Referat „Error detection and correction“ im Proseminar „Computer Science Unplugged“
Dozent Prof. M. Hofmann
Referent Pinto Raul, 48005464
Datum 19.11.2004

Error detection and correction

1. Fehlererkennung
    1.1 Parität
    1.2 CRC
2. Von der Erkennung zur Korrektur
    2.1 Erweiterte Parität
    2.2 Hamming Codes
    2.3 Reed-Muller Code
3. Anhang

1. Fehlererkennung

Bei jedem Transport von Information kann ein Fehler auftreten.
Auf jedem Bus, über Netzwerke und vor allem über Funk.
Aus diesem Grunde versucht die Informatik Information so zu verschlüsseln, daß Fehler erkannt werden können.

1.1 Fehlererkennung durch Parität

Um in einer Zeile Fehler zu erkennen kann man als leichtestes Mittel die Parität verwenden. Im Endeffekt ist diese ein Bit am Ende einer Zeile. Der Wert dieses Bits hängt von der Anzahl der mit „1“ gesetzten Bits ab. Das Paritätsbit macht die Anzahl der 1er in der Zeile immer gerade.

Beispiel:

Bitfolge	***Parität***
10010010	1
11011011	0

Diese Art der Fehlererkennung ist sehr simpel. Sie wird zum Beispiel bei der seriellen Datenübertragung verwendet. Dies kennt man vielleicht noch aus den Einstellungen der seriellen Schnittstelle (COM) in denen man jede mögliche Art der Parität einstellen kann.

Das Problem dieser Technik ist die Unfähigkeit, mehrere Fehler zu erkennen. Enthält eine Zeile zum Beispiel zwei umgekippte Bits, stimmt die Summe wieder. Auch kann nicht unterschieden werden, ob 1, 3, 5 usw. Fehler aufgetreten sind.

Die Technik der Parität ist veraltet und wird kaum noch verwendet. Stattdessen wird CRC verwendet.

1.2 Fehlererkennung durch CRC

Die Fehlererkennung durch CRC (= Cyclic Redundancy Code) basiert auf der Idee, die zu übertragende Bitfolge als Polynom anzusehen. Dieses Polynom ähnelt der Umrechnung in Dezimalzahlen.

Beispiel:

Bitfolge	***Polynom***
10010010	$x^7+x^4+x^1$
11011011	$x^7+x^6+x^4+x^3+x^1+x^0$

(Ist x=2 rechnet man so den Wert von binär auf dezimal um)

Die Bitfolge wird nun mit um ein paar weitere Bits so erweitert, daß die daraus

entstehende Bitfolge durch ein vorher definiertes Polynom (das sogenannte Generatorpolynom) teilbar ist.

Algorithmus zur Berechnung der CRC-Prüfsumme:

Sei m die Länge und r der Grad des Generatorpolynoms G(x). M(x) das Polynom des zu übertragenden Bitstrings der Länge m.
1. Hänge am Bitstring r 0-Bits an. Der resultierende Rahmen hat nun die Länge m+r entsprechend dem Polynom $x^{r*M(x)}$
2. Dividiere den Bitstring $x^{r*M(x)}$ durch den Bitstring des Generatorpolynoms G(x) mit modulo-2-Division
3. Subtrahiere den Divisionsrest von dem $x^{r*M(x)}$ entsprechendem Bitstring (mit modulo 2). Das Resultat T(x) ist die zu übertragende Bitfolge mit Prüfsumme.

Beispiel: (mit Bluetooth Generatorpolynom $x^5+x^4+x^2+1$)

Bitfolge	*Rest*	*Übertragung*
10010010	11010	1001001011010
11011011	10101	1101101110101

(Rechnungen siehe Anhang)

Da der Rest sozusagen auf das Ursprungspolynom aufaddiert wurde, muß der Empfänger nur noch die Daten durch das Generatorpolynom teilen und schauen, ob der Rest 0 ist. Ist er das nicht, ist ein Fehler in der Übertragung aufgetreten.

