Die Lösungen dieser Gleichungen sind:

$$\begin{aligned} P(x_1) &= 0.28 \\ P(x_2) &= 0.32 \\ P(x_3) &= 0.4 \end{aligned}$$

7.4.2 Berechnung der Verbund- und bedingten Entropie

Die Verbundentropie $H(X,Y)$ für ein Markov-Modell kann durch die folgende Formel berechnet werden:

$$H(X,Y) = \sum_{i,j} P(x_i, y_j) \log_2 \left(\frac{1}{P(x_i, y_j)} \right)$$

Hierbei ist $P(x_i, y_j)$ die Verbundwahrscheinlichkeit des Übergangs von Zustand x_i zu Zustand y_j. Die bedingte Entropie $H(Y|X)$ wird dann wie folgt berechnet:

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

Beispiel:
Angenommen, die berechnete Verbundentropie ist 3.066 bit und die Entropie von X ist 1.57 bit, dann ist die bedingte Entropie:

$$H(Y|X) = 3.066 - 1.57 = 1.496 \text{ bit}$$

8 Kanalmodell

In der Informationstheorie ist ein **Kanalmodell** ein mathematisches Modell, das beschreibt, wie Informationen durch einen Kommunikationskanal übertragen werden. Ein wesentliches Element dieses Modells ist die **Kanalmatrix**, auch Übergangsmatrix genannt, die angibt, wie die Eingangssignale (gesendete Symbole) durch den Kanal in Ausgangssignale (empfangene Symbole) überführt werden. Diese Matrix beinhaltet die bedingten Wahrscheinlichkeiten $P(Y|X)$, wobei jedes Element der Matrix, die Wahrscheinlichkeit darstellt, dass ein bestimmtes Ausgangssymbol y empfangen wird, gegeben ein gesendetes Eingangssymbol x.

8.1 Kanalmatrix

Die allgemeine Form einer Kanalmatrix $P(Y|X)$ für einen Kanal mit m Eingangssymbolen und n Ausgangssymbolen ist:

$$P(Y|X) = \begin{pmatrix} P(y_1|x_1) & P(y_2|x_1) & \cdots & P(y_n|x_1) \\ P(y_1|x_2) & P(y_2|x_2) & \cdots & P(y_n|x_2) \\ \vdots & \vdots & \ddots & \vdots \\ P(y_1|x_m) & P(y_2|x_m) & \cdots & P(y_n|x_m) \end{pmatrix}$$

8.2 Bestimmung der Kanalmatrix $P(Y|X)$

Praktisch

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

Rechnerisch

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

$$\begin{bmatrix} y_0 & y_1 \end{bmatrix} = \begin{bmatrix} x_0 & x_1 \end{bmatrix} \cdot \begin{bmatrix} a & b \\ b & a \end{bmatrix}$$

$$y_0 = x_0 \cdot a + x_1 \cdot b$$
$$y_1 = x_0 \cdot b + x_1 \cdot a$$
$$P(Y/X) = \begin{bmatrix} a & b \\ b & a \end{bmatrix}$$

8.2.1 Nicht gestörter Kanal (rauschfrei)

Ein nicht gestörter Kanal hat die Einheitsmatrix als Kanalmatrix, unabhängig von der Anzahl der Ein- und Ausgänge des Kanals. Die Einheitsmatrix zeigt an, dass jedes gesendete Symbol x_i fehlerfrei und ohne Störung als entsprechendes Empfangssybol y_i empfangen wird.

- Jedes Element auf der Hauptdiagonalen der Matrix hat den Wert 1, was einer Wahrscheinlichkeit von 100% für die korrekte Übertragung angibt.
- Alle anderen Elemente in der Matrix haben den Wert 0, was bedeutet, dass keine Übertragungsfehler zwischen den verschiedenen Symbolen stattfinden.

Beispiel für einer Kanalmatrix mit drei möglichen Ein- und Ausgängen (3x3)

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

8.2.2 Vollständig gestörter Kanal

Ein vollständig gestörter Kanal hat eine Kanalmatrix, bei der alle Elemente gleich sind, was bedeutet, dass die empfangenen Signale nicht von den gesendeten abhängen. Daher können die übertragenen Informationen bei der Ankunft nicht zuverlässig rekonstruiert werden.

- Jedes Element der Kanalmatrix hat denselben Wert, der der umgekehrten Anzahl der möglichen Ausgangssignale entspricht, was eine gleichmässige Wahrscheinlichkeit für jedes Ausgangssignal unabhängig vom Eingang darstellt.
- Diese Art der Kanalmatrix führt dazu, dass keine nützliche Information vom Sender zum Empfänger übertragen wird, da die Übertragung völlig zufällig ist.

Beispiel für eine Kanalmatrix eines vollständig gestörten Binärkanals (2x2)

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

8.2.3 Verlustfreie Matrix

Für eine verlustfreie Matrix muss die Summe einer Zeile 1 ergeben.

Beispiel für eine verlustfreie (aber nicht rauschfreie) Kanalmatrix mit drei Eingängen und drei Ausgängen

$$P(Y|X) = \begin{pmatrix} 0 & 0.5 & 0.5 \\ 1 & 0 & 0 \\ 0.7 & 0.2 & 0.1 \end{pmatrix}$$

8.3 Berechnung fehlender Wahrscheinlichkeiten und Kanalmatrix

Wenn zwei der drei grundlegenden Informationen bekannt sind – die Wahrscheinlichkeiten der Eingangssignale $P(x_i)$, die Wahrscheinlichkeiten der Ausgangssignale $P(y_j)$ oder die Kanalmatrix – kann das dritte Element berechnet werden.

