Zusammenfassung DigCod

1 INHALT

2 MATHEMATISCHE GRUNDLAGEN

2.1 BEGRIFFE

- **Mengen**: $\mathbb{Z}$ (ganze Zahlen), $\mathbb{Z} = \{..., -2, -1, 0, 1, 2, ...\}$
- **Rechenoperationen**: +, -, *, /
- **Wertigkeit**: entspricht der Basis hoch der Stelle der ursprünglichen Zahl, angefangen mit 0 beim Least Significant Bit (LSB), z.B. bei Hexadezimal 16^1

Allgemein: $N = d_n R^n + ... + d_1 R^1 + d_0 R^0$

Ziffer $\in Z_{10}$ — Das entsprechende Zahlensystem — **Wertigkeit**

- **Algebraische Strukturen**:
  - **Gruppe** (werden Elemente aus der Menge verknüpft, so muss das Ergebnis wieder ein Element der Menge ergeben): $G(\mathbb{Z}, +)$ $\quad a + b = c \quad$ $, wobei\ a, b, c \in \mathbb{Z}$
  - **Ring**: $R(\mathbb{Z}, +, *)$
  - **Körper**: $K(\mathbb{Z}, +, *)$
  - ein Ring ist ein Körper, wenn:
    - jedes Element des Rings außer der Null ein (multiplikatives) Inverses hat
    - die Multiplikation kommutativ ist: ab = ba
    - das Distributivgesetz gilt: a(1-b) = a - ab
    - wenn er eine 1 hat (multiplikatives neutrales Element)

2.2 MODULO

Wenn wir eine Multiplikation durchführen und die Abgeschlossenheit im $\mathbb{Z}_5 = \{0,1,2,3,4\}$ haben wollen, müssen wir $mod\ 5$ rechnen.

Beispiel: $3 * 4 = 12$ aber $3 * 4\ mod\ 5 = 2$

$-17 = 2 \cdot (-7) + (-3)$ oder $-17 = 3 \cdot (-7) + 4$

$17 = q \cdot x - 7 \rightarrow 17 = -1 \cdot (-24) - 7$ oder $17 = -2 \cdot (-12) - 7$

$17 \mod 7 = x$	$x = 3$
$-17 \mod -7 = x$	$x = -3$ oder $x = 4$
$17 \mod x = 7$	$x = 10$
$17 \mod x = -7$	$x = -12$ oder $x = -24$
$17 \mod x = 17$	$x = 18$ (alles > 17 erlaubt)
$x \mod 17 = 7$	$x = 7$

2.3 DATENWÖRTER

Alle vier Darstellungsweise hier sind dasselbe:

- Zahl: $1001_2 = 9_{10}$
- Tupel: $(1,0,0,1)$
- Vektor: $[1,0,0,1]^T$
- Polynom: $u^3 + u^0 = u^3 + 1$

2.4 POLYNOME

Dezimalzahl 1205 als Polynom: $u^4 + 2u^3 + 5$

Beispiel eines Polynoms 5-ten Grades mit Koeffizienten aus $\mathbb{Z}_2 = \{0,1\}$:

$1u^5 + 0u^4 + 1u^3 + 0u^2 + 0u^1 + 1u^0 \quad = \quad u^5 + u^3 + 1$

2.4.1 Polynom-Multiplikation

$(u5 \; + \; u2 \; + \; u0)(u2 \; + \; u0) \; mod \; 2 =$

$u^7 + u^5 + u^4 + u^2 + u^2 + u^0 \; mod \; 2 =$

$u^7 + u^5 + u^4 + 2u^2 + u^0 \; mod \; 2 =$

$u^7 + u^5 + u^4 + 1$

2.4.2 Polynom-Division

Aufgabe: $\frac{x^6}{x^3+x+1}$

$$
\begin{array}{rlllll}
 & x^6 & & & & : x^3 + x + 1 = x^3 + x + 1 \\
- & x^6 & +x^4 & +x^3 & & \\
\hline
 & & x^4 & +x^3 & & \\
 & - & x^4 & & +x^2 & +x \\
\hline
 & & & x^3 & +x^2 & +x \\
 & & - & x^3 & & +x \quad +1 \\
\hline
 & & & & x^2 & \quad +1
\end{array}
$$

$$\frac{x^6}{x^3 + x + 1} = (x^3 + x + 1) \text{ Rest } (x^2 + 1)$$

1. Dividieren: $\frac{x^6}{x^3+x+1} = x^3 \; (wegen \; \frac{x^6}{x^3} = \; x^3)$
2. Multiplizieren: $x^3(x^3 + x + 1) = x^6 + x^4 + x^3$
3. Subtrahieren: $x^6 - (x^6 + x^4 + x^3) = \; x^4 + x^3$
4. Nun geht es von vorne los
5. Dividieren: $\frac{x^4+x^3}{x^3+x+1} = x \; (wegen \; \frac{x^4}{x^3} = \; x)$
6. Multiplizieren: $x(x^3 + x + 1) = x^4 + x^2 + x$
7. Subtrahieren: $(x^4 + x^3) - (x^4 + x^2 + x) = x^3 + x^2 + x$
8. Usw.

2.4.3 Polynom-Division rückgängig machen

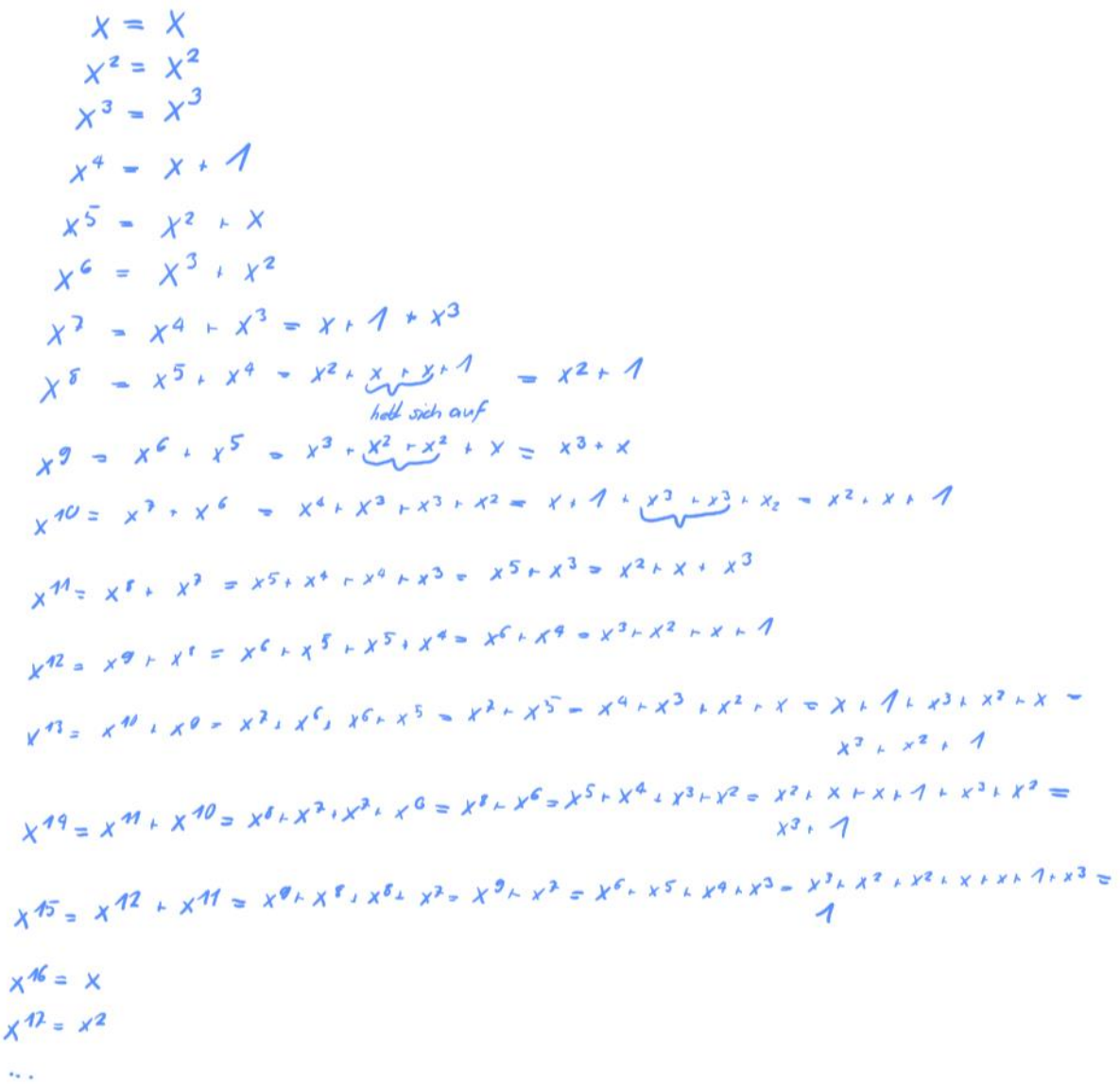
Als Resultat einer Polynomdivision haben Sie (x^4+1) Rest $(x+1)$ erhalten. Was war vermutlich die ursprüngliche Division?

$$(x^4 + 1) + \frac{x + 1}{x^4 + 1} = \frac{(x^4 + 1)^2 + x + 1}{x^4 + 1} = \frac{x^8 + x}{x^4 + 1}$$

Mod 2 nicht vergessen!

2.5 Zyklische Gruppe / Galois

Primitives Polynom: $x^4 + x + 1$



Die Zykluslänge ist somit 15.

Folgende Elemente wurden erzeugt: $\{0010, 0100, 1000, 0011, 0110, 1100, 1011, 0101, 1010, 0111, 1110, 1111, 1101, 1001, 0001\}$

Die binären Zahlen sind aus der Polynomschreibweise oben abgeleitet.