Tritt ein Fehler auf, ist das sozusagen ein Vektor E(x) der auf den übertragenen Bitstring aufaddiert wurde, also T(x) + E(x).

Folgende Fehler können erkannt werden:
1. Singlebitfehler: $E(x) = x^i$
   G(x) muss mindestens 2 Terme haben, dann werden solche erkannt
2. 2 Singlebitfehler: $E(x) = x^i + x^j$, $i > j$
   G(x) darf nicht durch x^{k+1} teilbar sein, für k=1 ... Framelänge
3. Ungerade Anzahl von Bitfehlern:
   G(x) muss den Faktor x + 1 enthalten
4. Bursts[1]:
   Ein Polynom des Grads r erkennt alle Burstfehler einer Länge <= r, dazu muss G(x) einen x^0-Term enthalten. Bursts der Länge r + 1 werden mit einer Wahrscheinlichkeit von $0{,}5^{(r-1)}$ nicht erkannt.

Die CRC-Fehlererkennung wird zum Beispiel bei Ethernet, ZIP und PNG verwendet.

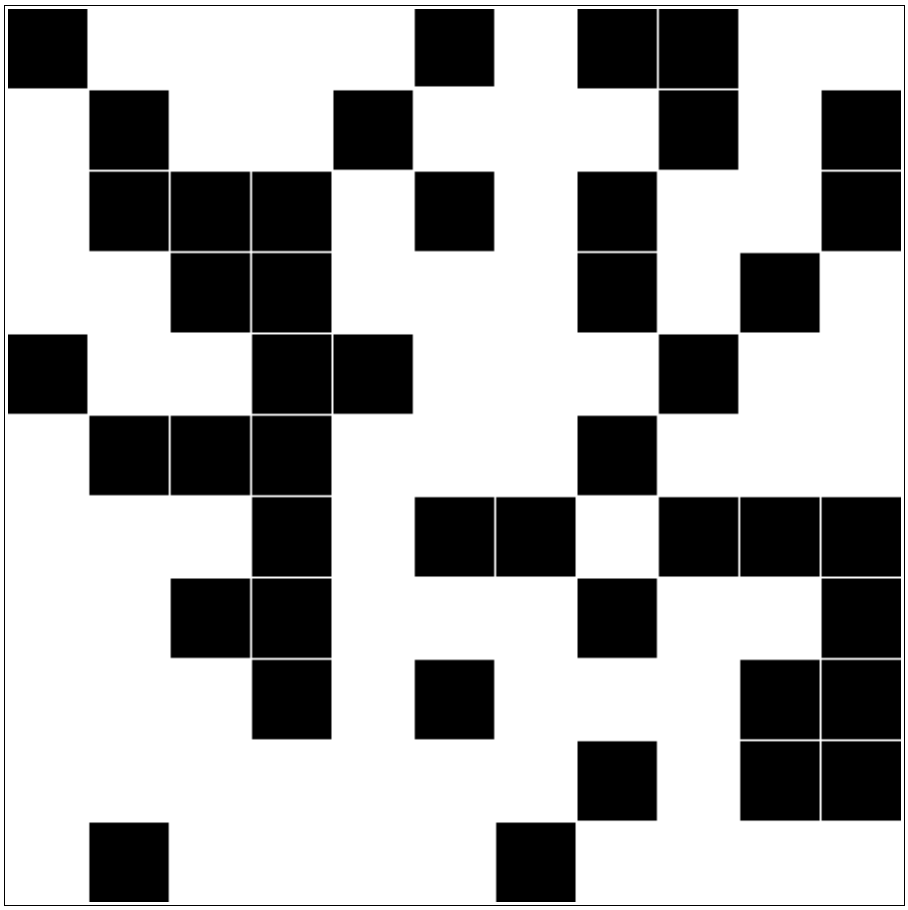
1 Ein Burst ist eine 1, gefolgt von 0 oder 1 und dann wieder eine 1

2. Von der Erkennung zur Korrektur

Allein Fehler zu erkennen hilft nicht immer. In Computernetzwerken, egal welcher Größe, können Daten auch mehrmals übertragen werden.
Dies ist allerdings nicht möglich, wenn sich um Meßgeräte handelt. Eine Sonde der NASA kann ihre Daten zum Beispiel nur einmal aufnehmen und verschicken. Eine Speicherung der Daten ist aus Energie- und Platzgründen nicht möglich.
Aus diesem Grund werden bei der Funkübertragung aus dem Weltraum auf die Erde besondere Kodierungen verwendet, die Fehler nicht nur erkennen lassen sondern sogar korrigieren können.