8.3.1 Berechnung der Kanalmatrix

Gegeben sind die Wahrscheinlichkeiten der Eingangssignale $P(x_1) = 0.3$ und $P(x_2) = 0.7$

und die Wahrscheinlichkeit der Ausgangssignale $P(y_1) = 0.34$ und $P(y_2) = 0.66$

Die Kanalmatrix $P(Y|X)$ wird wie folgt bestimmt:

$$P(X) = \begin{pmatrix} 0.3 & 0.7 \end{pmatrix}$$

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

Das resultierende lineare Gleichungssystem aus dem Produkt $P(Y) = P(X) \cdot P(Y|X)$ ergibt:

$$\begin{pmatrix} 0.34 & 0.66 \end{pmatrix} = \begin{pmatrix} 0.3 & 0.7 \end{pmatrix} \cdot \begin{pmatrix} a & b \\ b & a \end{pmatrix}$$

Dies führt zu den Gleichungen:

$$0.34 = 0.3a + 0.7b$$

$$0.66 = 0.3b + 0.7a$$

Lösen dieser Gleichungen ergibt $a = 0.9$ und $b = 0.1$, und somit:

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

8.3.2 Berechnung der Eingangswahrscheinlichkeiten $P(x_i)$

Wenn die Kanalmatrix $P(y_j|x_i)$ und die Ausgangswahrscheinlichkeiten $P(y_j)$ bekannt sind, können die Eingangswahrscheinlichkeiten $P(x_i)$ mit der totalen Wahrscheinlichkeit berechnet werden:

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

Beispiel

Unter der Annahme, dass wir bereits $P(y_j)$ berechnet haben und die Werte der Kanalmatrix kennen, verwenden wir das Bayes'sche Theorem, um $P(x_1|y_1)$ und $P(x_2|y_1)$ zu berechnen:

$$P(x_1|y_1) = \frac{P(y_1|x_1) \cdot P(x_1)}{P(y_1)} = \frac{0.9 \cdot 0.3}{0.34} \approx 0.7941$$

$$P(x_2|y_1) = \frac{P(y_1|x_2) \cdot P(x_2)}{P(y_1)} = \frac{0.1 \cdot 0.7}{0.34} \approx 0.2059$$

8.3.3 Berechnung der Ausgangswahrscheinlichkeiten $P(y_j)$

Die Ausgangswahrscheinlichkeiten $P(y_j)$ für jeden Kanalzustand y_j können mittels der totalen Wahrscheinlichkeit berechnet werden, die die Summe der Produkte der Eingangswahrscheinlichkeiten $P(x_i)$ und der entsprechenden bedingten Wahrscheinlichkeiten $P(y_j|x_i)$ aus der Kanalmatrix ist. Diese Methode wird allgemein in der Kommunikationstheorie angewendet, um die Verteilung der Ausgangssignale eines Kanals zu bestimmen.

Für einen Kanal mit Eingängen $x_1, x_2, \ldots, x_n$ und Ausgängen $y_1, y_2, \ldots, y_m$ wird $P(y_j)$ berechnet als:

$$P(y_j) = \sum_{i=1}^{n} P(x_i)P(y_j|x_i)$$

Beispiel

Gegeben sei ein binärer Kanal mit zwei Eingängen x_1 und x_2 und zwei Ausgängen y_1 und y_2. Die Eingangswahrscheinlichkeiten seien $P(x_1) = 0.3$ und $P(x_2) = 0.7$. Die Kanalmatrix sei:

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

Die Wahrscheinlichkeiten der Ausgangssignale werden dann berechnet als:

$$P(y_1) = P(x_1) \cdot P(y_1|x_1) + P(x_2) \cdot P(y_1|x_2) = 0.3 \cdot 0.9 + 0.7 \cdot 0.1 = 0.34$$

$$P(y_2) = P(x_1) \cdot P(y_2|x_1) + P(x_2) \cdot P(y_2|x_2) = 0.3 \cdot 0.1 + 0.7 \cdot 0.9 = 0.66$$

8.4 Entropie am Kanaleingang $H(X)$

Die Entropie $H(X)$ eines Zufallsprozesses am Kanaleingang, der durch die Wahrscheinlichkeitsverteilung $P(X)$ der Eingangssymbole charakterisiert wird, ist definiert als:

$$H(X) = -\sum_{i=1}^{m} P(x_i) \cdot \log_2 P(x_i) \quad \text{für alle } i \text{ mit } 1 \leq i \leq m$$

$H(X) =$ Entropie (Streuung) in bits / Zeichen
$P(x_i) =$ Wahrscheinlichkeit des Auftretens von Symbol x_i
$m =$ Anzahl der unterschiedlichen Symbole in der Eingabe

Beispiel

Die Auftrittswahrscheinlichkeit ist gegeben mit $P(x_1) = 0.3$ und $P(x_2) = 0.7$.

Die Entropie am Kanaleingang $H(X)$ ergibt sich folgendermassen:

$$\begin{aligned} H(X) &= -\sum_{i=1}^{m} P(x_i) \cdot \log_2 P(x_i) \\ &= -(P(x_1)) \cdot \log_2 P(x_1) - (P(x_2)) \cdot \log_2 P(x_2) \\ &= -0.3 \cdot \log_2 P(0.3) - 0.7 \cdot \log_2(0.7) \\ &= 0.8813 \text{ bit / Zeichen} \end{aligned}$$

8.5 Entropie am Kanalausgang $H(Y)$

Die Entropie $H(Y)$ am Kanalausgang misst die durchschnittliche Unsicherheit oder den Informationsgehalt der empfangenen Symbole. Sie wird berechnet als:

$$H(Y) = -\sum_{j=1}^{n} P(y_j) \cdot \log_2 P(y_j) \quad \text{für alle } j \text{ mit } 1 \leq j \leq n$$

$H(Y) =$ Entropie (Streuung) in bits / Zeichen
$P(y_j) =$ Wahrscheinlichkeit des Auftretens von Symbol y_j
$n =$ Anzahl der unterschiedlichen Symbole am Kanalausgang

Beispiel

Gegeben ist die Auftrittswahrscheinlichkeit mit $P(x_1) = 0.3$ und $P(x_2) = 0.7$ und die folgende Kanalmatrix P:

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

Die Entropie am Kanaleingang $H(Y)$ ergibt sich folgendermassen:

Zuerst müssen die y-Punkte berechnet werden (siehe 8.3.3):

$$\begin{aligned} H(Y) &= -\sum_{j=1}^{n} P(y_j) \cdot \log_2 P(y_j) \\ &= -(P(y_1) \cdot \log_2 P(y_1) + P(y_2) \cdot \log_2 P(y_2)) \\ &= -0.3 \cdot \log_2(0.3) - 0.7 \cdot \log_2(0.7) \\ &= 0.9248 \text{ bit / Zeichen} \end{aligned}$$

8.6 Transinformation T

Die Transinformation T oder $I(X;Y)$, auch gegenseitige Information genannt, misst die Reduktion der Unsicherheit über die Eingangssymbole durch die Kenntnis der Ausgangssymbole. Sie ist definiert als die Differenz zwischen der Entropie am Kanalausgang und der bedingten Entropie des Eingangs gegeben den Ausgang:

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

wobei $H(Y)$ die Entropie am Kanalausgang ist und $H(Y|X)$ die bedingte Entropie des Eingangs gegeben den Ausgang darstellt.

