# Informatik Cheat Sheet

Informatik ist die Wissenschaft der systematischen Verarbeitung von Informationen, insbesondere der automatisierten Verarbeitung durch Rechenanlagen.

| | |
|---|---|
| Theoretische I. | Algorithmen, Leistungsfähigkeit derselben, Entscheidbarkeit ... |
| Praktische I. | Software, Konzepte zur Organisation und Verwaltung von Daten |
| Technische I. | Hardware, elektronischer Datenaustausch |
| Angewandte I. | Einsatz von Informatik in anderen Disziplinen (Wirtschaft, Medizin ...) |

## Geschichte

### Grundlagen

#### Theorie

| | |
|---|---|
| Rechengesetze (15. Jh.) | Adam Ries |
| Binäre Zahlendarstellung (1703) | Gottfried Leibniz |
| Boolesche Algebra (1847) | George Boole |
| Entscheidbarkeitsprobleme (1931) | Kurt Gödel |
| Turingmaschine (1936) | Alan Turing |
| Von-Neumann-Architektur | John v. Neumann |

#### Technik

| | |
|---|---|
| Bipolartransistor (1947) | Bell Laboratories |
| Programmiersprache Fortran (1955) | IBM |
| Chomsky-Hierarchie (1956) | Noam Chomsky |
| Unix (1969) | Bell Laboratories |

### Generationen

| | |
|---|---|
| Generation -1 (-1886) | Mechanische Rechmaschinen (Abakus, ..., Lochkartengesteuerten Zählmaschine) |
| Generation 0 (-1943) | Elektro-mechanische Rechenmaschinen (Z1-Z3) |
| Generation 1 (-1950) | Elektronische Rechner mit Röhren (Eniac, Z4) |
| Generation 2 (-1960) | Elektronische Rechner mit Transistoren (TRADC) |
| Generation 3 (-1968) | Rechner mit integrierten Schaltkreisen (Texas Instruments) |
| Generation 4 (-1984) | Hochintegrierte Computer (IBM-PC, Macintosh) |
| Generation 5 (seit 1992) | Vernetzte und parallele Computer (8 Petaflops mit 80 000 8-Kern Prozessoren) |

### Mooresches Gesetz

„Die Komplexität integrierter Schaltkreise mit minimalen Komponentenkosten verdoppelt sich regelmässig.“ Etwa alle 20 Monate verdoppelt sich die Anzahl Transistoren pro Fläche. Speicherkarten werden günstiger.

### Grössenangaben

| Metrisch | Anzahl | Binärpräfix | Anzahl |
|---|---|---|---|
| Kilobyte [kB] | $10^3$ Byte | Kibibyte [KiB] | $2^{10}$ Byte |
| Megabyte [MB] | $10^6$ Byte | Mebibyte [MiB] | $2^{20}$ Byte |
| Gigabyte [GB] | $10^9$ Byte | Gibibyte [GiB] | $2^{30}$ Byte |
| Terabyte [TB] | $10^{12}$ Byte | Tebibyte [TiB] | $2^{40}$ Byte |
| Petabyte [PB] | $10^{15}$ Byte | Pebibyte [PiB] | $2^{50}$ Byte |
| Exabyte [EB] | $10^{18}$ Byte | Exbibyte [EiB] | $2^{60}$ Byte |
| Zetabyte [ZB] | $10^{21}$ Byte | Zebibyte [ZiB] | $2^{70}$ Byte |
| Yotabyte [YB] | $10^{24}$ Byte | Yobibyte [YiB] | $2^{80}$ Byte |

iB zu B: Exponent = $\log_2$ Bytes

## Begriffe

| | |
|---|---|
| Bit | [0,1], Masseinheit für übertragene Datenmengen und Informationsgehalt |
| Byte | 8-Tupel von Bit, Masseinheit für Speicherung |
| Word | Grundverarbeitungsdatengrösse in einem Rechner. Immer eine 2er Potenz |

### Algorithmus

Ein Algorithmus ist eine Verarbeitungsvorschrift, die entweder durch einen Menschen oder eine Maschine (mechanisch/elektronisch) effizient durchgeführt werden kann, und bei der die Abfolge der endlichen Verarbeitungsschritte eindeutig festgelegt und ist und und bei welcher die einzelnen Verarbeitungsschritte eindeutig spezifiziert sind. Für den gleichen Input liefert ein Algorithmus also immer den gleichen Output.
Bsp. Euklid GGT von $a$ und $b$:
Widerhole:

1. $r = amodb$ Bestimme den Rest von $a/b$
2. $a = b$ Weise a den Wert von $b$ zu
3. $b = r$ Weise b den Wert von $r$ zu

bis $(r = 0)$

### Information

Gegeben ist ein Alphabet $Z = \{z_1, z_2, \dots z_n\}$, jedes Zeichen $z_i$ habe die Auftrittswahrscheinlichkeit $p_z$.

| | |
|---|---|
| Informationsgehalt | $I(z) = -\log_2 p_z$ Bit |
| Entropie | $H = \sum_{z \in Z} p_z \cdot I(p_z) = -\sum_{z \in Z} p_z \cdot \log_2 p_z$ Bit |
| Redundanz | $R = \frac{\text{Bits gesamt}}{\text{Bits genutzt}} - 1$ |

## Zahlensysteme

### Darstellung allgemein

Jede gebrochene Zahl $z$ mit $n$-Stellen vor und $m$ Stellen nach dem Komma $(n, m \in \mathbb{N}\backslash\{0\})$ mit einer Basis $B$ und den Ziffern $b \in \{0, 1, \dots, B-1\}$ lässt sich als Summe darstellen:

$$z = \sum_{i=-m}^{n-1} b_i \cdot B^i$$

### Konvertierung

#### Zahlensystem zur Basis $B$ ins Dezimalsystem

$n$ = Vorkommastellen, $m$ = Nachkommastellen

| | |
|---|---|
| Vor | $z = (\dots((b_n \cdot B + b_{n-1}) \cdot B + b_{n-2}) \cdot B + \dots + b_1) \cdot B + b_0$ |
| Nach | $z = (\dots((b_m \cdot 1/B + b_{-m+1}) \cdot 1/B + b_{-m+2}) \cdot 1/B + \dots + b_{-1}) \cdot 1/B$ |

#### Dezimalsystem in System zur Basis $B$

**Vorkommastellen** Eingabe Vorkommastellen $n$, Basis $B$
Wiederhole

1. $y = n : B$ Rest $z$ (Dividiere durch $B$, behalte $z$)
2. $n = y$ (Rechne mit Divisionsergebnis ohne Rest weiter)

bis $(y = 0)$
Die Reste $z$ von unten nach oben ergeben die die gesuchte Zahl.

Beispiel: $n = 122$, gesucht Darstellung zur Basis 8 (Oktal)

$$\begin{aligned} 122 \div 8 &= 15 \text{ Rest } 2 \\ 15 \div 8 &= 1 \text{ Rest } 7 \\ 1 \div 8 &= 0 \text{ Rest } 1 \rightarrow 122_{10} = 172_8 \end{aligned}$$

**Nachkommastellen** Eingabe Nachkommastellen $m$, Basis $B$
Wiederhole

1. $y = m \cdot B$ (Multipliziere $m$ mit $B$)
2. $m = \text{frac}(y)$ (Setze den Nachkommateil von $y$ zum neuen $m$)

bis $(m = 0)$
Die ganzzahligen Anteile von oben nach unten ergeben die gesuchte Zahl.

Beispiel: $n = 0{,}3984375$, gesucht Darstellung zur Basis 8 (Oktal)

$$\begin{aligned} 0{,}3984375 \cdot 8 &= 3{,}1875 \text{ Ganzzahlanteil: } 3 \\ 0{,}1875 \cdot 8 &= 1{,}5 \text{ Ganzzahlanteil: } 1 \\ 0{,}5 \cdot 8 &= 4{,}0 \text{ Ganzzahlanteil: } 4 \rightarrow 0{,}3984375_{10} = 0{,}314_8 \end{aligned}$$

### Einerkomplement

Erstes Bit = Vorzeichen (1 für negativ), Zahl = Invertierung der positiven Zahl. Überlauf ist, wenn zu einer 1 ganz links, eine 1 addiert wird. Diese wird der Ziffer ganz rechts hinzugefügt.
Nachteile: Redundante Darstellung der 0, separate Betrachtung des Überlaufs $\rightarrow$ 2 Additionen

### Zweierkomplement

Beim Zweierkomplement werden negativen Zahlen des Einerkomplements um 1 erhöht. Beim Überlauf wird ein Warnmeldungsflag (Overflow) auf wahr gesetzt.

