Codierung und Sicherung

https://iuk.one/1010-1016

Clemens H. **Cap**
ORCID: 0000-0003-3958-6136

Department of Computer Science
University of **Rostock**
Rostock, Germany
clemens.cap@uni-rostock.de

3. 1. 2021 Vers. 3

Ziel

Zentrale Aufgaben der Codierung:

- Jedenfalls: Erkennen, ob ein Fehler aufgetreten ist
- Optional: Den Fehler eigenständig beheben.

Ist eine Fehlerbehebung durch die Codierung alleine nicht möglich
so muß der Data Link Layer das Problem auf Protokollebene beheben.

1. Code

Ziele: Wir lernen Grundbegriffe und erste Beispiele von Codes kennen und erarbeiten uns das wichtige Konzept des Hamming-Abstands.

Worte und Konkatenation

Sei A eine endliche Menge. A heißt **Alphabet**, die Elemente nennen wir **Symbole**.

Ein **Wort** über A ist eine endliche Folge von $n \in \mathbb{N}_0$ Symbolen aus A.

n heißt die **Länge** des Wortes.

Notation:

- $a_1 a_2 \ldots a_n$ für ein Wort der Länge n.
- ε für das Wort der Länge 0. Es heißt auch das **leere Wort**.
- A^* für die Menge **aller Worte** über A.
- A^+ für die Menge **aller nicht-leeren Worte** über A.

Die **Konkatenation** ist eine Funktion $\cdot : A^* \times A^* \to A^*$. Sie ist definiert durch

$a_1 a_2 \ldots a_n \ \cdot \ b_1 b_2 b_k = a_1 a_2 \ldots a_n b_1 b_2 \ldots b_k$ im generischen Fall und im Spezialfall:

$\varepsilon \cdot \varepsilon = \varepsilon$ sowie $\varepsilon \ \cdot \ a_1 \ldots a_k = a_1 a_2 \ldots a_k$ und $a_1 a_2 \ldots a_k \ \cdot \ \varepsilon = a_1 a_2 \ldots a_k$.

Monoid

Definition: Monoid
Sei M eine Menge und $\cdot : M \times M \to M$ eine Funktion auf M.

Das Paar $(M, \cdot)$ heißt ein **Monoid**, wenn folgende zwei Axiome erfüllt sind:

(1) $\forall w_1, w_2, w_3 \in A^* : (w_1 \cdot w_2) \cdot w_3 = w_1 \cdot (w_2 \cdot w_3)$ **Assoziativgesetz**

(2) $\forall w \in A^* : \varepsilon \cdot w = w \cdot \varepsilon = w$ **Neutralität von ε**

Satz: Die Menge A^* aller Worte über einer endlichen Menge A, zusammen mit der Konkatenation $\cdot$ ist ein Monoid.

Beweis: Unmittelbar aus der Definition der Konkatenation.

Man nennt A^* deshalb auch das **Wortmonoid** über A.

Code

Sei A eine endliche Menge und A^* die zugehörige Wortmenge.

Definition: Code

Ein **Code** über A ist eine Teilmenge $\mathcal{C} \subseteq A^*$ des Wortmonoids über A.

Interpretation:
Die Definition ist ziemlich unspektakulär!
Die Menge sagt nur, welche Worte als richtig angesehen werden.
Das Wichtige in der Codierungstheorie kommt erst mit

1. Zusatzannahmen über die **Art** und die **Verteilung** der Fehler
2. Anwendungen im Kontext von **Codierung und Decodierung**.

Beispiel: Paritäts-Code

Der **Code gerader (ungerader) Parität der Länge** n ist die Menge aller binären Worte der Länge n, mit *gerader (ungerader)* Anzahl von 1-Symbolen.

Beispiel: $n = 8$, gerade Parität.

Sender will 7-Bit Ascii **A** 100 0001 senden und sendet 0100 0001.
Empfänger erhalte

1. 1111 1110, meldet Paritätsfehler.
2. 1100 0011, Parität ok, Paritätsbit abgetrennt, ist Buchstabe **C**.
   Zu viele Fehler, als daß der Paritäts-Code das entdecken könnte.
3. 1100 0001, Parität falsch, Ergebnis verworfen obwohl Nutzlast ok.
   Fehler müssen nicht nur in der Nutzlast passieren.
4. 0100 0001, Parität ok, Paritätsbit abgetrennt, ist Buchstabe **A**.

Beispiel: Paritäts-Code mit multiplen Paritätsbits

Bei hoher Fehlerrate könnte man versucht sein,
7 Datenbits mit 7 Paritätsbits auf einen 14 Bit Paritäts-Code zu erweitern.
Wiederum muß das Codewort geradzahlig viele 1-er enthalten.

Wie bewerten wir das?