Rechenbeispiel

Gegeben sei ein binärer Kanal mit den Eingangswahrscheinlichkeiten $P(x_1) = 0.3$ und $P(x_2) = 0.7$, und einer Kanalmatrix:

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

Die Entropie am Kanalausgang $H(Y)$ wurde bereits als 0.9248 bit/Zeichen berechnet, und die bedingte Entropie $H(Y|X)$ wie folgt bestimmt:

$$H(Y|X) = -\sum_{i=1}^{2} \cdot \sum_{k=1}^{2} P(x_i) \cdot P(y_k|x_i) \cdot \log_2(P(y_k|x_i))$$

$$= -(P(x_1) \cdot P(y_1|x_1) \cdot \log_2(P(y_1|x_1)) - (P(x_1) \cdot P(y_2|x_1) \cdot \log_2(P(y_2|x_1)))$$
$$- (P(x_2) \cdot P(y_1|x_2) \cdot \log_2(P(y_1|x_2)) - (P(x_2) \cdot P(y_2|x_2) \cdot \log_2(P(y_2|x_2)))$$

$$= -(0.3 \cdot 0.9 \cdot \log_2(0.9)) - (0.3 \cdot 0.1 \cdot \log_2(0.1)) - (0.7 \cdot 0.1 \cdot \log_2(0.1)) - (0.7 \cdot 0.9 \cdot \log_2(0.9))$$

$$= 0.469 \text{ bit/Zeichen}$$

Die Transinformation T berechnet sich:

$$T = 0.9248 - 0.469 = 0.4558 \text{ bit/Zeichen}$$

8.7 Maximale Symbolrate R_{max}

Die maximale Symbolrate R_{max} gibt an, wie viele Symbole pro Sekunde maximal übertragen werden können, basierend auf der Bandbreite des Kanals und der Entropie am Kanaleingang:

$$R_{max} = \text{Bandbreite des Kanals} \cdot H(X) = \text{Übertragungsrate} \cdot T$$

Die Übertragungszeit ergibt sich folgendermassen:

$$t(s) = \frac{L \text{ (Datenmenge in bits)}}{T \cdot \text{ Übertragungsrate}} = \frac{L \text{ (Datenmenge in bits)}}{R_{max}}$$

Beispiel

Die maximale Übertragungsrate R_{max} bei einer Übertragungsrate von 1 kbit/s beträgt und T = 0.4558 $\frac{\text{bits}}{\text{Zeichen}}$

$$R_{max} = 1 \text{ kbit/s} \cdot T = 455.8 \text{ bit/s}$$

8.8 Entscheidungsfindung nach dem Maximum-Likelihood-Verfahren

Die Entscheidungsfindung (Detektion) nach dem Maximum-Likelihood-Verfahren wählt für jedes empfangene Symbol y_j das wahrscheinlichste gesendete Symbol x_i basierend auf der Kanalmatrix. Die Zuordnung erfolgt durch:

$$\hat{x}_j = \arg\max_{x_i} P(y_j|x_i)$$

$$\begin{aligned} \hat{x}_j &= \text{geschätztes Eingangssymbol für das empfangene Symbol } y_j, \\ P(y_j|x_i) &= \text{Wahrscheinlichkeit, dass } y_j \text{ empfangen wird, gegeben } x_i. \end{aligned}$$

Beispiel

Betrachten wir die folgende Kanalmatrix:

$$P(Y|X) = \begin{bmatrix} 0.2 & 0.5 & 0.3 \\ 0.7 & 0.2 & 0.1 \\ 0.4 & 0 & 0.6 \end{bmatrix}$$

Für jedes empfangene Symbol y_j, ist der Entscheider (rot markiert) jeweils die Zahl mit der höchsten Wahrscheinlichkeit jeder Spalte der Matrix:

$$P(Y|X) = \begin{bmatrix} 0.2 & \color{red}{0.5} & 0.3 \\ \color{red}{0.7} & 0.2 & 0.1 \\ 0.4 & 0 & \color{red}{0.6} \end{bmatrix}$$

Der Entscheider sieht also wie folgt aus:

$$\begin{aligned} y_1 &\to x_2 \\ y_2 &\to x_1 \\ y_3 &\to x_3 \end{aligned}$$

9 Blockcodes

9.1 Hammingdistanz zur Fehlererkennung

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

$$h = e* + 1$$
$$e = \lfloor \frac{h-1}{2} \rfloor$$
$$e* = h - 1$$

h	Die Hammingdistanz (= Anzahl der Kontrollstellen)
e	Maximale Anzahl sicher korrigierbaren Fehler
$e*$	Maximale Anzahl sicher erkennbaren Fehler

Beispiel 1: Bestimmung der notwendigen Hammingdistanz
Die notwendige Hammingdistanz h eines Codes, um sicher 5 Fehler zu erkennen, kann mit der Formel:

$$h = e* + 1$$

berechnet werden, wobei $e*$ die Anzahl der sicher erkennbaren Fehler ist. Für $e* = 5$ ergibt sich:

$$h = 5 + 1 = 6$$

Beispiel 2: Berechnung der Hammingdistanz eines Codes
Gegeben sind die Codewörter des folgenden Codes:

Codewort	Zeichen
00000000	A
11000000	B
10001100	C
01010000	D
01010101	E
10000110	F
11111111	G

Die kleinste Hammingdistanz h (Unterschied) zwischen zwei Codewörtern (z.B. A und B) wird bestimmt als:

$$h = 2$$

Dies deutet auf eine minimale Differenz von zwei Bitänderungen zwischen den ähnlichsten Codewörtern hin.

9.1.1 Mögliche / gültige Codewörter

2^m = Anzahl gültigen Codewörter (Anzahl der Spalten der Matrix ohne Einheitsmatrix)
2^{m+k} = Anzahl möglicher Codewörter (Grösse der Einheitsmatrix)

m	Nachrichtenstellen
k	Kontrollstellen

Beispiel
Gegeben ist ein Blockcode mit $m = 10$ Nachrichtenstellen, $k = 5$ Kontrollstellen, Hammingdistanz $h = 4$.

Die Anzahl der gültigen Codewörter ergibt sich:

$$2^m \Rightarrow 2^{10} = 1024$$

Die Anzahl der möglichen Codewörter ergibt sich:

$$2^{m+k} \Rightarrow 2^{15} = 32768$$

9.1.2 Dichtgepacktheit eines Codes

Ein Code wird als *dichtgepackt* bezeichnet, wenn alle möglichen Codewörter in den Korrigierkugeln um jedes gültige Codewort enthalten sind. Eine Korrigierkugel für ein Codewort umfasst alle Codewörter, die innerhalb einer bestimmten Hammingdistanz e von diesem Codewort liegen. Wenn die Hammingdistanz $h = 4$ ist, kann der Code bis zu $e = \frac{h-1}{2} = 1.5 \Rightarrow 1$ Fehler korrigieren.