3 Wahrscheinlichkeit

3.1 Begriffe

- Ergebnismenge: Menge aller möglichen Ausgänge, wird mit Ω bezeichnet
- Ein einzelnes Element aus der Ergebnismenge wird mit ω bezeichnet

3.2 WAHRSCHEINLICHKEIT

Wahrscheinlichkeit heisst auf Englisch probability, daher sind die Formeln jeweils mit p

- Wahrscheinlichkeit eines Ereignisses: $P(A) = \frac{Anzahl\ der\ günstigen\ Ereignisse\ A}{Anzahl\ aller\ Ereignisse\ |\Omega|}$
- Wahrscheinlichkeit, dass ein Ereignis nicht auftritt: $P(\bar{A}) = 1 - P(A)$
- Wahrscheinlichkeit eines unmöglichen Ereignisses: $P(\theta) = 0$
- Wertebereich der Wahrscheinlichkeit: $0 \leq P(A) \leq 1$
- Seien E1 und E2 zwei Ereignisse.
    - Formel, wenn Ereignis E2 nur eintreten kann, wenn E1 bereits eingetreten ist: $P(E1E2) = P(E1) \cdot P(E2|E1)$

      $P(E2|E1) = \frac{P(E1E2)}{P(E1)}$
    - Formel, wenn die Ereignisse unabhängig sind: $P(E1E2) = P(E1) \cdot P(E2)$

3.3 KOMBINATORIK

Buchstaben:

- Anzahl der Zahlenkugeln in der Kiste n
- Anzahl der Ziehungen k
- Anzahl Möglichkeiten N

Es gibt 4 Varianten:

- Geordnet, mit Wiederholung/Zurücklegen (die Reihenfolge ist relevant und die gleiche Zahl kann mehrmals vorkommen) $N = n^k$
- Geordnet, ohne Wiederholung/Zurücklegen (die Reihenfolge ist relevant und die gleiche Zahl kann nicht mehrmals vorkommen) $N = \frac{n!}{(n-k)!}$
- Ungeordnet, mit Wiederholung/Zurücklegen (die Reihenfolge ist egal und die gleiche Zahl kann mehrmals vorkommen) $N = \frac{(n+k-1)!}{k! \cdot (n-1)!}$
- Ungeordnet, ohne Wiederholung/Zurücklegen (die Reihenfolge ist egal und die gleiche Zahl kann nicht mehrmals vorkommen) $N = \frac{n!}{(n-k)! * k!} = \binom{n}{k}$

3.4 LOTTO

Lotto ist ein Beispiel einer geordneten Stichprobe ohne Wiederholung.

Beispiel Formel für 5 richtige aus 49 Zahlen: $P(4T) = \frac{\binom{6}{5} * \binom{43}{1}}{\binom{49}{6}}$

3.5 BITFEHLER

Wahrscheinlichkeit Bitfehler: 10^{-3}

Wahrscheinlichkeit, dass Datenblock mit 153600 Bits fehlerhaft ist: $P = 1 - (1 - 10^{-3})^{153600}$

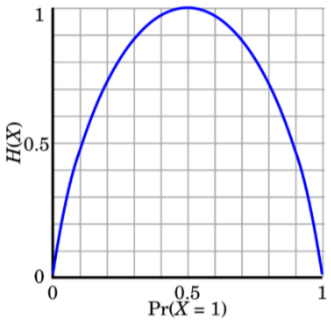
Wahrscheinlichkeit, dass höchstens 3 Fehler auftreten: $P = P(0F) + P(1F) + P(2F) + P(3F) =$

$$\binom{153600}{0} * 10^{-3^0} * (1 - 10^{-3})^{153600} + \binom{153600}{1} * 10^{-3^1} * (1 - 10^{-3})^{153599} + \binom{153600}{2} * 10^{-3^2} * (1 - 10^{-3})^{153598} + \binom{153600}{3} * 10^{-3^3} * (1 - 10^{-3})^{153597}$$

4 INFORMATIONSTHEORIE

4.1 BEGRIFFE UND FORMELN

- Zeichenvorrat: z.B. $X = \{a, b, c, d\}$
- Anzahl der Zeichen im Zeichenvorrat: $N = 4$
- Entscheidungsgehalt: Mass für den Aufwand, der zur Bildung einer Nachricht notwendig ist.
  $H_0 = log_2(N) = [bit]$
- Entscheidungsfluss
  $H_0^* = \frac{H_0}{\tau} = [bit/s]$
- Informationsgehalt: sagt aus, wie viele Elementarentscheidungen zur Bestimmung dieses Zeichens zu treffen sind (je kleiner die Wahrscheinlichkeit eines Zeichens, desto grösser der Informationsgehalt)
  $I(x_k) = log_2\left(\frac{1}{P(x_k)}\right) = [bit]$



- Mittlerer Informationsgehalt / Entropie
  $H(X) = \sum_{k=1}^{N} P(x_k) * I(x_k) = [bit/Zeichen]$
- Tatsächliche Codewortlänge
  $L(x_k) = Aufgerundet\ auf\ ganze\ Zahl\left(log_2\left(\frac{1}{P(x_k)}\right)\right) = [bit]$
- Entropie des Codes/Mittelwert der Codewortlänge
  $H_c(X) = L = \sum_{k=1}^{N} P(x_k) * L(x_k) = [bit/Zeichen]$

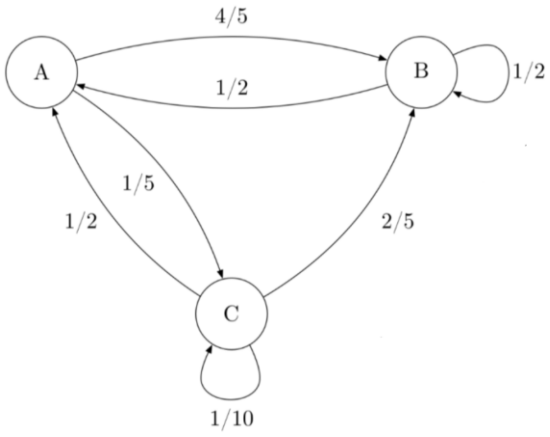
4.2 SHANNON'SCHES CODIERUNGSTHEOREM

- $H(X) \leq L \leq H(X) + 1$
- Redundanz der Quelle: $R_Q = H_0 - H(X) = [bit/Zeichen]$
- Redundanz des Codes: $R_C = L - H(X) = [bit/Zeichen]$

4.3 BEDINGTE WAHRSCHEINLICHKEITEN (MIT GEDÄCHTNIS)

In der Sprache sind gewisse Zeichen nach gewissen Zeichen wahrscheinlicher als andere.

Wir haben folgendes Markov-Diagramm:



Daraus lässt sich folgendes ablesen:

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

Daraus lässt sich $p(x)$ mithilfe eines GLS berechnen:

$$\left|\begin{array}{c} P(A) = 0P(A) + {}^{1}/_{2}\,P(B) + {}^{1}/_{2}\,P(C) \\ P(B) = {}^{4}/_{5}\,P(A) + {}^{1}/_{2}\,P(B) + {}^{2}/_{5}\,P(C) \\ P(C) = {}^{1}/_{5}\,P(A) + 0P(B) + {}^{1}/_{10}\,P(C) \\ 1 = P(A) + P(B) + P(C) \end{array}\right| \qquad \left|\begin{array}{l} P(A) = {}^{9}/_{27} = 0.333 \\ P(B) = {}^{16}/_{27} = 0.593 \\ P(C) = {}^{2}/_{27} = 0.074 \end{array}\right|$$

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

Daraus lässt sich dann $p(x,y) = p(x) * p(y|x)$ berechnen:

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

$P(A,B) = P(A) * P(B|A) = {}^{1}/_{3} * {}^{4}/_{5} = {}^{4}/_{15}$
Usw.

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

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

Bedingte Entropie: $H(Y|X) = H(X,Y) - H(X)$

$$H_0 \geq H(Y) \geq H(Y|X)$$ H_0 ist von der Sprache gegeben

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

5 KOMPRIMIERUNG

Das Ziel der Datenkomprimierung ist, den Aufwand der Datenspeicherung und Datenübertragung zu reduzieren. Das heisst Entfernen von Redundanz und Irrelevanz.

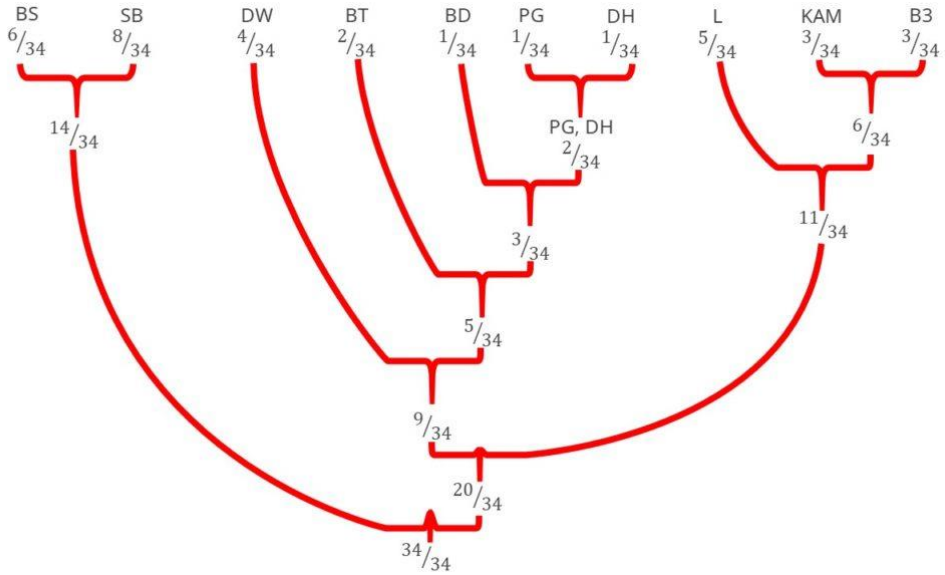
5.1 HUFFMAN-CODIERUNG

Bei dieser Codierung werden oft vorkommenden Zeichen kurze Codeworte zugewiesen und seltener vorkommenden Zeichen längere Codeworte. Die Präfixeigenschaft (kein anderes CW beginnt mit einer Zeichenfolge, die ein anderes CW definiert) wird erfüllt.