Beispiel: Paritäts-Code mit multiplen Paritätsbits

Bei hoher Fehlerrate könnte man versucht sein,
7 Datenbits mit 7 Paritätsbits auf einen 14 Bit Paritäts-Code zu erweitern.
Wiederum muß das Codewort geradzahlig viele 1-er enthalten.

Wie bewerten wir das?

Bewertung: Negativ, denn

- wir erkennen wiederum nur Fehler an ungeradzahlig vielen Positionen,
- wir bieten dem Fehlerteufel aber mehr Positionen als Angriffsmöglichkeit und
- wir benötigen insgesamt mehr Bits zum Übertragen.

Beispiel: Tripel-Code

Der **Tripel-Code** ist die Menge $\mathcal{T} = \{000, 111\}$.

Interpretation: Jedes Symbol wird drei Mal gesendet.

Beachte: Die Vorstellung, daß in einem Code jeweils einige Bits Nutzlast-Bits und die übrigen Bits Prüf-Bits sind, ist im allgemeinen **falsch** und zu eng.

Im Allgemeinen transportieren **alle Bits** im Code einen "Anteil" an der Nutzlast, nur bei ganz einfachen Codes sieht das aufgeteilt aus.

Genauer sieht man das erst in der Theorie der Quellen- und Kanalcodierung.

Hier wird es (nur) bei den Hamming-Codes ein wenig deutlich.

Beispiel: Tripel-Code in der Anwendung

Situation: In möglicherweise fehlerhafter Kommunikation wird der Tripel-Code $\mathcal{T} = \{000, 111\}$ eingesetzt.

Frage 1: Der Empfänger erhalte das Wort $w = 001$. Was wissen wir?

Banal: Es ist ein Fehler aufgetreten. Geht mehr?

Frage 2: Der Empfänger erhalte das Wort $w = 000$. Was wissen wir?

Wichtig: Wir wissen **nicht**, daß **kein** Fehler aufgetreten ist!

Frage 3: Können wir mehr sagen? Ggf. unter einschränkenden Annahmen?

Auflösung: Fehlererkennung und Fehlerbehebung

Antwort 1: Empfang des Wortes $w = 001$

- Es trat ein Übertragungsfehler auf, da $w \notin \mathcal{T}$.
- **Wenn** wir annehmen, daß höchstens 1-bit Fehler auftreten:
  Es wurde sicher das Wort 000 gesendet.
- **Wenn** wir annehmen, daß höchstens 2-bit Fehler auftreten:
  Es könnte neben 000 auch Wort 111 gesendet worden sein.

Antwort 2: Empfang des Wortes $w = 000$

- Wir wissen nichts. Es könnte auch 111 gesendet worden sein mit Tripelfehler.
- **Wenn** wir annehmen, daß höchstens 1-bit Fehler auftreten:
  Es wurde sicher das Wort 000 gesendet.
- **Wenn** wir annehmen, daß höchstens 2-bit Fehler auftreten:
  Es wurde sicher das Wort 000 gesendet.

Zusatzannahmen

Leitsatz zur Fehlererkennung

Jene Fehler, durch welche die Codemenge verlassen wird, können **immer** erkannt werden.
Völlige Fehlerfreiheit kann **niemals** sichergestellt werden.

Leitsatz über Zusatzannahmen

Unter Zusatzannahmen können **mehr Fehler erkannt** werden.
Unter Zusatzannahmen können **manche Fehler sogar korrigiert** werden.

Hamming-Abstand

Der **Hamming-Abstand** $d\colon A^* \times A^* \hookrightarrow \mathbb{N}_0$ ist eine *partiell definierte* Funktion, die Abstände $d(w_1, w_2)$ zwischen zwei Worten $w_1, w_2 \in A^*$ mißt.

Der Hamming-Abstand $d(w_1, w_2)$ ist *nur für Worte gleicher Länge* definiert.
Er ist die **Anzahl der unterschiedlichen Positionen** in den Worten:

$$d(\ a_1 a_2 \dots a_n, b_1 b_2 \dots b_n\ ) = \#\ (\ \{j \in \{1, 2, \dots, n\} \mid a_j \neq b_j\}\ )$$

Der Hamming-Abstand auf der Menge aller Worte der Länge n, also $d_n\colon A^n \times A^n \to \mathbb{R}_0^+$ ist eine **Metrik (Abstandsmaß)**, denn er erfüllt die entsprechenden Axiome:

1. **Positiv definit:** $\forall w_1, w_2 : \quad d(w_1, w_2) \geq 0 \ \wedge\ d(w_1, w_2) = 0 \Leftrightarrow w_1 = w_2$
2. **Symmetrisch:** $\forall w_1, w_2 : \quad d(w_1, w_2) = d(w_2, w_1)$
3. **Dreiecksungleichung:** $\forall w_1, w_2, w_3 : \quad d(w_1, w_2) + d(w_2, w_3) \geq d(w_1, w_3)$

Hamming-Abstand eines Blockcodes

Ein **Blockcode** über einem Alphabet ist ein Code mit Worten fester Länge.
Es gilt also nicht nur $\mathcal{C} \subseteq A^*$, sondern sogar auch $\mathcal{C} \subseteq A^n$.