Die Frage, ob ein Code dichtgepackt ist, lässt sich durch Berechnung der Anzahl der Codewörter in den Korrigierkugeln aller gültigen Codewörter beantworten. Dies wird wie folgt berechnet:

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

n	Gesamtzahl der Codestellen ($n = m + k$)
m	Nachrichtenstellen
w	Fehlerposition / Iterator (meistens i genannt)

Beispiel

Für $m = 10$, $k = 5$, $n = m + k = 15$ ergibt sich:

$$2^{10} \cdot \left(\binom{15}{0} + \binom{15}{1} \right) = 1024 \cdot (1 + 15) = 1024 \cdot 16 = 16384$$

Da 16384 kleiner als $2^{15} = 32768$, die Gesamtzahl der möglichen Codewörter, ist, sind nicht alle Codewörter in Korrigierkugeln enthalten. Somit ist der Code nicht dichtgepackt, was bedeutet, dass nicht jeder mögliche Fehlerzustand innerhalb einer Kugel um ein gültiges Codewort liegt und daher einige Fehlermuster nicht korrigierbar sind.

9.2 Hamming Blockcode

Der Hamming Blockcode ist eine spezielle Klasse von Blockcodes, die für ihre Fähigkeit zur Fehlerkorrektur und einfache Konstruierbarkeit bekannt sind. Die Codes basieren auf der Hamming-Prüfmatrix, die es ermöglicht, Fehler zu erkennen und zu korrigieren.

9.2.1 Prüfmatrix und Kontrollstellen

Die Prüfmatrix eines Hamming-Codes besteht typischerweise aus einer Einheitsmatrix und zusätzlichen Spalten, die für die Paritätskontrolle benötigt werden. Die Einheitsmatrix in der Prüfmatrix eines Hamming-Codes hat eine besondere Bedeutung:

- **Dimension der Einheitsmatrix**: Die Dimension der Einheitsmatrix in der Prüfmatrix entspricht der Anzahl der Kontrollstellen k im Code. Jede Zeile der Einheitsmatrix korrespondiert zu einer Kontrollstelle.
- Im idealen Hamming-code ist die Anzahl der Zeilen in der Einheitsmatrix gleich der Anzahl der Kontrollstellen k. In einer Prüfmatrix mit 4 Zeilen, würde man also eine 4×4 - Einheitsmatrix erwarten.

Beispiel: Gegeben sei die folgende Prüfmatrix eines Hamming-Blockcodes:

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

Die Anzahl der Kontrollstellen k ist hierbei 4.

9.2.2 Kontrollstellen aus Codewort bestimmen

Für ein gegebenes Codewort werden die Werte der Nachrichtenbits in die Prüfgleichungen eingesetzt, um die Kontrollbits zu berechnen. Diese Berechnung erfolgt durch eine XOR-Operation der entsprechenden Bits, die durch die Prüfmatrix bestimmt sind. Das Resultat jedes XOR bildet ein Kontrollbit, das zur Fehlererkennung und -korrektur dient.

Beispiel: Kontrollstellen für das Codewort 10110000010 bestimmen
Die Prüfmatrix sieht so aus:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & (x_{12})1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & (x_{13})1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & (x_{14})1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & (x_{15})1 \end{bmatrix}$$

Die hypothetischen Nachrichtenbits , basierend auf dem gegebenen Codewort, sind:

$$x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 1, x_5 = 0, x_6 = 0, x_7 = 0, x_8 = 0, x_9 = 0, x_{10} = 1, x_{11} = 0$$

Nun setzen wir diese Werte in die Gleichungen für die Kontrollbits ein (Jedes «1» der jeweiligen Zeile wird dazu addiert):

$$\begin{aligned} x_{12} &= (x_1 + x_2 + x_3 + x_4 + x_6 + x_7 + x_8) \mod 2 \\ &= (1 + 0 + 1 + 1 + 0 + 0 + 0) \mod 2 \\ &= 3 \mod 2 \\ &= 1 \\ x_{13} &= (x_1 + x_2 + x_3 + x_5 + x_6 + x_9 + x_{10}) \mod 2 \\ &= (1 + 0 + 1 + 0 + 0 + 0 + 1) \mod 2 \\ &= 3 \mod 2 \\ &= 1 \\ x_{14} &= (x_1 + x_2 + x_4 + x_5 + x_7 + x_9 + x_{11}) \mod 2 \\ &= (1 + 0 + 1 + 0 + 0 + 0 + 0) \mod 2 \\ &= 2 \mod 2 \\ &= 0 \\ x_{15} &= (x_1 + x_3 + x_4 + x_5 + x_8 + x_{10} + x_{11}) \mod 2 \\ &= (1 + 1 + 1 + 0 + 0 + 1 + 0) \mod 2 \\ &= 4 \mod 2 \\ &= 0 \end{aligned}$$

Die berechneten Kontrollbits $x_{12} = 1$, $x_{13} = 1$, $x_{14} = 0$, und $x_{15} = 0$ werden an das Ende des Nachrichtenteils des Codeworts angehängt, was das vollständige Codewort zu $10110000010\mathbf{1100}$ macht.

9.2.3 Fehlersyndrom

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

Beispiel:
Bestimmung des Fehlersyndroms für verschiedene Störungsfälle.

- Wenn nur x_2 gestört ist, ergibt sich das Fehlersyndrom durch die zweite Spalte der Prüfmatrix, da jede Zeile der Matrix einer Prüfstelle entspricht und die Einträge der Spalte anzeigen, welche Prüfstellen durch x_2 beeinflusst werden. Das Syndrom ist:
$$\begin{bmatrix}1\\1\\1\\0\end{bmatrix}$$
- Sind x_2 und x_{11} gestört, wird das Syndrom durch die Addition (mod 2) der zweiten und elften Spalte der Prüfmatrix berechnet:
$$\left(\begin{bmatrix}1\\1\\1\\0\end{bmatrix}+\begin{bmatrix}0\\0\\1\\1\end{bmatrix}\right) \mod 2 = \begin{bmatrix}1\\1\\0\\1\end{bmatrix}$$

Das Syndrom gibt direkt die beeinflusste Kontrollbit-Position an und wird genutzt, um die Fehlerstelle zu korrigieren, indem das entsprechende Bit im Codewort umgekehrt wird.