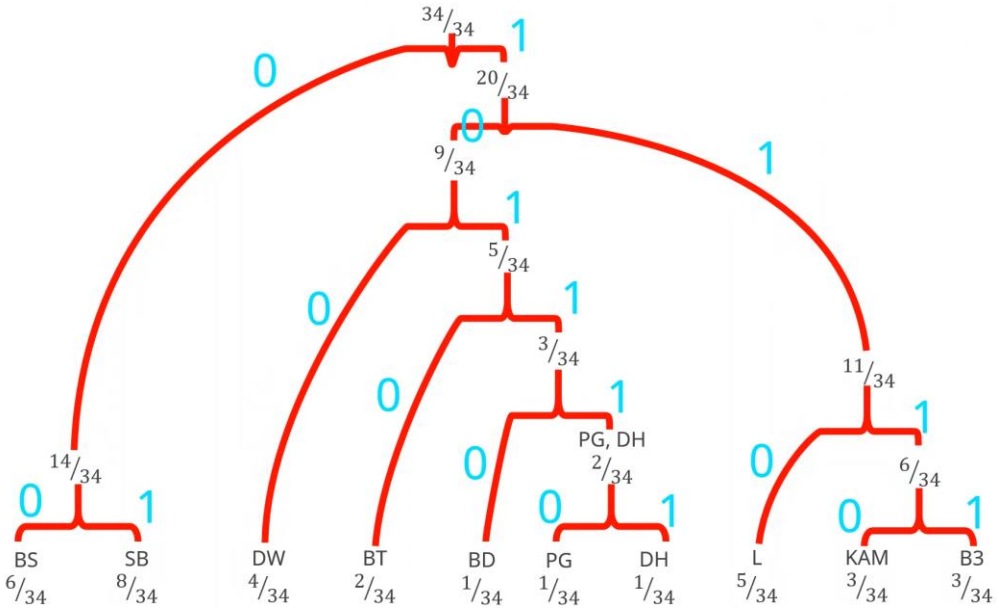
1. Die Zeichen aufsteigend nach Wahrscheinlichkeit aufschrieben

PG	DH	BD	BT	B3	KAM	DW	L	BS	SB
$^1/_{34}$	$^1/_{34}$	$^1/_{34}$	$^2/_{34}$	$^3/_{34}$	$^3/_{34}$	$^4/_{34}$	$^5/_{34}$	$^6/_{34}$	$^8/_{34}$

2. Die zwei Zeichen mit der niedrigsten Auftretenswahrscheinlichkeit auswählen und zu einem Zeichen verschmelzen. Die Auftretenswahrscheinlichkeit wird addiert und das Zeichen neu einsortiert. Diesen Schritt wiederholen, bis es nur noch einen Knoten mit der Wahrscheinlichkeit 1 gibt.



3. Nun muss der Baum umgedreht werden, allen linken Kanten eine 0 und allen rechten Kanten eine 1 zugewiesen werden. Die Codewörter können dann von oben her abgelesen werden.



5.2 Lauflängenkomprimierung

Lauflängenkomprimierung wird auch Run Length Encoding (RLE) oder Run Length Coding (RLC) genannt. Sie wird von Bildformaten wie BMP, PCX oder TIFF verwendet. Sie basiert auf der Verkürzung von Wiederholungen aufeinanderfolgender Elemente.

Beispiel bei einem Wort: Agggbbehfffgggg → A3g2beh3f4g

Bei der Codierung von Bitfolgen ist es etwas einfacher, es existieren ja nur Folgen von 0 und 1. Der Codierer und Decodierer müssen sich anfänglich darauf einigen, ob mit 0 oder 1 begonnen wird.

Beispiel: 1111 1110 0000 1000 0001 11111111 1110 0000 1000 0001 1111 → 7 5 1 6 5 (Start mit 1)
Da die grösste Zahl im Code 7 ist, werden 3 Bits benötigt pro Stelle ($2^3 = 8$).
Das Codewort ist also: 111 101 001 110 101

5.3 Lempel-Ziv

Dieses Verfahren ist gut geeignet, wenn der zu komprimierende Code wiederkehrende Muster oder Phrasen hat.

5.3.1 Codierung

Zeichenkette: ␣ab␣abc␣aba␣c ≙ 12535

Wörterbuch:

a = 1
b = 2
c = 3
d = 4
ab = 5
ba = 6
abc = 7
ca = 8
aba = 9
ac = 10

letztes W.	aktuelles W.	Eintrag	Code Ausgabe
␣	a		
␣a	b	ab = 5	1
b	a	ba = 6	2
a	b		
ab	c	abc = 7	5
c	a	ca = 8	3
a	b		
ab	a	aba = 9	5
a	c	ac = 10	1
c	EOF		3

5.3.2 Decodierung

Nachricht	Decodiert	Eintrag
7 5 10 11 12	7	–
7 5 10 11 12	5	10 → 75
7 5 75 11 12	75	11 → 57
7 5 75 57 12	57	12 → 755
7 5 75 57 755	755	–

Decoded: 757557755

6 VERSCHLÜSSELUNG

6.1 SYMMETRISCHE VERFAHREN

6.1.1 Substitionsverfahren/Caesar Chiffre

Klartext: bald ist weihnachten

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 mwx aimlreglxir

Zwar sieht der Chiffretext unlesbar aus, jedoch sind die statistischen Eigenschaften von Klar- und Chiffriertext unverändert. Wenn wir die Sprache kennen und die Probe gross genug ist, kann der Schlüssel leicht ermittelt werden. Ausserdem ist die Schlüsselanzahl recht übersichtlich, es braucht nicht mal einen Computer, um alle auszuprobieren. Sowohl Sprache als auch Textlänge können einen entscheidenden Einfluss haben auf den Erfolg der Analyse.

6.1.2 Transpositionsverfahren

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

Erstellen einer Tabelle zeilenweise

Auslesen spaltenweise

Chiffretext:
DTILNHGIECAMLLEHHLITAW
OWLRDUOEOEFEBRRHIERE

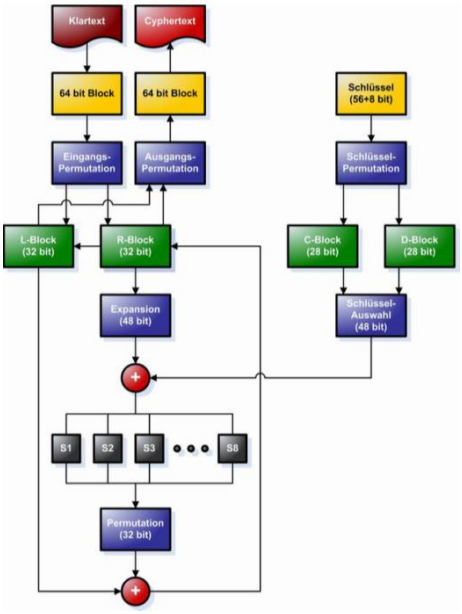
D	I	E	W	O	R
T	E	H	O	E	R
I	C	H	W	O	H
L	A	L	L	E	I
N	M	I	R	F	E
H	L	T	D	E	R
G	L	A	U	B	E

Hier sind Permutationen der Spalten möglich!

6.1.3 Vigenère-Chiffre

In diesem Beispiel soll «beispielklartext» mit dem Schlüssel «APFELSTRUDEL» verschlüsselt werden. Der Chiffretext ist «BTNWAAXCEOECTTCX». Man sucht den Klartextbuchstaben links in der 1. Spalte und geht in dieser Zeile so weit nach rechts, bis man in der obersten Zeile den Buchstaben des Schlüssels gefunden hat. Nun kann man dort den Chiffre-Buchstaben ablesen.

6.1.4 Data Encryption Standard (DES) / Advanced Encryption Standard (AES)



DES wurde als offizieller Standard für die US-Regierung im Jahr 1977 bestätigt und wird seither international vielfach eingesetzt. Die Beteiligung der NSA am Design des Algorithmus hat 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.

AES ist der Nachfolger von DES und gilt derzeit als sicher.

6.2 Asymmetrische Verfahren

6.2.1 RSA

6.2.1.1 Public und private Key bestimmen

1. Zwei beliebige Primzahlen, p und q bestimmen
2. $n = p \cdot q$ bestimmen
3. $\varphi(n) = (p-1) * (q-1)$ bestimmen
4. Eine beliebige Zahl e bestimmen, $1 < e < \varphi(n)$ & $ggT\big(e, \varphi(n)\big) = 1$
5. Multiplikatives Inverses b von a bestimmen: $d \cdot e \equiv 1 \; mod \; \varphi(n)$
6. Öffentliche Schlüssel e und n veröffentlichen
7. Der private Schlüssel d bleibt geheim

6.2.1.2 Verschlüsselung

1. $(\text{Buchstabe mit Buchstabentabelle in Zahl übersetzt})^e \; mod \; n$

6.2.1.3 Entschlüsselung

1. $(Ergebnis \; der \; Verschlüsselung)^d \; mod \; n$
2. Ergebnis der Entschlüsselung mithilfe der Buchstabentablle wieder in Text umwandeln

6.2.1.4 Multiplikatives Inverses

Wenn $a \cdot b \equiv 1 \; mod \; q$, dann ist b das multiplikative Inverse von a
Gibt es nur, wenn $ggT(a, q) = 1$

6.2.1.5 Eulersche Phi-Funktion

Beispiel: $\mathbb{Z}_4 = \{1,2,3\}$

$\mathbb{Z}_n^* = \{x \in \mathbb{Z}_n \mid 1 \leq x \leq n, \;\; ggT(n,x) = 1 \qquad oder \qquad x \; hat \; multiplikatives \; Inverses \; in \; \mathbb{Z}_n \}$

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

Liste:

$\varphi(1) = 1$
$\varphi(2) = 1$
$\varphi(3) = 2$
$\varphi(4) = 2$
$\varphi(5) = 4$
$\varphi(6) = 2$
$\varphi(7) = 6$
$\varphi(8) = 4$
$\varphi(9) = 6$
$\varphi(10) = 4$
$\varphi(11) = 10$
$\varphi(12) = 4$

$\varphi(n * m) = \varphi(n) \cdot \varphi(m)$ gilt aber nur, wenn $ggT(n,m) = 1$
$\varphi\big(n^k\big) = n^k - n^{k-1}$
$\varphi(27) = \varphi(3^3) = 3^3 - 3^2 = 18$

6.2.1.6 *Erweiterter Euklidischer Algorithmus*

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

Tabelle:

Ablauf	x	y	q	r	u	s	v	t
Initialisieren	a	b	$x\ div\ y$	$x - q * y$	1	0	0	1
Wiederholen	y_{-1}	r_{-1}	$x\ div\ y$	$x - q * y$	s_{-1}	$u_{-1} - q_{-1} * s_{-1}$	t_{-1}	$v_{-1} - q_{-1} * t_{-1}$

Beispiel:

x	y	q	r	u	s	v	t
99	79	1	20	1	0	0	1
79	20	3	19	0	1	1	-1
20	19	1	1	1	-3	-1	4
19	1 (ggT)	19	0	-3	4	4	-5 (mod 99 = Multiplikat. Inverses)