Der **Hamming-Abstand** $h(\mathcal{C})$ **eines Blockcodes** $\mathcal{C}$ ist der kleinste Hamming-Abstand $d(w_1, w_2)$, den der Code über zwei verschiedene Code-Worte w_1, w_2 realisieren kann.

$$h(\mathcal{C}) := \min_{\substack{\forall w_1, w_2 \in \mathcal{C} \\ w_1 \neq w_2}} d(w_1, w_2)$$

Beispiel 1: Der Hamming-Abstand eines Paritäts-Codes ist zwei.
Beispiel 2: Der Hamming-Abstand des Tripel-Codes ist drei.

Fehler-Erkennung und Fehler-Behebung

Fehler-Erkennung:

1. Das System erkennt die Tatsache eines Fehlers.
2. Das System kann aber den Fehler unter Umständen nicht reparieren.
3. Auf die Tatsache des Fehlers muß anderweitig reagiert werden.

Fehler-Behebung:

1. Das System erkennt die Tatsache eines Fehlers.
2. Das System kann den Fehler selbständig beheben.
3. Das System kann so tun, als ob kein Fehler passiert ist. (sog. Fehler-Transparenz)

Hamming-Abstand und Fehler-Erkennung

Hamming-Abstand 1: Keine Fehler-Erkennung möglich.
Es gibt im Code mindestens zwei verschiedene Worte w_1 und w_2, die sich in genau einer Position unterscheiden. *Jene* Fehler, bei der aus w_1 durch Fehler in einer Position w_2 wird, können nicht erkannt werden.

Hamming-Abstand 2: 1-Positions-Fehler sind erkennbar.
Worte w_1 und w_2 unterscheiden sich in mindestens 2 Positionen. Bewirkt die Übertragung in einem Wort w einen Fehler an höchstens 1 Position, so kann das Ergebnis niemals ein Codewort sein, da das nächste Codewort mindestens 2 Positionen anders ist. 1-Positions-Fehler können daher *stets* erkannt werden.

Hamming-Abstand h: Maximal $(h-1)$-Positions-Fehler sind erkennbar.
Worte w_1 und w_2 unterscheiden sich in mindestens h Positionen. Bewirkt die Übertragung in einem Wort w einen Fehler an höchstens $h-1$ Positionen, so kann das Ergebnis niemals ein Codewort sein, da das nächste Codewort an mindestens h Positionen anders ist. Fehler in maximal $h-1$ Positionen können *stets* erkannt werden.

Hamming-Abstand und Fehler-Behebung (1)

Hamming-Abstand 2: Keine Fehler-Behebung möglich.
Es gibt im Code mindestens zwei Worte w_1 und w_2, die sich an genau zwei Positionen unterscheiden. Bewirkt die Übertragung im Wort w_1 einen Fehler an nur einer Position, so kann das Ergebnis aber auch durch einen Fehler im Wort w_2 entstanden sein. Die fehlerhafte Übertragung kann nicht eindeutig dem Wort w_1 oder w_2 zugeordnet werden.

Beispiel: $w_1 = abXYZ$ $w_2 = rsXYZ$.
Das empfangene Wort $asXYZ$ kann bei einem maximal-1-Positions-Fehler sowohl aus w_1 als auch aus w_2 entstanden sein.

Hamming-Abstand und Fehler-Behebung (2)

Hamming-Abstand 3: 1-Positions-Fehler sind behebbar.
Worte w_1 und w_2 unterscheiden sich in mindestens 3 Positionen. Bewirkt Übertragung in einem Codewort w Fehler an höchstens 1 Position, so kann das Ergebnis nicht auch durch 1-Positions-Fehler an einem anderen Codewort entstanden sein, da alle anderen Codeworte im Abstand 2 oder größer sind. Unter Annahme, daß nur 1-Positions-Fehler entstehen, kann das fehlerhafte Wort dem richtigen Ausgangswort zugeordnet werden.

Beispiel: $w_1 = abcXYZ$ $w_2 = rstXYZ$.
Das Wort $ascXYZ$ kann bei einem maximal-1-Position-Fehler nur aus w_1 entstanden sind und nicht aus w_2.

Hamming-Abstand und Fehler-Behebung (3)

Hamming-Abstand h: Maximal $\left\lceil \frac{h}{2} \right\rceil - 1$ Positions-Fehler sind behebbar.