9.3 Zyklischer Hammingcode & Grad des Polynoms

Das Generatorpolynom für den zyklischen Hammingcode ist gegeben durch:

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

Jedes Codewort in einem zyklischen Hammingcode kann durch Multiplikation eines Nachrichtenpolynoms mit $g(x)$ und anschliessender Reduktion modulo $x^n - 1$ erzeugt werden.

- Der **Grad des Polynoms entspricht der Anzahl der Kontrollstellen** k (In diesem Fall 3, da x^3)
- Es gilt $n = 2^k - 1 = m + k$

Beispiel:

Gegeben ist ein Generatorpolynom $g(x) = 1 + x + x^3$

Ermittle alle gültigen Codewörter durch die schnelle Mehrfachaddition:

Grad des Polynoms entspricht der Anzahl der Kontrollstellen: $x^3 \rightarrow k = 3$

Da $n = 2^k - 1 = 2^3 - 1 = 7$ und $m = n - k = 7 - 3 = 4$

Der Vektor basierend auf dem Generatorpolnyom sieht folgendermassen aus:

$$\begin{bmatrix}1\\0\\1\\1\end{bmatrix}$$

Alle gültigen Codewörter haben die Länge n

9.3.1 Zykluslänge

$$n = 2^{\text{Grad des Polynoms}} - 1$$

9.3.2 Vektor eines Generatorpolynoms

Der höchste Exponent im Polynom bestimmt die Länge des Vektors. Wenn z.B. der höchste Exponent 3 ist, hätte der Vektor vier Elemente (von x^0 bis x^1)

Folgendes Polynom $g(x) = 1 + x + x^3$ entspricht auch der folgenden Schreibweise:

$$1 \cdot x^0 + 1 \cdot x^1 + 0 \cdot x^2 + 1 \cdot x^3 = x^0 + x^1 + x^3 = 1 + x + x^3$$

Somit leitet sich folgender Vektor ab:

$$\begin{bmatrix} 1 & (x^3) \\ 0 & (x^2) \\ 1 & (x^1) \\ 1 & (x^0) \end{bmatrix}$$

9.3.3 Schnelle Mehrfachaddition

1. Der Vektor des Polynoms wird als Ausgangspunkt verwendet
2. Der Vektor wird um k Positionen nach links verschoben
3. Jeder dieser Verschiebungen ist ein gültiges Codewort
4. Der Prozess wird fortgesetzt, bis alle Codewörter durch zyklische Verschiebungen generiert wurden
5. Jedes Codewort hat eine Codewortlänge von n

Beispiel:

Angenommen wir haben $n = 7, k = 3$ und einen Vektor:

$$\begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

1. Schreibe ein Wort (grün gefärbt) hin, das codiert werden soll
2. Schreibe nun den Vektor ”1011“ (rot gefärbt) darunter, so, dass die Summe 0 ergibt
3. Wiederhole diese Schritte solange, bis die Summe 0 ergibt
4. Das Codewort ergibt sich nun aus dem Input-Wort (links vom vertikalen Strich) + der Summe rechts vom vertikalen Strich
5. Wiederhole diese Schritte für alle möglichen Input-Wörter

Beim Input-Wort ”0001“ . . .

0	0	0	1				
			1	0	1	1	
				0	1	1	

. . . ergibt sich das Codewort: 0001011

Beim Input-Wort ”1000“ . . .

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

. . . ergibt sich das Codewort: 1000101

9.4 CRC-Code

CRC (Cyclic Redundancy Check) ist ein Fehlererkennungsverfahren, das in digitalen Netzwerken und Speichermedien verwendet wird, um die Integrität von Daten zu überprüfen. CRC-Codes nutzen ein Generatorpolynom, welches zur Konstruktion eines spezifischen CRC verwendet wird. Die Basis für CRC ist eine Polynomdivision, wobei der Rest der Division das CRC darstellt.

9.4.1 Generieren / Konstruieren eines CRC-Codes / Polynom

Ein CRC-Code wird generiert, indem das Datenpolynom durch das Generatorpolynom geteilt wird. Das Generatorpolynom ist üblicherweise so gewählt, dass es primitiv ist, d.h., es hat keine Faktoren ausser sich selbst und 1 im Bereich der modularen Arithmetik.

Für den CRC-Code mit dem Generatorpolynom $g(x) = 1 + x + x^4$ multipliziert man dieses mit $1 + x$ unter Verwendung der Modulo-2-Arithmetik, was folgendes ergibt:

(Multipliziere alle Zahlen des linken Terms zuerst mit 1 und dann mit x)

$$(1 + x + x^4) \cdot (1 + x) = 1 + x + x^4 + x + x^2 + x^5 \mod 2$$

Daraus resultiert das CRC-Generatorpolynom:

$$g(x) = 1 + x^2 + x^4 + x^5$$

Die Bits-Darstellung dieses Polynoms ist 110101.

9.4.2 Eigenschaften des CRC-Codes (Kontrollstellen)

Der Grad des höchsten CRC-Generatorpolynom $g(x)$bestimmt die Anzahl der Kontrollstellen k, die in diesem Fall 5 beträgt.

9.4.3 Hammingdistanz

Die Hammingdistanz eines CRC-Codes ist mindestens 4, was bedeutet, dass der Code bis zu drei Fehler erkennen und einen Fehler korrigieren kann.

9.4.4 Gültige Codewörter

1. Das rückgekoppelten Schieberegister wird mit den Zuständen 0, 0, 0 initialisiert
2. Danach wird $x1 = 1$ eingelesen. D.h. beim Register u^0 wird eine 1 eingeschoben, $u1 = u^0 \bigoplus 1 = 1$, u^3 erbt den alten Wert von u^2, somit $u^3 = 0$. Das führt zu der Tabelle: $x_1 = 1$ 0, 1, 1
3. Wiederhole diese Schritte für alle x_i
4. Das Ergebnis des letzten Schritts, sind die Kontrollstellen

Bemerkung: der $\bigoplus$ entspricht einem XOR-Operator

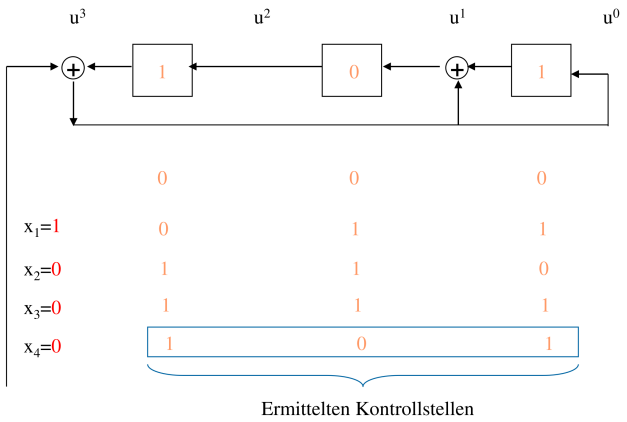


Abbildung 6: Rückgekoppeltes Schieberegister

9.5 Korrigierkugel

Die Korrigierkugel ist ein zentraler Begriff in der Codierungstheorie, der die Fehlertoleranz eines Codes illustriert. Sie zeigt visuell, wie viele Fehler innerhalb eines Codewortes korrigiert werden können.