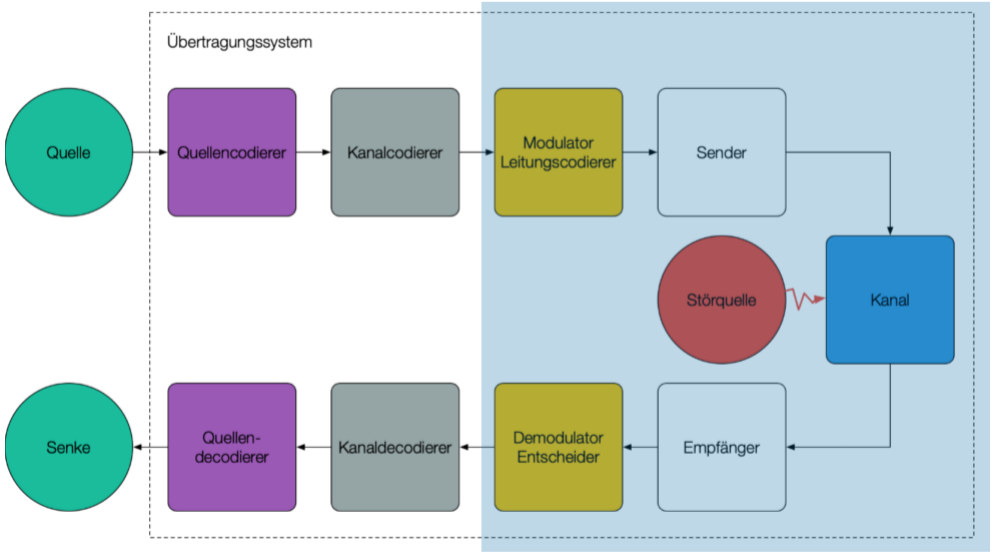
6.2.2 Probleme Handhabung asymmetrisches Verfahren

- Performance: Wie Sie sich nach den bisher gelösten Aufgaben vorstellen können, sind asymmetrische Algorithmen sehr langsam (ca. 10'000x langsamer als symmetrische Verfahren).
- 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 Empfängern muss die Nachricht mit dem öffentlichen Schlüssel jedes einzelnen Empfängers verschlüsselt werden.
- Man-In-The-Middle: Wenn es ein Angreifer schafft, seinen öffentlichen Schlüssel als den des eigentlichen Empfängers anzupreisen, kann er mit seinem privaten Schlüssel die Nachricht entschlüsseln. Um nicht aufzufallen, kann er zudem die Nachricht weiter mit dem öffentlichen Schlüssel des eigentlichen Empfängers wieder verschlüsseln und dann die Nachricht weiterschicken. Um das zu verhindern, muss die Authentizität des öffentlichen Schlüssels gewährleistet sein.

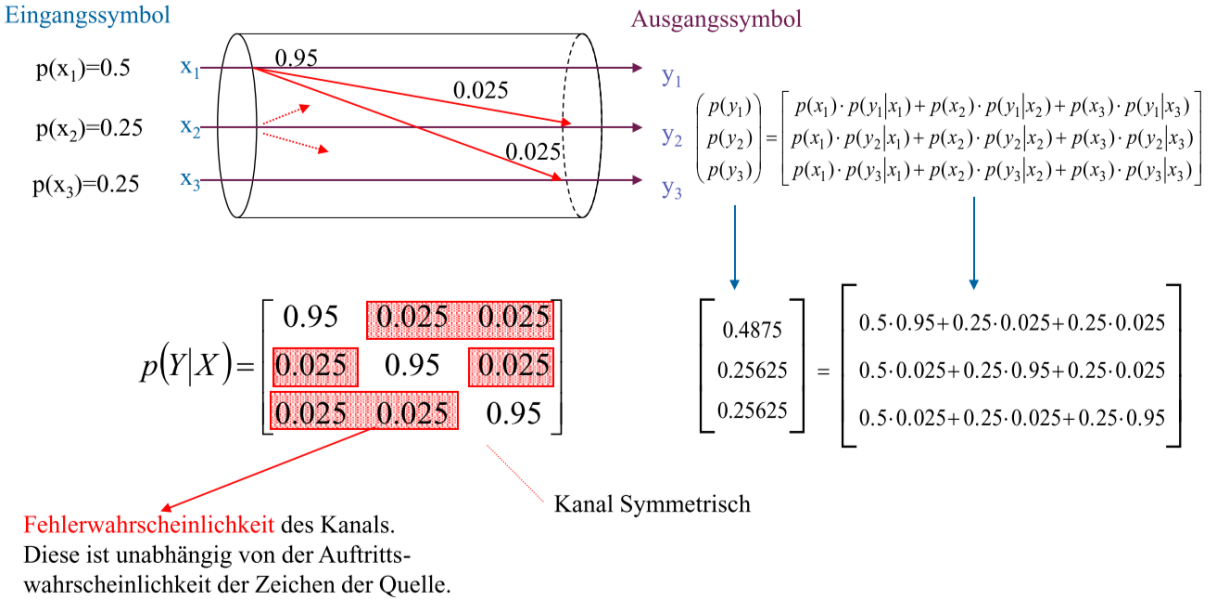
6.2.3 Kombination Komprimierung/Verschlüsselung

Da eine gute Verschlüsselung ihre Daten pseudozufällig macht, hat eine anschliessende Datenkompression keine Wirkung mehr - die Zeichen werden gleich wahrscheinlich. Sie müssen daher die Daten zuerst komprimieren und erst anschliessend verschlüsseln. Zudem maximiert die Komprimierung die Entropie der Nachricht und reduziert so etwas die Angriffsfläche für statistische Angriffe auf die Verschlüsselung.

7 KANALMODELL



7.1 KANALMATRIX



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

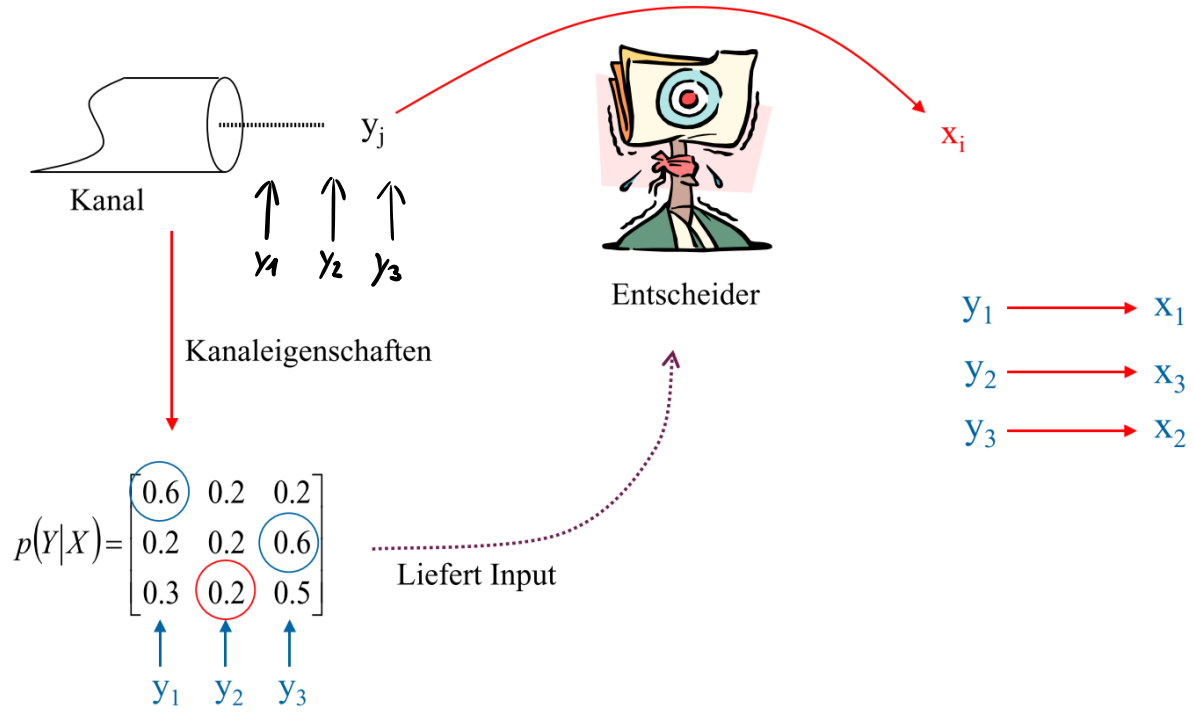
$$\begin{pmatrix}p(y_1)\\ p(y_2)\\ p(y_3)\end{pmatrix}=\begin{bmatrix}p(x_1)\cdot p(y_1|x_1)+p(x_2)\cdot p(y_1|x_2)+p(x_3)\cdot p(y_1|x_3)\\ p(x_1)\cdot p(y_2|x_1)+p(x_2)\cdot p(y_2|x_2)+p(x_3)\cdot p(y_2|x_3)\\ p(x_1)\cdot p(y_3|x_1)+p(x_2)\cdot p(y_3|x_2)+p(x_3)\cdot p(y_3|x_3)\end{bmatrix}$$

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

T=Transinformation

Nicht gestört (Einheitsmatrix, T=1): $\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}$ Vollständig gestört (T=0): $\begin{bmatrix}1/3 & 1/3 & 1/3\\ 1/3 & 1/3 & 1/3\\ 1/3 & 1/3 & 1/3\end{bmatrix}$

7.2 MAXIMUM-LIKELIHOOD-VERFAHREN



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

Pro Spalte den höchsten Wert einkreisen, wenn alle gleich sind das Zeichen nehmen, das noch nicht abgedeckt wurde. Der Entscheider nimmt an, wenn er ein y_i empfangen hat, dass das x_i mit der höchsten Wahrscheinlichkeit gesendet wurde.