Formaler Beweis:

Das Wort w_1 werde durch Übertragung zum Wort w_1'.
Das Wort w_2 werde durch Übertragung zum Wort w_2'.

Wir nehmen an, daß maximal k Positions-Fehler entstehen.

Das bedeutet: $d(w_1, w_1') \leq k$ und $d(w_2, w_2') \leq k$

Hamming-Abstand h bedeutet: $d(w_1, w_2) \geq h$.

Kriterium dafür, daß Fehler stets behebbar sind, ist, daß die Worte w_1' und w_2' immer verschieden sind.

Das ist äquivalent zu $d(w_1', w_2') > 0$.

Es reicht also aus, unter den gegebenen Bedingungen $d(w_1', w_2') > 0$ zu beweisen.

Hamming-Abstand und Fehler-Behebung (4)

Aufgrund der Dreiecksungleichung gilt: $d(w_1, w_2) \leq d(w_1, w_1') + d(w_1', w_2') + d(w_2', w_2)$.

Das ist äquivalent zu $d(w_1', w_2') \geq d(w_1, w_2) - d(w_1, w_1') - d(w_2', w_2)$.

Lt. Voraussetzung ist $d(w_1, w_2) - d(w_1, w_1') - d(w_2', w_2) \geq h - k - k$.

$h - k - k > 0$ gilt nun genau dann wenn $h > 2k$ oder wenn $k < \frac{h}{2}$.

Für natürliche Zahlen k und h ist das äquivalent zu $k \leq \lceil \frac{h}{2} \rceil - 1$ (überlegen !)

Wir haben also gezeigt: Wenn $k \leq \lceil \frac{h}{2} \rceil - 1$ dann ist $d(w_1', w_2') > 0$.

Somit sind Fehler in $\lceil \frac{h}{2} \rceil - 1$ oder weniger Positionen behebbar.

SECDED Codes

Codes mit Hamming-Abstand 3 sind häufig.

Sie heißen auch SECDED-Codes, denn sie

1. können 1-Bit-Fehler beheben: Single Error Correction: **SEC**
2. können 1-&-2-Bit-Fehler erkennen: Double Error Detection: **DED**

Beachte: SEC Modus und DED Modus können nicht kombiniert werden, da sich die Annahmen der Modi unterscheiden.

Beispiel: Der Tripel-Code $\{000, 111\}$ ist SECDED Code (Prüfe: Hamming-Abstand 3)
Empfänger erhält 011.

- Im SEC Modus: Richtige Reaktion ist: Das ist 111.
- Im DED Modus: Richtige Reaktion ist: Fehler.
  Denn: 011 könnte bei Doppelfehler auch 000 gewesen sein und darf daher nicht korrigiert werden.

2. Hamming-Codes

Ziele: Wir lernen ein wichtiges Konstruktionsprinzip für SECDED-Codes kennen und betrachten Codierung und Decodierung genauer.

Paritäts-Code

Der **3-stellige Paritätscode gerader Parität** ist $\mathcal{C} = \{000, 011, 101, 110\}$.
Er lebt in $\{0,1\}^3$ und hat Hamming-Abstand 2.

Sender			Empfänger		
DB	PB	SB	EB	Stat	DB
00	0	000	000	Ok	**00**
			001	Err	
			010	Err	
01	1	011	011	Ok	**01**
			100	Err	
10	1	101	101	Ok	**10**
11	0	110	110	Ok	**11**
			111	Err	

Tab. 1: Sender: DB: Datenbits. **PB:** Prüfbit. **SB:** Gesendete Bits. **Empfänger: EB:** Empfangene Bits. **Stat:** Prüfstatus. **DB:** Datenbits. *Genau genommen* gehören zu *jeder Anwendung* eines Codes auch solche Anweisungen, wie Sender-seitig codiert und Empfänger-seitig decodiert werden muß.

(3,1)-Hamming-Code (auch: Tripel-Code)

Der (3,1)-Hamming-Code ist $\mathcal{C} = \{000, 111\}$ in $\{0,1\}^3$; er hat Hamming-Abstand 3.

Empfänger

EB 321	SEC	DB	DED
000	Ok	0	Ok
001	E1	0	Err
010	E2	0	Err
011	E3	1	Err
100	E3	0	Err
101	E2	1	Err
110	E1	1	Err
111	Ok	1	Ok

Tab. 2: Empfänger: EB: Empfangene Bits (durchnummeriert). **SEC:** Prüfstatus im SEC Modus. **DB:** Erkanntes Datenbit im SEC Modus. **DED:** Prüfstatus im DED Modus. Es bleibt aber noch unklar, wie der Empfänger vorgegangen ist. Keines der drei EB entspricht immer dem DB – was aber wegen möglicher Fehler nicht verwundert.