2.1 Fehlerkorrektur durch Parität

Analog zur Fehlererkennung gibt es bei der Fehlerkorrektur auch eine Art der Parität. Man verwendet bei einer Matrix zum Beispiel nicht nur das letzte Bit einer jeder Zeile als Parität sondern auch die erste Zeile.
Die folgende Matrix soll dies zeigen. Jedes schwarze Kästchen entspricht einer 1, ein weißes „unausgemaltes“ einer 0.



Da die Anzahl der schwarzen Kästchen in jeder Zeile als auch in jeder Spalte gerade sein sollte, kann genau das Kästchen erkannt werden, welches umgekippt ist.

Kippen allerdings 3 Kästchen kann man im schlimmsten Fall diesen Fehler schon nicht mehr erkennen und erst recht nicht beseitigen.

2.2 Fehlerkorrektur durch Hamming-Code

Dies ist eine spezielle Lösung um Code mit einer vorgegebenen Hamming-Distanz zu erzeugen. In s-stellige Codewörter werden q Prüfpositionen eingeführt, so daß $s-q$ Bits für die Information verbleiben.
Mit den q Prüfpositionen können 2^q Zustände unterschieden werden, nämlich die korrekte Übertragung und 2^q-1 Fehlerpositionen beim Auftreten eines Fehlers. Daher gilt:
$2^q \geqslant s+1$
Will man nur einen Fehler erkennen, so erhalten wir die Formel
$2^{s-q} \leqslant 2^s/(1+s) \Rightarrow 1+s \leqslant 2^s/2^{s-q} \Rightarrow s^q \geqslant s+q$
Wählen wir nun $q=3$ Prüfpositionen, dann folgt daraus:
$8=2^3 \leqslant s+1$
Es reicht also $s=7$ entsprechend einem Code mit 4 Informations- und 3 Prüfbits.
Zur Codierung ordnet man nun alle 7 Kombinationen der Prüfbits die Fehlern entsprechen in einem als *Kontrollmatrix* M bezeichneten Schema an:

Fehlerhafte Stelle	***Prüfbits:***	***p1***	***p2***	***p3***
1		0	0	1
2		0	1	0
3		0	1	1
4		1	0	0
5		1	0	1
6		1	1	0
7		1	1	1

Vor der eigentlich Codierung müssen nun noch die Positionen der Prüfstellen festgelegt werden. Dabei ist zu beachten, daß alle Prüfbits voneinander linear unabhängig sein müssen. So wären zum Beispiel die Positionen 1, 2 und 3 wegen der Linearkombination 3=1+2 als Prüfpositionen nicht zulässig. Im Regelfall wählt man daher die Postitionen als Zweierpotenzen, also 1, 2, 4, ...
In diesem Fall würde man für ein Codewort c also die Form wählen, wobei ix und py mit $i \in \{1,2,3,4\}$ und $y \in \{1,2,3\}$:
$c=(p1\ p2\ i1\ p3\ i2\ i3\ i4)$

Die Werte der Prüfbits können aus dem homogenen Gleichungssystem berechnet werden:

$p3+i2+i3+i4=0$
$p2+i1+i3+i4=0$
$p1+i1+i2+i4=0$

Setzt man nun nacheinander alle möglichen Kombinationen für i1, i2, i3, i4 ein, so folgt der gesucht Hamming-Code:

x_i	*p1*	*p2*	*i1*	*p3*	*i2*	*i3*	*i4*
0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0
3	1	0	0	0	0	1	1
4	0	1	0	1	0	1	0
5	0	1	0	0	1	0	1
6	0	0	1	0	1	1	0
7	0	0	1	1	0	0	1

x_i	*p1*	*p2*	*i1*	*p3*	*i2*	*i3*	*i4*
8	1	1	1	1	1	1	1
9	0	0	0	1	1	1	1
11	0	1	1	1	1	0	0
12	1	0	1	0	1	0	1
13	1	0	1	1	0	1	0
14	1	1	0	1	0	0	1
15	1	1	0	0	1	1	0

In der Praxis wird diese Tabelle aber nicht gespeichert, da ihre Berechnung durch die XOR-Operation sehr leicht ist.

Wird ein Codewort c nun empfangen, muß festgestellt werden, ob es korrekt ist. Dazu nutzt man aus, daß jedes Codewort Lösung des Gleichungssystems $x \cdot M = 0$ ist. Empfängt man nun ein Wort y, so berechnet man $y \cdot M$. Ergibt dieses 0, so ist kein Fehler aufgetreten und die Informationsbits können von den entsprechenden Stellen abgelesen werden.
Ist das Ergebnis nicht 0, so gibt es binär kodiert die Fehlerposition an. Den Fehler kann man einfach durch die Inversion des betreffenden Bits korrigieren.

Beispiel:
Sei y=010011. Die Multiplikation mit M ergibt:

$$(1\,0\,1\,0\,0\,1\,1)\cdot\begin{pmatrix}0&0&1\\0&1&0\\0&1&1\\1&0&0\\1&0&1\\1&1&0\\1&1&1\end{pmatrix}=(0\,1\,1)$$

Das Ergebnis der Multiplikation ist offenbar nicht der Nullvektor, sondern die binäre darstellung der Ziffer 3. Damit ist die 3. Stelle von links fehlerhaft.
Der korrekte Code ist also 1000011

2.3 Fehlererkennung durch Reed-Muller-Code

Die NASA selbst verwendet allerdings einen komplizierteren Code, der auf den Hadamardmatrizen basiert.
Grundlage dieser Fehlererkennung sind die sogenannten Hadamardmatrizen. Diese sind rekursiv definiert durch:

$$H_2=\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}$$
$$H_{2k}=\begin{pmatrix}H_k & H_k\\ H_k & -H_k\end{pmatrix}, k\in\mathbb{N}$$

Die Besonderheit dieser Matrizen ist, daß sich jede Zeile einer H_{2k}-Matrix in k Zeichen von jeder anderen Zeile unterscheidet. Man sagt, die Zeilen haben einen „Hamming-Abstand“ von k.

Beweis durch Induktion über k:

I.V.: $HA(H_{2k})=k$

I.A.: $HA(H_2)=HA\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}=1$

I.S.: $HA(H_{2(2k)})=HA\begin{pmatrix}H_{2k} & H_{2k}\\ H_{2k} & -H_{2k}\end{pmatrix}$

1. Fall: Zwei Zeilen in der oberen Hälfte.
   Nach I.V haben alle Zeilen in H_{2k} einen HA von k. Da diese Zeilen nun doppelt so lang sind unterscheiden sie sich in der linken Hälfte durch k Stellen von jeder anderen Zeile und in der rechten Hälfte genauso.
   => Hamming-Abstand ist 2
2. Fall: Zwei Zeilen in der unten Hälfte:
   Die Argumentation für die linke Hälfte folgt der in Fall eins.
   Da es nur die Zustände 1 und -1 gibt sind in $-H_{2k}$ die Zeilen genau invertiert zu H_{2k}. Da aber alle Stellen invertiert sind, unterscheiden sich alle Zeilen weiterhin in k Stellen.
   => Hamming-Abstand ist 2
3. Fall: Eine Zeile in der oberen, eine in der unteren Hälfte:
   Da die linke Hälfte für die obere und untere Hälfte gleich ist, ist der interessanteste Fall, wenn die beiden Zeilen so gewählt sind, daß deren erste Hälfte gleich ist. Da nun eine Zeile der Länge $2k$ und deren Inversion verglichen werden unterscheiden sie sich in genau $2k$ Stellen.
   => Hamming-Abstand ist 2

In der NASA Sonde Mariner 9 wurde die H_{32}-Matrix (siehe Anhang) verwendet. Diese Matrix wird auch Reed-Muller-Code (32,64,16) genannt.