Berechnen Sie die Fehlerwahrscheinlichkeit, wenn die Eingangszeichen gemäss folgender Häufigkeitsanalyse auftreten:

$$X_1 : 550$$
$$X_2 : 1200$$
$$X_3 : 3000$$

Total: $550 + 1200 + 3000 = 4750$

$x_1 = \frac{550}{4750} = 0{,}116$

$x_2 = \frac{1200}{4750} = 0{,}253$

$x_3 = \frac{3000}{4750} = 0{,}632$

$P(\text{keine Fehler}) = 0{,}5 \cdot p(x_1) + 0{,}7 \cdot p(x_2) + 0{,}6 \cdot p(x_3) =$

$0{,}5 \cdot 0{,}116 + 0{,}7 \cdot 0{,}253 + 0{,}6 \cdot 0{,}632 =$

$0{,}625$

$P(\text{Fehler}) = 1 - 0{,}625 = 0{,}375$

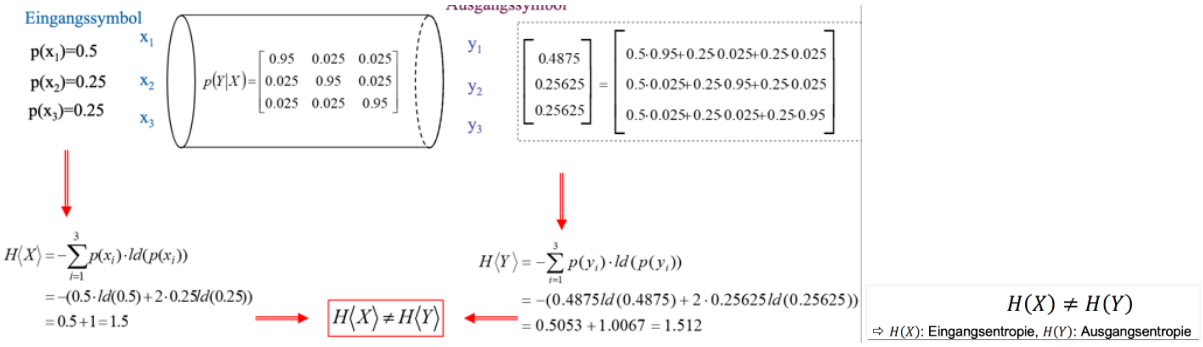
3. Kann ein anderer Entscheider zu einem besseren Ergebnis, d.h. zu einer geringeren Fehlerwahrscheinlichkeit führen?

Ja $y_1 \to x_3$, $y_2 \to x_1$, $y_3 \to x_3$

$1 - (0.5 \cdot 0{,}116 + 0{,}4 \cdot 0{,}632 + 0{,}6 \cdot 0{,}632) = 0{,}31$, jedoch x_2 nicht abgedeckt

7.3 TRANSINFORMATION

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



- Transinformation beschreibt den maximalen, fehlerfreien Informationsfluss über einen Kanal. (Es gibt noch keine Aussage wie dies zu erreichen ist)
- Verändert sich die Entropie der Quelle, verändert sich auch die Transinformation.
- Nimmt die Fehlerwahrscheinlichkeit zu, so verringert sich die Transinformation.
- Transinformation wird durch die Quelle bestimmt
- 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.
- $T = H(X) - H(X|Y) = [bit/Zeichen]$
- $T = H(Y) - H(Y|X) = [bit/Zeichen]$
- $R_{Max} = T \cdot [Zeichen/s]$ Beispiel: $R_{Max} = 0.47 \cdot 1kbit/s = 0.47 \cdot 1000 = 470bit/s$

7.4 Verbundentropie

Das paarweise Auftreten aller Möglichkeiten am Kanaleingang und Kanalausgang.

$$\boldsymbol{H(X,Y)} = -\sum_{k=1}^{N}\sum_{i=1}^{N} p(x_k, y_i) * \log_2\big(p(x_k, y_i)\big)$$

7.5 Äquivokation/Verlust/Rückschlussentropie

Beschreibt die Ungewissheit über ein gesendetes Zeichen bei bekannten Empfangszeichen.

$$\boldsymbol{H(X|Y)} = -\sum_{k=1}^{N}\sum_{i=1}^{N} p(y_i) * p(x_k|y_i) * \log_2\big(p(x_k|y_i)\big)$$

$\mathrm{H(X|Y)} = 0$ bei fehlerfreiem Kanal

7.6 Irrelevanz/Rauschen/Streuentropie

Beschreibt die Ungewissheit der empfangenen Zeichen bei bekanntem Sendezeichen.

Schriftart

$$\boldsymbol{H(Y|X)} = -\sum_{k=1}^{N}\sum_{i=1}^{N} p(x_k) * p(y_i|x_k) * \log_2\big(p(y_i|x_k)\big)$$

Beispiel eines verlustfreien und rauschbehafteten Kanals: $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.4 & 0.6 \end{bmatrix}$

Die Summe der Zeilen muss 1 sein. Die Matrix ist asymmetrisch, weil x_3 nicht existiert.

8 Blockcodes

8.1 Begriffe und Formeln

- Anzahl Worte = 2 bei Binärcode
- n = Anzahl Codestellen
- m = Anzahl Nachrichtenstellen
- k = Anzahl Kontrollstellen, wenn man eine Prüfmatrix hat, kann k aus der Einheitsmatrix am Ende der Prüfmatrix abgelesen werden, z.B. für $k = 4$:

11 Stellen

$$\left[\begin{array}{ccccccccccc|cccc} 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right]$$

  Ausserdem ist kann k aus dem Generatorpolynom abgelesen werden, es ist die höchste Potenz.
- $n = m + k$
- $\mathrm{n} = 2^k - 1$
- Gültige Codeworte: 2^m
- Mögliche CW: 2^{m+k}

8.2 HAMMINGCODE

Codewort
00000000
11000000
10001100
01010000
01010101
10000110
11111111

- Hammingdistanz h: beschreibt den minimalen Abstand zwischen zwei gültigen Codeworten im gesamten Coderaum. Wenn man eine Liste von Codeworten hat, kann man auch schauen, in wie vielen Bits sich die Codeworte unterscheiden, z.B. bei $h = 2$:
- h ist die Anzahl Prüfvektoren, die addiert werden müssen, um den Nullvektor zu erhalten
- meistens $h = k$
- Anzahl erkennbare Fehler $e^* = \mathrm{h} - 1$
- Anzahl korrigierbare Fehler bei geradem h: $e = \frac{h-2}{2}$
- Anzahl korrigierbare Fehler bei ungeradem h: $e = \frac{h-1}{2}$
- Treten mehr Fehler auf als korrigierbar sind, so wird entweder falsch korrigiert oder der Fehler wird nicht erkannt.

8.2.1 Kontrollstellen eines Codeworts berechnen

Es wird die Matrix aus 8.1 Begriffe und Formeln verwendet, CW = 10110000010 und h = 4
Aus dem Codewort lesen wir ab, dass x_1, x_3, x_4, x_{10} relevant sind (alle Einsen).
Aus der Matrix lesen wir folgende Formeln ab (alle Einsen der Zeilen von oben nach unten):

$x_{12} = x_1 + x_2 + x_3 + x_4 + x_6 + x_7 + x_8 \mod 2 = 1$

$x_{13} = x_1 + x_2 + x_3 + x_5 + x_6 + x_9 + x_{10} \mod 2 = 1$

$x_{14} = x_1 + x_2 + x_4 + x_5 + x_7 + x_9 + x_{11} \mod 2 = 0$

$x_{15} = x_1 + x_3 + x_4 + x_5 + x_8 + x_{10} + x_{11} \mod 2 = 0$

Die Kontrollstellen sind somit 1100 und das komplette Codewort 10110000010 1100

8.2.2 Fehlersyndrome

Fehlersyndrome können aus der Prüfmatrix abgelesen werden. Es sind jeweils die Spalten mit dem Fehler. Bei mehreren Fehlern müssen die Spalten addiert und mod 2 gerechnet werden.

Beispiel aus der Matrix in 8.1 Begriffe und Formeln, Fehler bei x_2: $\begin{pmatrix}1\\1\\1\\0\end{pmatrix}$

oder bei x_2 und x_{11}: $\begin{pmatrix}1\\1\\1\\0\end{pmatrix} + \begin{pmatrix}0\\0\\1\\1\end{pmatrix} mod\ 2 = \begin{pmatrix}1\\1\\0\\1\end{pmatrix}$

8.2.3 Mehrfachaddition zur Bestimmung aller gültigen Codeworte

Benötigt: Generatorpolynom binär & Anzahl Nachrichtenstellen m (siehe 8.1 Begriffe und Formeln)
Zuerst müssen alle möglichen Nachrichten erstellt werden, es gibt 2^m Nachrichten. Bei $m = 2$ wären es z.B. $00, 01, 10$ und 11.
Danach wird für jede mögliche Nachricht das Generatorpolynom mehrfach addiert, bis vor dem vertikalen Strich nur noch Nullen stehen. Beispiel:

0101 ← mögliche Nachricht
101 | 1 ← Generatorpolynom
000 | 100 → 0101|100

Das erste mögliche Codewort sind immer alles Nullen und das Letzte alle Einsen.

0110 | 101 | 1
011 | 10 | 1
11
01 | 1 | 01
011
0 | 001 → 0110|001

8.2.4 Mehrfachaddition zur Bestimmung der Prüfmatrix

Benötigt: Generatorpolynom binär & Anzahl Nachrichtenstellen m (siehe 8.1 Begriffe und Formeln)

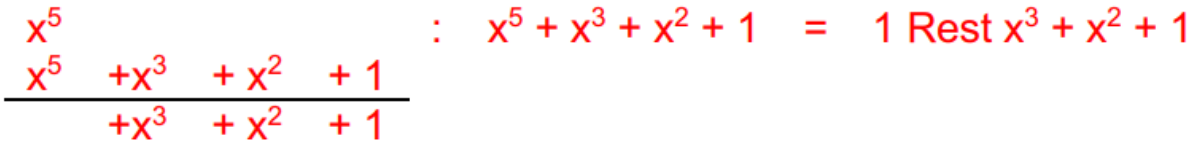
Zuerst müssen Codewörter erstellt werden, bei dem jeweils genau eine Nachrichtenstelle 1 ist, z.B. $0001, 0010, 0100$ und 1000.