Systematischer Entwurf von Hamming-Codes (1)

Wir haben 3 Bits g_1, g_2, g_3 und versuchen einen SEC Modus zu konstruieren.

In jedem der 3 Bits kann ein Fehler auftreten.

Der Empfänger muß also Funktionen haben, sodaß jeweils eine Änderung nur in Bit g_1 oder nur in Bit g_2 oder nur in Bit g_3 ersichtlich wird. Da der Empfänger Fehler korrigieren will, muß er diese drei Situationen **unterscheiden** können.

Codierung 3er Situationen braucht **mindestens zwei** binäre Funktionen: f_1 und f_2.

Für die **wechselseitigen Abhängigkeiten** soll gelten:

	f_1	f_2
g_1	X	
g_2		X
g_3	X	X

Änderung in g_1 bewirkt Änderung (nur) von f_1
Änderung in g_2 bewirkt Änderung (nur) von f_2
Änderung in g_3 bewirkt Änderung von f_1 und f_2.
Funktional abhängig notiert: $f_1(g_1, g_2)$ und $f_2(g_2, g_3)$.
Versuche xor: Jede Argument-Änderung bewirkt Wertänderung.
Also: $f_1 = g_1 \oplus g_3$ und $f_2 = g_2 \oplus g_3$

Systematischer Entwurf von Hamming-Codes (2)

Empfänger

g_3	g_2	g_1	$f_2 = g_3 \oplus g_2$	$f_1 = g_3 \oplus g_1$	$(f_2 f_1)_2$	SEC	DB
0	0	0	0	0	0	Ok	0
0	0	1	0	1	1	E1	0
0	1	0	1	0	2	E2	0
$\underline{0}$	1	1	1	1	3	E3	$\underline{1}$
$\underline{1}$	0	0	1	1	3	E3	$\underline{0}$
1	0	1	1	0	2	E2	1
1	1	0	0	1	1	E1	1
1	1	1	0	0	0	Ok	1

Tab. 3: f_2 und f_1 binär gelesen codieren die Nummer des fehlerhaften Bits – siehe die jeweils rot, grün und blau hervorgehobenen Werte. Das vom Empfänger rekonstruierte Datenbit DB ist identisch mit dem empfangenen Bit g_3 – wie in cyan hervorgehoben ist. In den zwei unterstrichenen Fällen stimmt das nicht, was aber ok geht, da das genau jene Fälle sind, in denen die Fehlerbehebung einen Fehler im Bit 3, also g_3 prognostiziert. Die zwei Ok-Zeilen ohne Fehler erlauben nun noch die Konstruktion der Sender-Tabelle.

Systematischer Entwurf von Hamming-Codes (3)

Sender			Empfänger			
DB	**SB**		**EB**	**SEC**	**DB**	**DED**
0	000		000	Ok	0	Ok
			001	E1	0	Err
			010	E2	0	Err
			011	E3	1	Err
			100	E3	0	Err
			101	E2	1	Err
			110	E1	1	Err
1	111		111	Ok	1	Ok

Tab. 4: Wir können nun auch die Sender-Tabelle ergänzen. **Sender:** **DB:** Datenbit. **SB:** Gesendete Bits. **Empfänger:** **EB:** Empfangene Bits. **SEC:** Prüfstatus im SEC Modus. **DB:** Erkanntes Datenbit im SEC Modus. **DED:** Prüfstatus im DED

Weitere Hamming-Codes

Hamming-Codes sind Codes nach diesem Konstruktions-Prinzip.
Sie haben Hamming-Abstand 3 und eigenen sich für SEC-DED Betrieb.

Prüf Bits	**Total** Bits	**Daten** Bits	**Übliche Bezeichnung**
2	3	1	(3,1)-Hamming-Code (auch: Tripel-Code)
3	7	4	(7,4)-Hamming-Code
4	15	11	(15,11)-Hamming-Code
5	31	26	(31,26)-Hamming-Code
...			
m	$2^m - 1$	$2^m - m - 1$	$(2^m - 1,\ 2^m - m - 1)$-Hamming-Code

Tab. 5: Übersicht über Hamming-Codes.

3. CRC

Ziele: Der CRC ist der wichtigste und am häufigsten eingesetzte Code. Wir lernen Mathematik, Elektronik und Anwendung kennen.

Bedeutung des CRC

Motivation: Hamming-Codes finden noch zu wenig Fehler
und sind nicht gut auf die Wort-Breiten der Informatik angepaßt.

Besser ist der **CRC**: Cyclic Redundancy Check

Mathematik:

1. Theorie der Polynom-Division auf endlichen Körpern – ist sehr gut untersucht.
2. Familie von Verfahren, durch ein Generator-Polynom parametrisiert.
3. Sehr gute Fehlerkorrektur-Verhalten sind möglich.