Da sich jede Zeile dieser Matrix in genau 16 Stellen von der nächsten Zeile unterscheidet, haben diese einen Hamming-Abstand von 16. Werden während der Übertragung 8 Stellen verändert ist die angekommene Zeile von genau zwei Zeilen gleich weit „entfernt“, aber es kann erkannt werden, daß ein Fehler aufgetreten ist. Dieser Code hat also die Fähigkeit bis zu 7 Fehler zu korrigieren und 8-15 zu erkennen.
Beim empfangen wird die empfangene Zeile mit jeder Zeile der Matrix verglichen und die unterschiedlichen Stellen gezählt. Die Zeile, die die wenigsten unterschiedlichen Stellen hat, ist die richtige.

Weitere Beispiele denen Fehlererkennende Codes eingesetzt werden

ISBN:
0-13-911991-4

s := (0*10)+(1*9)+(3*8)+(9*7)+(1*6)+(1*5)+(9*4)+(9*3)+(1*2)=
= 9 + 24 + 63 + 6 + 5 + 36 + 27 + 2 =
= 172

k := s mod 11 = 7

11 – k = 11 – 7 = 4

Personalausweis:
Die Personalausweisnummer besteht aus mehreren Zahlenfolgen.
Beispiel:

```
731731731    731731  731731
 *********    ******  ******
 wwwwNNNNNpD<<yyMMddX<YYmmDDx<<<<<<n
 ||||||||||  |||||||  |||||||      |
 ||||||||||  |||||||  |||||||      -\__ Pruefsumme ALLER Ziffern
 ||||||||||  |||||||  |||||||
 ||||||||||  |||||||  ||||||-\__ Pruefsumme Ablaufdatum
 ||||||||||  |||||||  ||||--\__ Ablaufdatum Tag
 ||||||||||  |||||||  ||--\__ Ablaufdatum Monat
 ||||||||||  |||||||  --\__ Ablaufdatum Jahr
 ||||||||||  |||||||
 ||||||||||  ||||||-\__ Pruefsumme Geb.-Datum
 ||||||||||  ||||--\__ Geb.-Tag
 ||||||||||  ||--\__ Geb.-Monat
 ||||||||||  --\__ Geb.-Jahr
 ||||||||||
 |||||||||| \__ Staatszugehoerigkeit D = Deutsch ?
 |||||||||| \__ Pruefsumme fuer wwwwNNNNN
 ||||-----\__ laufende Zaehlnummer
 ----_ Erstwohnsitz Kennzahl
```

Der Algorithmus der Prüfsumme:
1) Die erste Ziffer wird mit 7 multipliziert,
   die zweite Ziffer wird mit 3 multipliziert,
   die dritte Ziffer wird mit 1 multipliziert,
   und so weiter (die vierte wieder mit 7, die fuenfte mit 3 ...)

2) Die so erhaltenen Zahlen werden alle zu einer Summe aufaddiert.

3) Die Pruefsumme ist die Einerstelle der Summe (also Summe modulo 10).