Danach wird für jedes Codewort das Generatorpolynom mehrfach addiert, bis vor dem vertikalen Strich nur noch Nullen stehen. Beispiel:

Das Ergebnis ist dann eine Prüfmatrix mit k Zeilen und n Spalten. Am Ende befindet sich die Einheitsmatrix. Die Additionsbeispiel ist in der Matrix die zweite Spalte, weil das zweite Bit auf 1 gesetzt wurde:

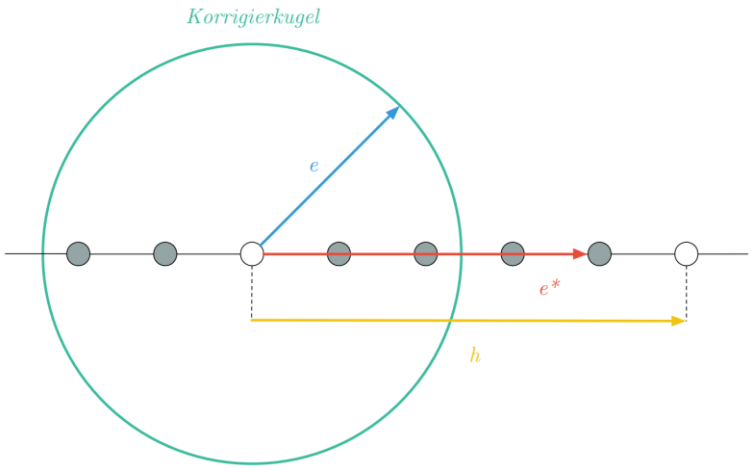
8.2.5 Polynomdivision zur Ermittlung eines Fehlersyndroms an Stelle

Benötigt: Stelle (hier: x^5) & Generatorpolynom (hier: $x^5 + x^3 + x^2 + 1$)



8.3 KORRIGIERKUGEL

Das Zentrum der Korrigierkugel liegt auf jedem gültigen Codewort (weiss) und schliesst alle umliegenden, ungültigen Codeworte (grau), die sicher korrigiert werden können, ein. Die Anzahl der sicher korrigierbaren Fehler e kann deshalb als Radius der Korrigierkugel betrachtet werden. Die Hammingdistanz h ist die kürzeste Distanz zwischen zwei gültigen Codeworten unter Betrachtung aller Paare von gültigen Codeworten im gesamten Coderaum. Die Anzahl der sicher erkennbaren Fehler e^* ist deshalb $h - 1$.



8.4 Dichtgepackter Code

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

Der Coderaum ist dichtgepackt wenn: $2^m * \sum_{w=0}^{e} \frac{n!}{w!(n-w)!} = 2^n$

Beispiel für $n = 15,\ m = 10, k = 5, e = 1$:

$$2^{10} * (\frac{15!}{0!\,(15-0)!} + \frac{15!}{1!\,(15-1)!}) = 2^{15}$$

$$2^{10} * (1 + 15) \neq 2^{15}, also\ nicht\ dichgepackt$$

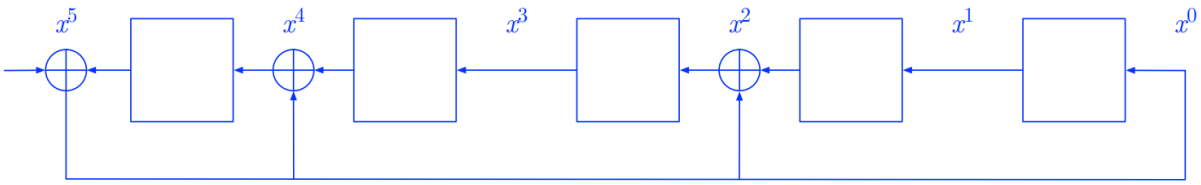
8.5 Zyklische Abramson Codes / CRC Codes

Hammingdistanz $h = 4$

Wird gebildet durch Multiplikation des primitiven Polynoms mit $(1 + x)$

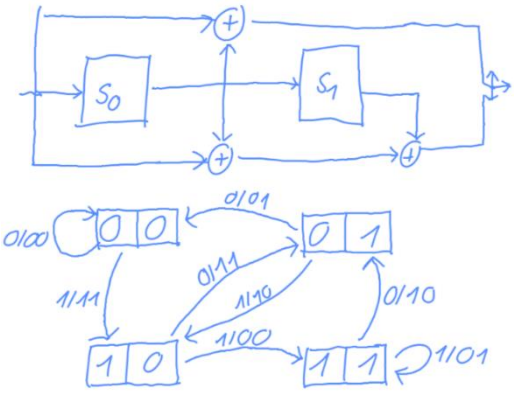
$n = 2^{k-1} - 1$

Rückgekoppeltes Schiebregister für $1 + x^2 + x^4 + x^5$:



Um ein reduzibles Generatorpolynom zu reduzieren, teilt man es durch $(1 + x)$

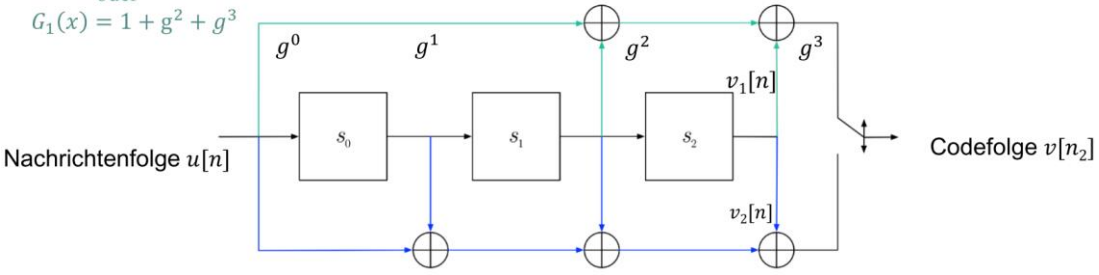
9 Faltungscodes



$u[n]$	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

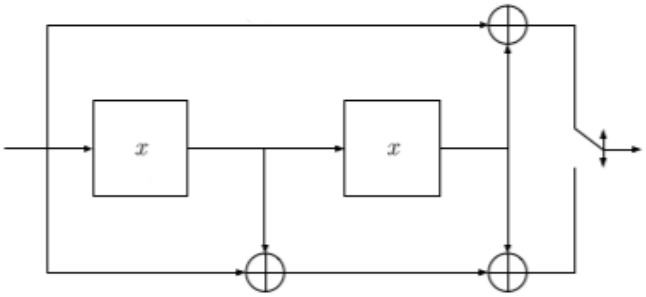
$\{g_1\} = \{1\ 0\ 1\ 1\}$
oder
$G_1(x) = 1 + g^2 + g^3$

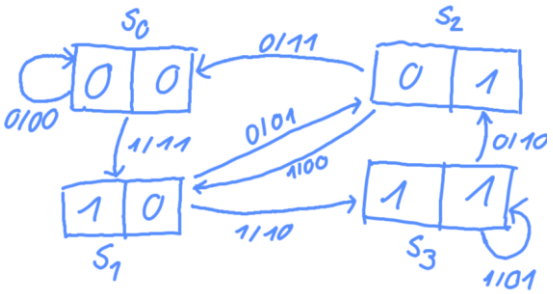


$\{G_2\} = \{1\ 1\ 1\ 1\}$
oder
$G_2(x) = 1 + g + g^2 + g^3$

Beispiel für Codierung von 101
Ausgabe ist: 11 01 00 10 11 11
In der Eingabe sind 3 Tailbits enthalten (6 in der Ausgabe).
Die Anzahl entspricht der höchsten Potenz des Generatorpolynoms.
Für die Impulsantwort gibt man dem Faltungscodierer eine 1 und dann nur noch Nullen. Die Impulsantwort ist dann das, was rauskommt dabei.
Coderate = Eingangsbits/Ausgangsbits

9.1 Zustandsdiagramm

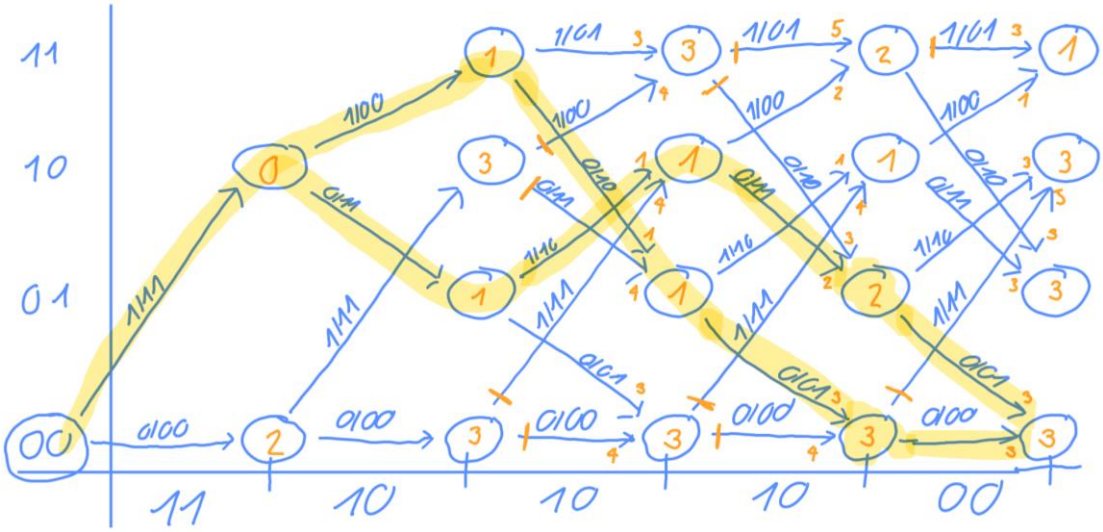




- Die 4 Boxen im Zustandsdiagramm sind jeweils alle möglichen Kombinationen der x in der Encoder-Schaltung
- Die Zahl vor dem / ist die Zahl, welche man vorne in die Schaltung reingibt
- Die Zahl nach dem / ist die Zahl, welche hinten an der Schaltung herauskommt mit Einbezug der reingegebenen Zahl und der aktuellen Belegung der x

9.2 Decodierung mithilfe des Netzdiagramms

Verw. Encoder-Schaltung und Zustandsdiagramm:
Empfangene Bits: 11 10 10 10 00
Netzdiagramm:

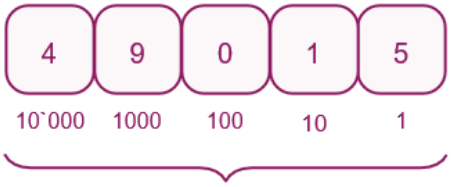


- Die orangen Zahlen sind jeweils die gemachten Fehler
- Wenn zwei Pfeile auf eine Kugel zeigen, wird die tiefere Zahl gewählt und der andere Weg geblockt
- Es ist definiert, dass Start und Ende bei 00 liegen
- Wenn alle orangen Einträge gemacht sind, geht man vom Ende aus und sucht den Weg mit den geringsten Kosten.
- Das gesendete Codewort (Lösung) sind dann die Zahlen vor dem / auf dem Weg abzüglich der Tailbits, in diesem Fall also 110 oder 101
- Talbots sind immer die Anzahl Quadrate im Schieberegister

10 BINÄRZAHLEN

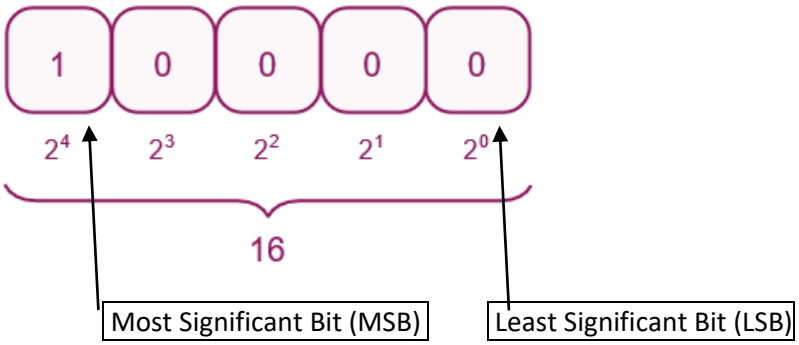
Computer arbeiten mit dem Binärsystem, weil technisch zwei Zustände einfach unterscheidbar sind.