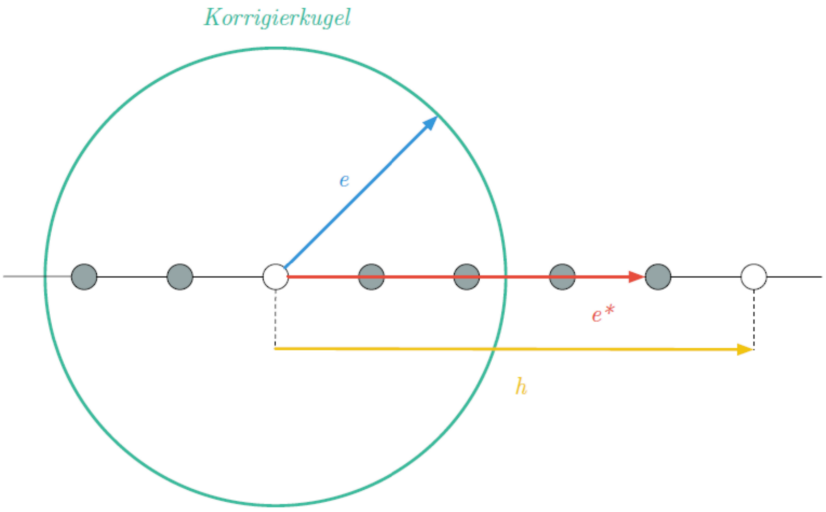


Abbildung 7: Korrigierkugel

- **Zentrum der Korrigierkugel:** Liegt bei jedem gültigen Codewort.
- **Radius der Korrigierkugel (e):** Entspricht der maximalen Anzahl an Fehlern, die sicher korrigiert werden können.
- **Hammingdistanz (h):** Definiert als die minimale Anzahl von Stellen, in denen sich zwei beliebige gültige Codewörter unterscheiden. Sie dient als Mass für die Fehlererkennungs- und -korrekturfähigkeit eines Codes.
- **Sicher erkennbare Fehler ($e*$):** Ist die maximale Anzahl von Fehlern, die sicher erkannt werden können, wobei $e* = h - 1$.
- **Ungültige Codeworte:** Liegen innerhalb der Korrigierkugel und stellen mögliche fehlerhafte Zustände des Codeworts dar, die zu einem gültigen Codewort korrigiert werden können.

Dieses Konzept wird in der Codierungstheorie verwendet, um zu bestimmen, wie robust ein Code gegenüber Übertragungsfehlern ist. Jede Kugel umfasst dabei ein Zentrum, das von einer Schale potenzieller Fehler umgeben ist, die jeweils korrigiert werden können, falls sie innerhalb des Radius e liegen.

9.6 Parity Blockcode

Der Parity Blockcode ist eine einfache Methode zur Fehlererkennung und -korrektur, die vor allem in Speicher- und Übertragungssystemen verwendet wird.

- **Prinzip:** Paritätsbits werden zu Daten hinzugefügt, um die Integrität der übertragenen oder gespeicherten Informationen zu sichern.
- **Fehlererkennung:** Ermöglicht das Erkennen einer ungeraden Anzahl von Fehlern innerhalb eines Codewortes.
- **Fehlerkorrektur:** Ein einfacher Parity Blockcode kann in der Regel nur einen Fehler korrigieren.
- **Formel:** Jedes Paritätsbit ist eine Funktion der Nachrichtenbits und wird so gewählt, dass die Gesamtanzahl der Einsen (einschliesslich des Paritätsbits) entweder gerade oder ungerade ist.

Beispiel:
Betrachten wir einen einfachen Parity Blockcode, der auf Längs- und Quersummen-Parität basiert. Die Datenmatrix ist wie folgt aufgebaut:

x_{11}	x_{12}	$\dots$	x_{1n}	$P1(n+1)$
x_{21}	x_{22}	$\dots$	x_{2n}	$P2(n+1)$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
x_{k1}	x_{k2}	$\dots$	x_{kn}	$P(k,n+1)$
$P(k+1)1$	$P(k+1)2$	$\dots$	$P(k+1)n$	$P(k+1)(n+1)$

In einem solchen 2-Dimensionalen Feld, bleibt die Anzahl der sicher erkennbaren Fehler konstant, unabhängig der Grösse von n und k. Das liegt daran, dass, wenn wir einen Fehler haben, der von der n- und p-Achse sicher bestimmt werden kann. Haben wir drei Fehler, so kann die eine Achse zwei Fehler sicher bestimmen und die andere Achse einen Fehler sicher bestimmen.

Bei 4 oder mehr Fehler, kann die Anzahl der Fehler nicht mehr sichergestellt werden. Siehe folgendes Beispiel:

x_{11}	x_{12}	$\dots$	x_{1n}	$P1(n+1)$
x_{21}	x_{22}	$\dots$	x_{2n}	$P2(n+1)$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
x_{k1}	x_{k2}	$\dots$	x_{kn}	$P(k,n+1)$
$P(k+1)1$	$P(k+1)2$	$\dots$	$P(k+1)n$	$P(k+1)(n+1)$

Da in der Spalte von k bereits ein Fehler x_{11} aufgetreten ist, kann ein zweiter Fehler x_{k1} nicht mehr erkannt werden. Das selbe gilt für die Zeilen von n. Somit ist die maximale Anzahl der erkennbaren Fehler = 3.