Hardware & Implementierung:

1. Endlicher Automat: Shift-Register mit Feedback und xor.
2. Kann on the fly – Bit für Bit – mitgerechnet werden.
   Ergebnis steht mit dem letzten Bit sofort zur Verfügung.
3. **Sehr** wenig Hardware-Aufwand, daher fast überall eingebaut.

Polynome über Körpern

Polynome sind "Ausdrücke der Form" $a_n \cdot X^n + a_{n-1} \cdot X^{n-1} + \ldots + a_1 \cdot X^1 + a_0 \cdot X^0$.
Die Koeffizienten a_j sind aus einer Menge $\mathbb{K}$; das X ist ein Symbol.

Präziser kann man Polynome als endliche Folgen oder n-tupel $(a_n, a_{n-1}, \ldots, a_1, a_0)$ mit bestimmten Rechenregeln definieren.

Typischerweise ist der **Koeffizientenbereich** $\mathbb{K}$ ein Körper.

Ein **Körper** ist eine Menge, auf der man addieren, subtrahieren, multiplizieren und durch Werte ungleich 0 dividieren kann mit bestimmten Gesetzen.
Die **reellen** Zahlen $\mathbb{R}$ und die **rationalen** Zahlen $\mathbb{Q}$ sind solche Körper.

Genauer: Siehe Mathematik-Grundvorlesungen.

Endliche Körper

Endliche Körper sind in der Informatik wichtig.

Der einfachste endliche Körper ist der **Galois-Körper** $GF(2)$.
Bezeichnet nach dem französischen Mathematiker Evariste Galois
und dem englischen Wort *field* für Körper.

$GF(2) = \{0, 1\}$ mit den Rechenoperationen $+$, $-$, $\cdot$ und $/$ wie folgt:

$+$	0	1
0	0	1
1	1	0

$-$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

$/$	0	1
0		0
1		1

Das ist genau das Rechnen mit Restklassen modulo 2.

Beispiel: $1 + 1 = 2 = 0$

Beispiel: $-1 = +1$ denn: -1 ist jene Zahl, die zu 1 addiert, 0 ergibt.

Man kann durch alle Zahlen ungleich 0 dividieren – und in GF(2) ist das nur die 1.

Bit-Ketten als Polynome über GF(2)

Bit-Ketten können als Polynome über GF(2) verstanden werden.

Polynome über GF(2) entsprechen Bit-Ketten.

Diese Identifikation erlaubt eine einfachere Formulierung
vieler Fragen der Codierungs-Theorie.

Beispiel: 1101 entspricht $1 \cdot X^3 + 1 \cdot X^2 + 0 \cdot X^1 + 1 \cdot X^0 = X^3 + X^2 + 1$

Wiederholung: Division von Polynomen über dem GF(2)

$$+x^3 \quad +x^2 \quad +x^1 \quad +0 \quad : (x+1) =$$

Wiederholung: Division von Polynomen über dem GF(2)

$$+x^3 \quad +x^2 \quad +x^1 \quad +0 \;\; : (x+1) = x^2$$

Wiederholung: Division von Polynomen über dem GF(2)

$$
\begin{array}{rrrrl}
+x^3 & +x^2 & +x^1 & +0 & :(x+1) = x^2 \\
+x^3 & +x^2 & & & \\
\hline
\end{array}
$$

Wiederholung: Division von Polynomen über dem GF(2)

$$
\begin{array}{cccc}
+x^3 & +x^2 & +x^1 & +0 \\
+x^3 & +x^2 & & \\
\hline
0 & 0 & &
\end{array}
: (x+1) = x^2
$$

Wiederholung: Division von Polynomen über dem GF(2)

$$
\begin{array}{cccc}
+x^3 & +x^2 & +x^1 & +0 \\
+x^3 & +x^2 & & \\
\hline
\cancel{0} & 0 & &
\end{array}
: (x+1) = x^2 + 1
$$

Wiederholung: Division von Polynomen über dem GF(2)

$$
\begin{array}{cccc}
+x^3 & +x^2 & +x^1 & +0 \\
+x^3 & +x^2 & & \\
\hline
\cancel{0} & 0 & & \\
 & & +x & +1 \\
\hline
\end{array}
: (x+1) = x^2 + 1
$$

Wiederholung: Division von Polynomen über dem GF(2)

$$
\begin{array}{cccc}
+x^3 & +x^2 & +x^1 & +0 \\
+x^3 & +x^2 & & \\
\hline
\cancel{0} & 0 & & \\
 & & +x & +1 \\
\hline
 & & 0 & 1
\end{array}
\;:\,(x+1) = x^2+1
$$

Wiederholung: Division von Polynomen über dem GF(2)

$$
\begin{array}{cccc l}
+x^3 & +x^2 & +x^1 & +0 & : (x+1) = x^2 + 1 \\
+x^3 & +x^2 & & & \\
\hline
\cancel{0} & 0 & & & \\
 & & +x & +1 & \\
\hline
 & & 0 & 1 &
\end{array}
$$

$$x^3 + x^2 + x^1 + 0 : (x+1) = (x^2+1) + \frac{1}{x+1} \qquad \text{Rest: } 1$$

Probe: $(x^2+1) \cdot (x+1) + 1 \stackrel{!}{=} x^3 + x^2 + x^1 + 0$

Es ist: $(x^2+1) \cdot (x+1) + 1 = x^3 + x^2 + x^1 + 2 = x^3 + x^2 + x^1 + 0$ ok!