Dezimalsystem:



4 * 10 000 + 9 * 1000`+ 0 * 100 +1 * 10 + 5 * 1

Binärsystem:



Nibble: 4 Bit
Byte: 8 Bit

Notation einzelner Bits von 1010: $b_0 = 0,\ b_3 = 1$ oder $b[0] = 0, b[3] = 1$

Notation von Unterbereichen von 1010: $b_{3..1} = 101$ oder $b[3..1] = 101$

Präfixe:

	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

10.1 HEXADEZIMALSYSTEM

Ein Nibble (4 Bit) können jeweils mit einer Hexadezimalstelle beschrieben werden. Daher ist es sehr gut geeignet, um Binärzahlen zu verkürzen.

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

10.1.1 Grösste darstellbare Zahl

Die grösste Hex-Zahl mit n Bit hat an der Most Significant Stelle eine $F, 1, 3, oder\ 7$.

$8_d\ Bit\ =\ 1111\ 1111_b = FF_h$
$9_d\ Bit\ =\ 1\ 1111\ 1111_b = 1FF_h$
$10_d\ Bit\ =\ 11\ 1111\ 1111_b = 3FF_h$
$11_d\ Bit\ =\ 111\ 1111\ 1111_b = 7FF_h$
$12_d\ Bit\ =\ 1111\ 1111\ 1111_b = FFF_h$
usw.

10.2 DARSTELLUNG VON ZAHLEN

$1011_b\ (binär) = 11_d\ (dezimal) = B_h\ (hexadezimal)$

10.3 Berechnungen

10.3.1 Umwandlung Dezimal – Binär

Man teilt jeweils durch 2. Wenn das Ergebnis eine ganze Zahl ist, wird eine 0 geschrieben. Wenn es eine .5 Zahl ist, wird eine 1 geschrieben und mit der Abrundung weitergerechnet. Abgebrochen wird bei 0.5. Die Binärzahl wird dann von unten nach oben abgelesen. Beispiel:

Dezimalzahl	***Operation***	***Ergebnis***	***Bit***
22	/2	11	0
11	/2	5.5	1
5	/2	2.5	1
2	/2	1	0
1	/2	0.5	1

22 in binär: 10110

10.3.2 Addition

Bei der Addition von Binärzahlen wird von rechts begonnen. Es wird jeweils das Bit des ersten Summanden, des zweiten Summanden und das Carry Bit addiert und dann $mod\ 2$ gerechnet. Das Carry Bit wird auf 1 gesetzt, wenn das vorherige Ergebnis mehr als 1 war vor $mod\ 2$. Beispiel:

Binär						*Dezimal*
1. Summand		1	1	0	1	13
2. Summand		1	0	0	1	9
Carry-Bit	1	0	0	1	-	=
Ergebnis	**1**	**0**	**1**	**1**	**0**	**22**

10.3.3 Subtraktion

Bei der Subtraktion von Binärzahlen wird von rechts begonnen. Es wird jeweils das Bit des Subtrahenden und Carry-Bits vom Minuenden abgezogen und dann $mod\ 2$ gerechnet. Das Carry Bit wird auf 1 gesetzt, wenn das Ergebnis kleiner als 0 war, vor $mod\ 2$. Beispiel:

Binär					***Dezimal***
Minuend	1	1	0	1	13
Subtrahend	1	0	0	1	9
Carry-Bit	0	0	0	-	=
Ergebnis	**0**	**1**	**0**	**0**	**4**

10.3.4 Multiplikation

Bei Präfixschreibweise: $128K * 64M = 2^7 * 2^{10} * 2^6 * 2^{20} = 2^{7+10+6+20} = 2^{43} = 2^3 * 2^{40} = 8T$

Bei der binären Multiplikation gelten die folgenden Regeln:

$0 * 0 = 0$
$0 * 1 = 0$
$1 * 0 = 0$
$1 * 1 = 1$

Danach wird jeweils der ganze erste Multiplikator mit einer Stelle (angefangen von rechts) des zweiten Multiplikators multipliziert. Nach jeder Stelle wird eins nach links eingerückt. Danach werden die Zahlen normal addiert.

```
          1101 · 1001
-------------------
              1101
+             0000
+            0000
+           1101
-------------------
Übertrag  0010000
Produkt   1110101
```

10.3.5 Division

Bei Präfixschreibweise: $\frac{32M}{64K} = \frac{2^5 * 2^{20}}{2^6 * 2^{10}} = \frac{2^{25}}{2^{16}} = 2^{25-16} = 2^9 = 512$

Bei der binären Division gelten folgende Regeln (Division durch 0 nicht definiert):

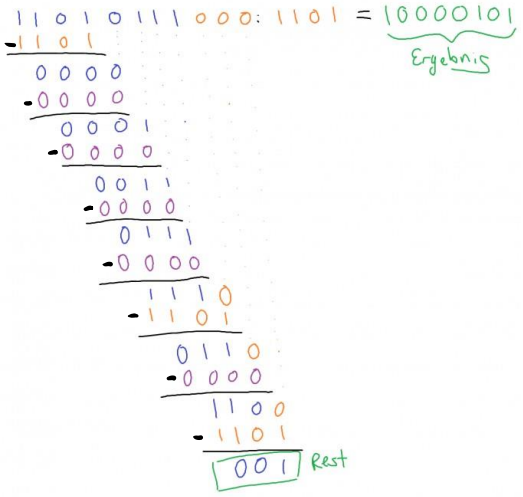
$0/1 = 0$
$1/1 = 1$

Danach probiert man zuerst, die Stelle ganz links der zu teilenden Zahl durch den ganzen Teiler zu teilen. Solange es nicht geht, nimmt man noch eine Stelle eins weiter rechts der zu teilenden Zahl dazu. Dann schreibt man beim Ergebnis auf, wie oft der Teiler in die Stellen reinpasst (beim ersten Durchlauf immer 1).

Nun subtrahiert man den Teiler von den Stellen und schreibt noch die nächste Stelle vom Dividenden runter ganz rechts vom Ergebnis. Nun gibt es zwei Möglichkeiten:

- Der Teiler passt ins erweiterte Subtraktions-Ergebnis. Es kommt eine 1 ins Ergebnis und der Teiler wird erneut subtrahiert.
- Der Teiler passt nicht ins erweiterte Subtraktions-Ergebnis. Es kommt eine 0 ins Ergebnis und 0 subtrahiert.

Das wird wiederholt, bis ganz unten eine Zahl kleiner als der Teiler befindet (Rest), es kann auch 0 sein.



10.4 Signed Integers

Dabei wird das MSB als Vorzeichen interpretiert, 0 ist positiv und 1 ist negativ.
Der Wertebereich reicht von -2^{n-1} bis $2^{n-1} - 1$, wobei n die Anzahl Bits ist.
Die negativen Zahlen werden dabei als Zweierkomplement gespeichert.
Der Wert einer vorzeichenbehafteten, negativen Binärzahl mit n Bit entspricht der Differenz aus vorzeichenlosem Wert und 2^n.

Schreibweise Zweierkomplement: $N(b) = -b$

10.4.1 Umwandlung negative Dezimalzahl ins Zweierkomplement (Inversionsverfahren)

Beispiel mit Zahl -5:

1. In die positive Binärzahl umwandeln (muss 0 als MSB haben): 0101
2. Alle Bits invertieren: 1010
3. Eins addieren: 1011

10.4.2 Umwandlung Zweierkomplement in negative Dezimalzahl (Inversionsverfahren)

Bespiel mit 1011:

1. Alle Bits invertieren: 0100
2. Eins addieren: 0101
3. Zahl ablesen und minus davorsetzen: -5

11 Boolsche Logik

11.1 Arität von Funktionen

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: $f() = 1$

11.2 Unäre Funktionen

Es gibt genau 4 unäre Funktionen, wobei nur die Identität (id) und Negation (not) von der Eingabe abhängen.
$not(x) = \neg x = \bar{x}$

x	$F(x)$	id(x)	not(x)	$T(x)$
0	0	0	1	1
1	0	1	0	1

11.3 Binäre Funktionen

Es gibt genau 16 binäre Funktionen.

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

NAND sind die Grundbausteine der Computertechnik, da sie technisch leicht als Transistoren zu realisieren sind. Alle Basisoperationen können damit dargestellt werden.

De Morgan Law: $\overline{x \vee y} = \bar{x} \wedge \bar{y}$

11.4 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. $x_0 \vee \overline{x_1} \vee \overline{x_2} \vee x_3 \vee x_4$

11.5 Disjunktive Normalformen

Disjunktive Normalform (DNF): Disjunktion von Konjunktionstermen, z.B. $A \vee B \vee C$
Kanonische disjunktive Normalform (KDNF): Disjunktion von Mintermen, z.B. $A\bar{B}C \vee AB\bar{C}$

Die KDNF kann schnell aus einer Wahrheitstabelle abgeleitet werden:

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

Die KDNF $\overline{x}\overline{y} \vee \bar{x}y \vee x\bar{y}$ kann zu einer DNF vereinfacht werden:

$\bar{x}(y \vee \bar{y}) \vee x\bar{y} \quad = \quad \bar{x} \vee x\bar{y} \quad = \quad \bar{x} \vee \bar{y}$

12 ASCII

Alle Zeichen eines Texts stammen aus einer endlichen Menge Z, dem Zeichensatz. Jedem Zeichen kann eindeutig eine natürliche Zahl zugeordnet werden (Encoding). Sender und Empfänger müssen das Encoding miteinander vereinbaren, damit sie mit 0 und 1 Text übertragen können. Ein verbreitetes Encoding ist der «American Standard Code for Information Interchange» (ASCII).