Hammingdistanz

Die Hammingdistanz ist um 1 grösser als die Anzahl der erkennbaren Fehler und somit: $h = 4$

10 Faltungscodes

10.1 Generatorpolynom aus Encoder-Schaltung bestimmen

Die Generatorenpolynome einer Schaltung können anhand der Verbindungen zwischen den Verzögerungselementen, den XOR-Gattern und den Ausgangspunkten ermittelt werden. Jedes XOR-Gatter kann als eine binäre Addition (Modulo-2-Addition) betrachtet werden.

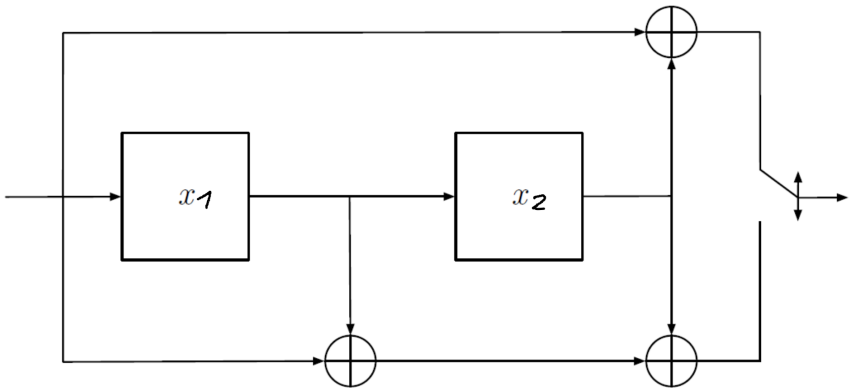


Abbildung 8: Encoder-Schaltung eines Faltungscoders

1 Für das obere Ausgangssignal:

1.1 Das direkte Signal entspricht dem Term 1

1.2 Das zweite XOR Signal geht durch zwei x, somit entspricht es dem Term x^2

1.3 Das ergibt das Polynom $g_1(x) = 1 + x^2$

2 Für das untere Ausgangssignal:

2.1 Das erste XOR-Gatter hat an einem Eingang 1 und am anderen x, somit ist der erste Term $g_1(x) = 1 + x$

2.2 Das zweite XOR-Gatter ”erbt“ an einem Eingang vom ersten Gatter $(1 + x)$ und am anderen liegt x^2 an, was zu $1 + x + x^2$ führt.

2.3 Das ergibt das Polynom ist also: $g_2(x) = 1 + x + x^2$

10.1.1 Zustandsdiagramm

x2	x1	Eingang	Ausgang (Oben/Unten)
0	0	0	0/0
0	0	1	1/1
0	1	0	0/1
0	1	1	1/0
1	0	0	1/1
1	0	1	0/0
1	1	0	1/0
1	1	1	0/1

10.2 Zustandsdiagramm

- Es gibt $2^{(\text{Anz. Schieberegister})}$ an Zuständen
- Bei jedem Zustandswechsel wird die folge von Bits notiert $\rightarrow$ Eingangsbits | Ausgangsbits

Vorgehen, um ein Zustandsdiagramm zu zeichnen:

1 Notiere zuerst alle Zustände (Es kann helfen die möglichen Schieberegisterzustände im State zu definieren)

2 In unserem Fall gibt es $2^2 = 4$ Zustände (00, 01, 10, 11))

3 Beginne beim ersten (initial-) Zustand 00 und mache einen Pfeil zum Zustand 01 und definiere de Verbindung mit 1/__

4 Von da ziehe einen Pfeil zu 10 wenn eine 0 eingelesen wird und ein Pfeil zu 11 wen eine 1 eingelesen wird

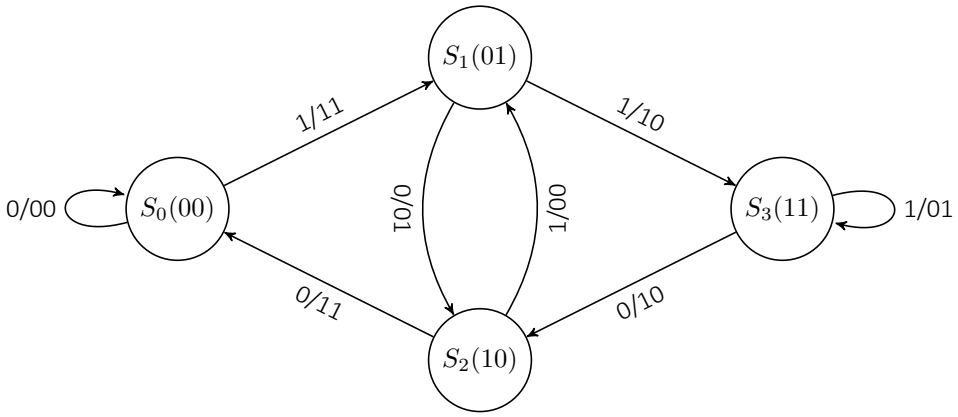
5 Mache das für alle Zustände (Von jedem Zustand gehen zwei Pfeile (0, 1) weg)

6 Jetzt haben wir bei allen Pfeilen 0/__ oder 1/__ . Schreibe nun die Ausgänge hin

7 Z.B. wäre beim Zustand $S_0(00)$ 0/__ die Ausgänge beide LOW, somit 0/00

8 Mache das für alle Pfeile / Verbindungen

Zustandsdiagramm mit den Zustandswechsel



10.3 Codieren / Decodieren

10.3.1 Codieren einer Nachrichtenfolge

Um eine Nachrichtenfolge zu Codieren, liest man die gegebenen Zustände ein, z.B.: u[n] = 1, 0, 0, 1, 1, 0, 1 und schaut wie die Ausgänge sich verhalten.

Beispiel:
Haben wir die Nachrichtenfolge: $\{u[n]\} = \{1, 0, 0, 1, 1, 0, 1\}$

- ...beginnen wir beim Startzustand S_0, mit einer 1 gehen wir in Zustand S_1, somit haben wir die Reihenfolge $S_0 \rightarrow S_1 \rightarrow \ldots$
- Das muss man nun mit allen Zeichen der Nachricht machen, somit kommt man zu folgender Reihenfolge: $S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_0 \rightarrow S_1 \rightarrow S_3 \rightarrow S_2 \rightarrow S_1$
- Wird noch verlangt, dass die Übertragung im Nullzustand endet, müssen die nötigen Zustände, um zu S_0 zurückzukehren noch ergänzt werden: $S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_0 \rightarrow S_1 \rightarrow S_3 \rightarrow S_2 \rightarrow S_1 \rightarrow (\text{Ergänzung}) S_2 \rightarrow S_0$
- Das Codewort entspricht nun den Werten (Pegel) der Ausgänge der jeweiligen Zustände. z.B. von S_0 zu S_1 wäre es 11. Schreibt man die ganze Zahlenfolge um (wobei der Startzustand S_0 NICHT notiert wird), kommt man zum Codewort:

$$\{u[n]\} = \{11, 01, 11, 11, 10, 10, 00, 01, 11\}$$

10.3.2 Decodieren einer Nachricht

Beim Decodieren beginnt man am Ende und "reversed-engineered" die Nachricht, sofern ein Netzdiagramm vorhanden ist oder man kann die Zustände im Zustandsdiagramm auslesen.