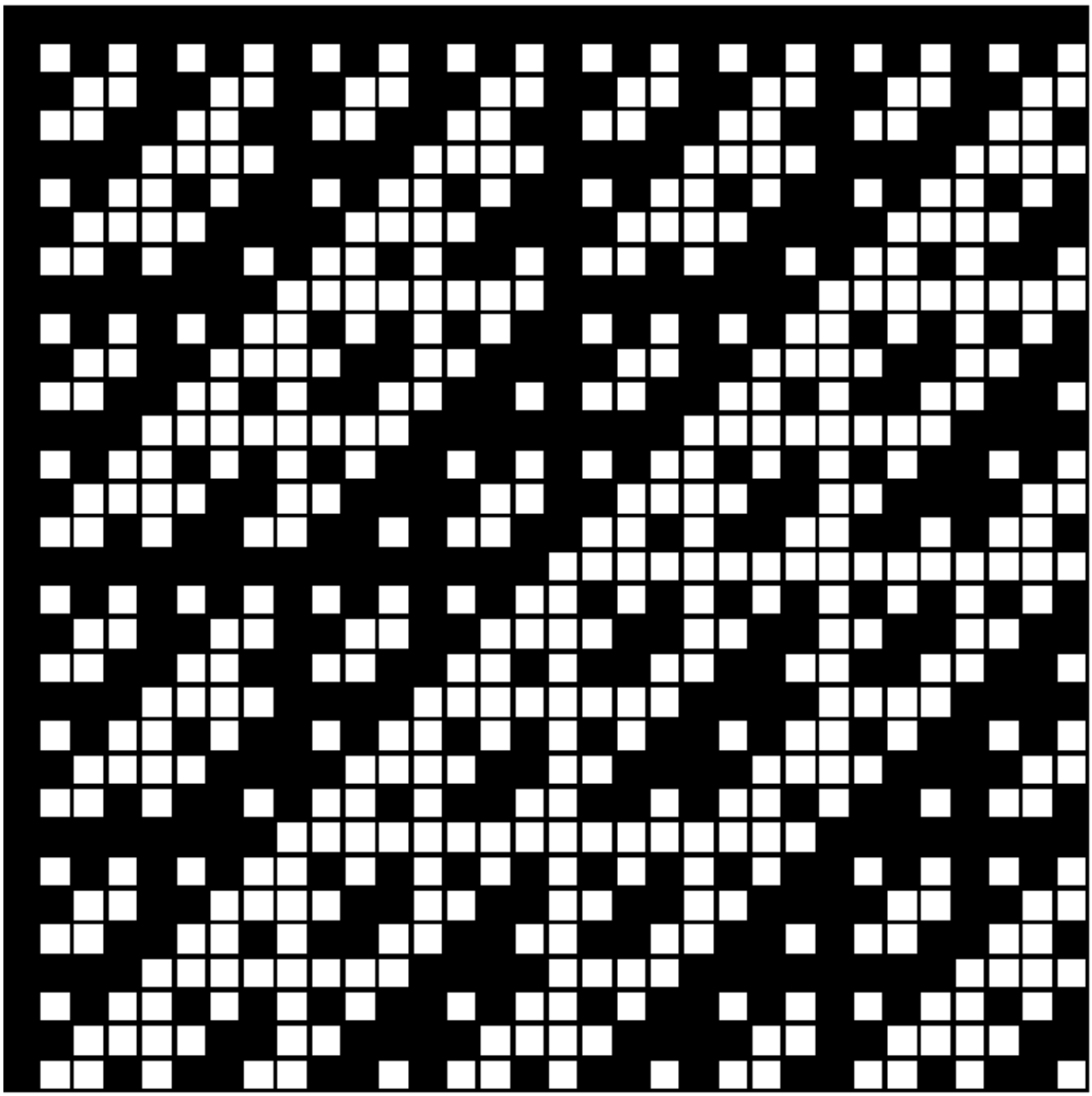
Anhang:

Berechnung der CRC Prüfsumme:

```
1001001000000 / 110101 = 11110010
110101
 100011
 110101
  101100
  110101
   110010
   110101
    00111000
      110101
       011010 = Rest

1101101100000 / 110101 = 1000101
110101
 000111100
    110101
     0100100
      110101
       100010
       110101
        10101 = Rest
```

Literatur:
http://de.wikipedia.org/wiki/Cyclic_Redundancy_Check

H_{32}-Matrix:
Jede 1 entspricht einem schwarzen Kästchen, jede -1 einem weißen Kästchen.

H_{32}