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	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

Bekannte, nicht obsolete Steuerzeichen:

\0	Null	Ende des Texts	\n	Line Feed	Cursor eine Zeile nach unten
\b	Backspace		\r	Carriage Return	Cursor an Anfang der Zeile
\t	Tabulator				

13 FIXKOMMAZAHLEN

Bei der Fixkommazahl wird an einer beliebigen Stelle einer Binärzahl ein Komma gesetzt. Viele Dezimalkommazahlen können nicht exakt dargestellt werden.

Beispiel für 12.875:

Stelle				a				$-a$
Zweierpotenz	2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}
Dezimal	8	4	2	1	.	0.5	0.25	0.125
Binäre Fixkommazahl	1	1	0	0	.	1	1	1

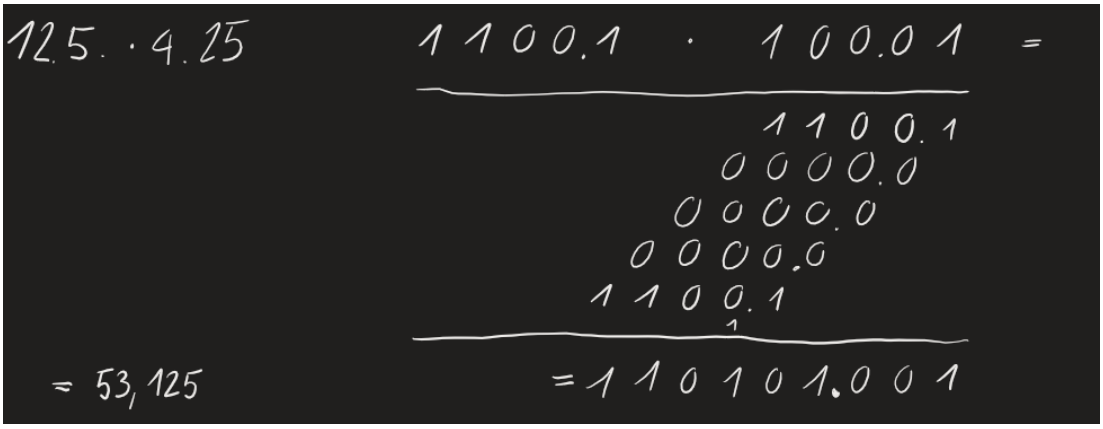
Bezogen auf unsigned Fixkommazahl mit 3 Bit vor dem Komma und 5 Bit nach dem Komma:

- Kleinste Zahl > 0: 2^{-5}
- Grösste Zahl: $2^3 - 2^{-5}$

Gleich wie bei ganzen Binärzahlen sind: Addition, Subtraktion, Zweierkomplement und Shift

13.1 MULTIPLIKATION

Bei der Multiplikation verschiebt sich das Komma, das Ergebnis hat so viele Nachkommastellen wie die beiden Multiplikatoren zusammen.



Bei der Division wird ebenfalls das Komma verschoben.

14 GLEITKOMMAZAHLEN

Gleitkommazahlen werden immer als normalisierte Zahlen geschrieben, das heisst mit genau einer 1 vor dem Komma. Beispiel: $1.1 * 2^{-1}$

Die führende 1 kann beim Codieren weggelassen werden, sie ist ein Hidden Bit. Somit hat man eine Stelle mehr für die Präzision.

Wert einer Float-Zahl an diesem Beispiel: $z = \overbrace{1}^{s} | \overbrace{1000\ 0100}^{k} | \overbrace{010\ 0100\ 0000\ 0000\ 0000\ 0000}^{m}$

$$W(z) = (-1)^s \cdot (1 + m) \cdot 2^{k-b} =$$
$$(-1)^1 \cdot (1 + 2^{-2} + 2^{-5}) \cdot 2^{2^7 + 2^2 - 127} = -1.28125 \cdot 2^5 = -41$$

Das Bias b beträgt bei 32-bit Float $2^7 - 1 = 127$ und bei 64-bit Double $2^{10} - 1 = 1023$

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)

- Grösste Float-Zahl: 0 | 1111 1110 | 111 1111 1111 1111 1111 1111
  $W(z) = (-1)^0 * (1 + (2^{23} - 1) * 2^{-23}) * 2^{2^8-2-127} = (2 - 2^{-23}) * 2^{127} \approx 3.4 * 10^{38}$
- Kleinste Float-Zahl > 0: 0 | 0000 0001 | 000 0000 0000 0000 0000 0000
  $W(z) = (-1)^0 * (1 + 0) * 2^{1-127} = 2^{-126} \approx 1.175 * 10^{-38}$

14.1 Dezimalzahl als Float darstellen

Beispiel: 5.8 als Single-Float

1. Umwandlung der Zahl vor dem Komma zur Binärzahl: $5 = 101$
2. Umwandlung der Zahl hinter dem Komma zur Binärzahl:

2^{-1}	$0.8 - 0.5 = 0.3$	1
2^{-2}	$0.3 - 0.25 = 0.05$	1
2^{-3}	$0.05 - 0.125$	0
2^{-4}	$0.05 - 0.0625$	0
2^{-5}	$0.05 - 0.03125 = 0.01875$	1
2^{-6}	$0.01875 - 0.015625 = 0.003125$	1
2^{-7}	$0.003125 - 0.0078125$	0
2^{-8}	$0.003125 - 0.00390625$	0

3. Nun fügen wir die beiden Binärzahlen zu einer Fixkommazahl zusammen, ausserdem haben wir ein Muster entdeckt: $101.\overline{1100}$
4. Nun wir das Komma so weit nach links geschoben, bis genau eine 1 links vom Komma steht
   Die Anzahl Stellen nach rechts werden als Exponent von 2 malgerechnet
   $1.01\overline{1100} * 2^2$
5. Bestimmung von s mithilfe des Vorzeichens: $positiv \rightarrow s = 0$
6. Bestimmung des Exponenten k: Exponent von Schritt 4 plus 127: $2 + 127 = 129$
7. Bestimmung der Mantisse m: Die Bits nach dem Komma von Schritt 4: $01\overline{1100}$
8. Ergebnis: $0|1000\ 0001|0111\ 0011\ 0011\ 0011\ 0011\ 001$

14.2 Addition von Floats

Beispiel Addition von $2.6 + 5.8 = 8.4$

1. Dezimalzahlen im Float-Format aufschreiben
   $0|1000\ 0000|010\ 0110\ 0110\ 0110\ 0110\ 0110$
   $+\ 0|1000\ 0001|011\ 1001\ 1001\ 1001\ 1001\ 1010$
2. Das Hidden Bit (immer 1 ganz links der Mantisse einfügen)
   $0|1000\ 0000|1010\ 0110\ 0110\ 0110\ 0110\ 0110$
   $+\ 0|1000\ 0001|1011\ 1001\ 1001\ 1001\ 1001\ 1010$
3. Den grösseren Exponenten in beide Floats schreiben
   $0|1000\ 0001|1010\ 0110\ 0110\ 0110\ 0110\ 0110$
   $+\ 0|1000\ 0001|1011\ 1001\ 1001\ 1001\ 1001\ 1010$
4. Die Differenz der beiden Exponenten berechnen
   $1000\ 0001 - 1000\ 0000 = 129 - 128 = 1$
5. Beim Float mit dem ursprünglich kleineren Exponenten so viele Nullen einfügen, wie bei Schritt 4 berechnet wurden. Die gleiche Anzahl wie eingefügt wurde fliegt hinten raus.
   $0|1000\ 0001|0101\ 0011\ 0011\ 0011\ 0011\ 0011\cancel{0}$
   $+\ 0|1000\ 0001|1011\ 1001\ 1001\ 1001\ 1001\ 1010$
6. Nun wird die Mantisse normal binär addiert. Jedoch nur die vordersten 23 Bit. Falls das Carry Bit 1 ist nach der Mantisse, wird der Exponent um 1 erhöht. Das Vorzeichen bleibt.
   $0|1000\ 0001|0101\ 0011\ 0011\ 0011\ 0011\ 0011\cancel{10}$
   $+\ 0|1000\ 0001|1011\ 1001\ 1001\ 1001\ 1001\ 1010\cancel{0}$
   $0|1000\ 0010|0000\ 1100\ 1100\ 1100\ 1100\ 1100$

14.3 Runden

- Round to nearest: es gibt zwei Varianten für .1
  - Ties to even: auf gerade Zahl, z.B. 0.1 auf 0 und 1.1 auf 10.0
  - Ties away from zero: immer auf die grössere Zahl
- Round toward 0
- Round toward $+\infty$
- Round toward $-\infty$

Rundungsfehler bei kleinster Zahl grösser 1:

- der absolute Fehler ist

$$W(z) - W(z') = 1 + 2^{-23} - (1 + 2^{-24}) = 2^{-23} - 2^{-24} = 2^{-24}$$

- Der relative Fehler ist

$$e = \frac{W(z) - W(z')}{W(z')} = \frac{2^{-24}}{1 + 2^{-24}} = \frac{1}{2^{24} + 1} < \frac{1}{2^{24}}$$

Maschinen-Epsilon:

- Alle Zahlen $1 < q < z'$ runden ab auf 1
- Der absolute Fehler ist dann jeweils $\leq 2^{-24}$ und damit auch der relative Fehler
- Alle Zahlen $z' = 0\,|\,0111\ 1111\,|\,000\ 0000\ 0000\ 0000\ 0000\ 000\mathbf{0\ 1}xxx\ldots$ werden auf z aufgerundet, aber der Fehler ist kleiner als bei z'
- $\varepsilon = 2^{-24} = 2^{-p}$ ist der grösstmögliche Fehler und heisst *Maschinen-Epsilon*
- Merksatz: «Die kleinste Zahl ε, so dass $1 + \varepsilon > 1$»