Beispiel:
Gegeben ist die verschlüsselte Nachricht: r[n] = 11, 01, 11, 11, 10, 10, 00, 01, 11

1 Man weiss der Startzustand ist S_0

2 Wenn die Outputs 11 sind, geht man in S_1. Somit benötigen wir eine 1 um in S_1 zu gehen.

3 Jetzt kommt 01, um von S_1 zu S_2 zu gehen, benötigen wir eine 0. Somit ist der nächste Wert 0

4 Wiederhole diese Schritte für alle codierten Wörter

5 Ziehe nun die letzten x Ziffern wegen den Tailbits (In unserem Fall zwei)

6 Somit entspricht die decodierte Nachricht:

$$\{u[n]\} = \{1, 0, 0, 1, 1, 0, 1\}$$

10.4 Tailbits bestimmen

Die Anzahl der Tailbits ergibt sich aus der Anzahl der Schieberegister. Hat man eine Encoder-Schaltung mit 4 Schieberegister, so hat man 4 Tailbits.

10.5 Anzahl Ausgangsbits bestimmen

Die Anzahl der Ausgangsbits entspricht der Anzahl an Schieberegister + Das Eingangsbit. D.h. in einer Schaltung mit 3 Schieberegister werden 4 Bits für die Berechnung der Ausgangsbits benötigt.

10.6 Block-Coderate

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

$$R = \frac{n}{k+m} \qquad \text{oder bei einem Faltungscodierer: } \frac{n}{2 \cdot (k+m)}$$

$n =$ Anzahl der codierten Bits
$k =$ Anzahl der gesendeten Bits
$m =$ Anzahl der Tail-Bits

R	Die Block-Coderate
n	Anzahl der codierten Bits
k	Anzahl der gesendeten Bits
m	Anzahl der Tail-Bits

Beispiel: Berechnung der Block-Coderate
Gegeben sei ein Faltungscodierer mit einem Encodergedächtnis von $m = 4$ und einer Anzahl von $k = 185$ Nutzdatenbits. Die Block-Coderate R berechnet sich dann als:

$$R = \frac{185}{2 \cdot (185+4)} \approx 0.489$$

Dies bedeutet, dass 48.9% der gesendeten Bits Nutzdatenbits sind, der Rest sind Tail-Bits und Redundanz zur Fehlerkorrektur.

10.7 Encoder-Schaltung aus Impulsantwort bestimmen

Gegeben sind die beiden Impulsantworten $g_1 = \{1, 1, 0\}$ und $g_2 = \{1, 1, 1\}$

- **Anzahl Schieberegister**: $2^{\text{Grösse des grössten Generatorpolynom}}$, in unserem Fall $g_1 = \{1, 1, 0\} \Rightarrow x^2 \rightarrow 2$ Schieberegister (Note: $g_1 = \{1\}$ wäre jetzt x^0, also kein Schieberegister)
- **Bestimmen der XOR-Gatter**: Betrachten wir $g_1 = \{1, 1, 0\}$, so haben wir bei x^0 (direkte Verbindung) und bei x^1 (erstes Schieberegister) eine "1". Somit sind diese beiden über ein XOR verbunden
- Bei $g_2 = \{1, 1, 1\}$ sind x^0 & x^1 und x^1 & x^2 verbunden

Die Schaltung sieht dann folgendermassen aus:

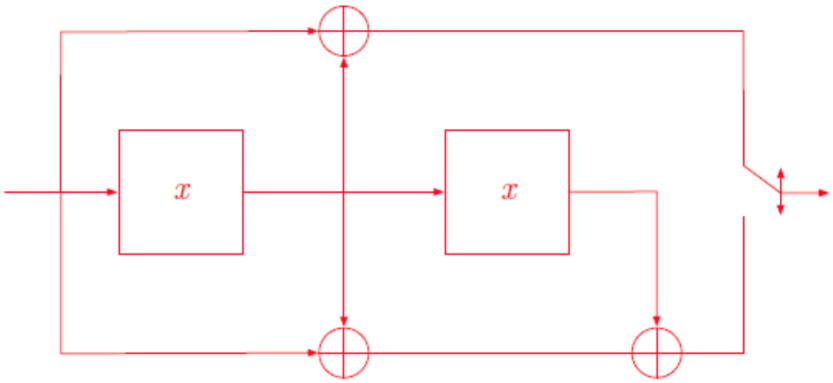


Abbildung 9: Encoder-Schaltung aus Impultsantwort

11 Diverses

11.1 Zaubertrick mit Tabellen

Bei jeder Tabelle, in der die gesuchte Zahl vorkommt, wird 2^{m-1} gerechnet. Zum Schluss muss die Summe zusammengerechnet werden und man kommt auf die gesuchte Zahl.

11.2 Stellenwertsystem vs. römisches Zahlensystem

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

11.3 Unterschied zwischen Binärzahl 0000 und 0000'0000

Beide Zahlen stellen denselben Wert dar, nämlich 0. Allerdings bezieht sich 0000 auf 4 Bit, 0000'0000 jedoch auf 8 Bit. Diese Unterscheidung ist wichtig, da durch die zusätzlichen Nullen auch mehr Speicherplatz impliziert wird.

11.4 TI-Nspire

Wahrscheinlichkeit

Die "Vektoren", um die Wahrscheinlichkeit zu berechnen, findet man unter: menu $\rightarrow$ (5) Wahrscheinlichkeit $\rightarrow$ (3) Kombinationen

Beispiel:

$$\frac{\text{nCr}(6,3)\cdot\text{nCr}(43,3)}{\text{nCr}(49,6)}$$

Modulo

Der Modulo-Operator befindet sich unter: menu $\rightarrow$ (2) Zahl $\rightarrow$ (6) Rest

Beispiel:

$$\text{remain}(7,4)\rightarrow 3$$

Polynomdivision

Die Polynomdivision findet man unter: menu $\rightarrow$ (3) Algebra $\rightarrow$ (3) Entwickle