CRC

Für jeden CRC ist ein Polynom G vorgegeben (sog. Generatorpolynom).
Sei n der Grad des Generatorpolynoms.

Sender: Will Daten M versenden (sog. Nutzlast).

1. Verlängert Nutzlast um n Nullen (sog. Gradnullen): $M \mapsto M \cdot X^n$
2. Ermittelt Rest R bei Division von $M \cdot X^n$ durch G.
   Für R gilt $M \cdot X^n = Q \cdot G + R$ also $R = M \cdot X^n - Q \cdot G$.
3. Sendet $S = M \cdot X^n + R$ an Empfänger.

Empfänger: Erhält Nachricht E

1. Berechnet den Rest bei Division durch G.
2. Wenn Rest 0: Nachricht akzeptiert.
3. Sonst: Fehlermeldung.

Warum funktioniert das?

Sender sendet $M \cdot X^n + R$ wobei $R = M \cdot X^N - Q \cdot G$ war.

Wenn kein Fehler passiert, dann kommt beim Empfänger an:
$M \cdot X^n + R = 2 \cdot M \cdot X^n - Q \cdot G = -Q \cdot G$.

Das ist durch G ohne Rest teilbar.

Fazit also:

- Fehlerfrei übertragene Nachrichten akzeptiert der Empfänger.
- Fehler, welche die Teilbarkeitsbedingung zerstören, erkennt der Empfänger und weist sie zurück.

Es ist möglich, Generatorpolynome zu entwickeln, die

- Alle Einzelfehler erkennen.
- Alle Doppelfehler erkennen.
- Jede ungerade Anzahl von Fehlern erkennen.
- Jeden Burst-Fehler bis zu einer Fehlerlänge m erkennen.

Beispiel für den CRC

Generator-Polynom: $X^5 + X^4 + X^2 + 1$ (Grad 5).

Nachricht: 1010 0011 01

Erweitere um die Gradnullen: 1010 0011 01 00000

Sinn der Übung:

- Man muß sich den Mechanismus genauer durchdenken.
- Wir erkennen den Zusammenhang mit der Hardware-Implementierung.

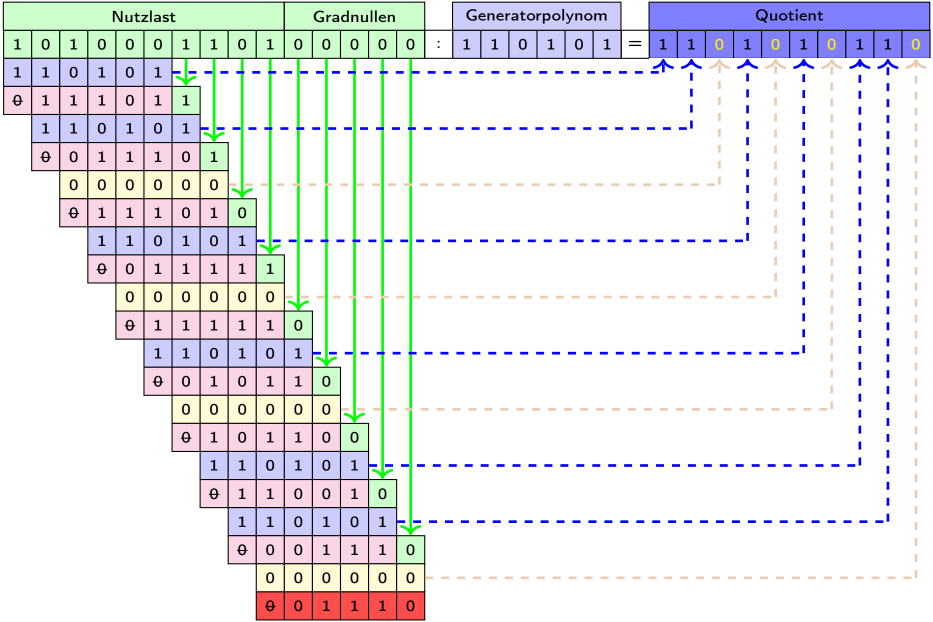


Abb. 1: Berechnungsschema für den CRC ohne mitgeschleppte Variable X.

Hardware-Implementierung

Das Schema ist **sehr einfach** in Hardware zu übersetzen!

Schreibe Koeffizienten des Generator-Polynoms von links nach rechts.
In der Zeile darunter baue eine Schaltung.

1. Zwischen die Koeffizienten kommt eine Speicherstelle
2. Unter die Einser kommt ein xor.
3. Unter die Nullen kommt ein Lötpunkt.
4. Verbinde von rechts nach links und wieder zurück
5. Schließe die xor an.

Abarbeiten:
Sender: Schiebe die Nachricht mit den Gradnullen hinein.
Am Ende bleibt der Rest stehen.
Empfänger: Schiebe die Nachricht mit den Prüfbits heinein.
Am Ende muß der Rest 0 herauskommen.

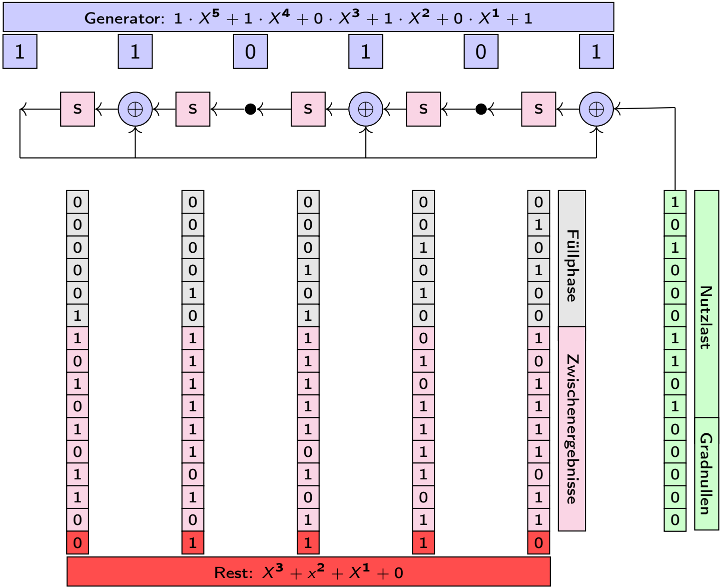
Wirklisch einfach!

Damit man sieht, daß das wirklich einfach ist,
muß man das einmal in seinem Leben 1-1 durcharbeiten.

Das soll das Farbschema ermöglichen.

Lernhilfe:

1. Wir erarbeiten uns das Berechnungsschema. (geschehen!)
2. Wir erarbeiten uns den Schaltungsaufbau.
3. Wir erarbeiten uns die Funktionsweise der Schaltung.
4. Wir vergleichen die 9 Zeilen in Magenta mit den Zwischenergebnissen.
   Die Inhalte stimmen im Schema und in der Hardware überein.



Generator von links nach rechts.

1. Zwischen Koeff.: Speicher.
2. Unter die 1: xor.
3. Unter die 0: Lötpunkt.

Verbinde von rechts nach links.
Schleife zurück in die xor.

Initial: 0

Nutzlast mit Gradnullen rechts rein.

Abb. 2: Umbau des Berechnungsschemas in eine Hardware-Implementierung

Beispiele

CRC-32 für Ethernet:
$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^{8} + X^{7} + X^{5} + X^{4} + X^{2} + X^{1} + 1$

Es gibt eigene angepasste CRC-Varianten unter anderem für USB, Bluetooth, SD Cards, Wifi, Ethernet, CAN-Bus, 3G, 4G, 5G

Anhang

Übersicht

Verzeichnis aller Abbildungen

Verzeichnis aller Tabellen (1/2)

Rechtliche Hinweise

Zitierweise dieses Dokuments

Wenn Sie Inhalte aus diesem Werk nutzen oder darauf verweisen wollen,
zitieren Sie es bitte wie folgt:

Clemens H. Cap: Codierung und Sicherung. Electronic document. https://iuk.one/1010-1016 3. 1. 2021.

Bibtex Information: https://iuk.one/1010-1016.bib

```
@misc{doc:1010-1016,
   author       = {Clemens H. Cap},
   title        = {Codierung und Sicherung},
   year         = {2021},
   month        = {1},
   howpublished = {Electronic document},
   url          = {https://iuk.one/1010-1016}
}
```

Typographic Information:
Typeset on January 3, 2021
This is pdfTeX, Version 3.14159265-2.6-1.40.21 (TeX Live 2020) kpathsea version 6.3.2
This is pgf in version 3.1.5b
This is preamble-slides.tex myFormat©C.H.Cap

Abkürzungsverzeichnis

Verzeichnis aller Folien

3. CRC

Legende:

- Fortsetzungsseite
- ○ Seite ohne Überschrift
- Bildseite