### Festpunktzahlen

Haben eine festgelegte Länge von Vorkommazahlen $(n)$ und Nachkommazahlen $(m)$. Sie decken jeweils nur einen bestimmten Wertebereich ab, gewisse Zahlen lassen sich exakt darstellen, andere nicht. Rechnen mit Werten sehr unterschiedlicher Grösse ist schwierig bis unmöglich. Festkommazahlen werden daher nur für Spezialanwendungen gebraucht.

## Fliesskommazahlen

(IEEE 754): $s$ = Vorzeichen (0 positiv, 1 negativ), $M$ = Mantisse, $B$ = Basis und $E$ = Exponent.

$$x = s \cdot B^E \cdot M$$

Die Mantisse ist 23 bzw. 52 Bit lang und sie ist normiert (Dargestellt als 1.xxx), wobei die führende 1 weggelassen wird um die Präzision um ein Bit zu erhöhen, also zu verdoppeln.

| | **Single Precision** |
|---|---|
| Vorzeichen | 1 Bit |
| Exponent | 8 Bit |
| Mantisse | 23 Bit |
| Gesamtlänge | 32 Bit |
| Biaswert | 127 |
| Wertebereich Exponent | $-126 \leq e \leq 127$ |
| Dezimalstellen | 7 ... 8 |
| Genauigkeit | $2^{-(23+1)} \approx 5{,}96 \cdot 10^{-8}$ |
| Minimum | $2^{-(23+126)} \approx 1 \cdot 10^{-45}$ |
| Maximum | $(1 - 2^{-24}) \cdot 2^{128} \approx 3{,}4 \cdot 10^{38}$ |
| | **Double Precision** |
| Vorzeichen | 1 Bit |
| Exponent | 11 Bit |
| Mantisse | 52 Bit |
| Gesamtlänge | 64 Bit |
| Biaswert | 1 023 |
| Wertebereich Exponent | $-1\,022 \leq e \leq 1\,023$ |
| Dezimalstellen | 15 ... 16 |
| Genauigkeit | $2^{-(52+1)} \approx 1{,}1 \cdot 10^{16}$ |
| Minimum | $2^{-(52+1\,022)} \approx 5 \cdot 10^{-324}$ |
| Maximum | $(1 - 2^{-52}) \cdot 2^{1\,024} \approx 1{,}8 \cdot 10^{308}$ |

Beispiel: Darstellung von $18{,}4_{10}$ als IEEE 754 Gleitpunktzahl:

1. Vorzeichen: positiv $\rightarrow 0$
2. Mantisse:
   a) Umwandlung in duale Festkommazahl ohne Vorzeichen:

$$18_{10} = 1\,0010_2$$
$$0.4_{10} = 0.01100\,\overline{1100}_2$$
$$18.4_{10} = 1\,0010.01100\overline{1100}_2$$

   b) Normalisierung

$$18.4_{10} = 1\,0010.01100\,1100\ldots_2 \cdot 2^0 = 1.0010\,01100\,\overline{1100}_2 \cdot 2^4$$

   Aus $2^4 \rightarrow e = 4$

3. Exponent:
   a) Exponent $e = 4$, Biaswert = 127;
   b) $E = e + Biaswert = 4_{10} + 127_{10} = 131_{10} = 1000\,0011_2$

Die zusammengesetzte Gleitpunktzahl ist:
0 10000011 00101100110011001100110

# Codierungen

## ASCII (1963)

ASCII steht für „American Standard Code for Information Interchange“, 7 Bit, 95 druckbare + 33 Steuerzeichen = 128 Zeichen.

| | | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
|---|---|---|---|---|---|---|---|---|---|
| **0** | $0000_2$ | NUL | DLE | SP | 0 | @ | P | ‘ | p |
| **1** | $0001_2$ | SOH | DC1 | ! | 1 | A | Q | a | q |
| **2** | $0010_2$ | STX | DC2 | “ | 2 | B | R | b | r |
| **3** | $0011_2$ | ETX | DC3 | # | 3 | C | S | c | s |
| **4** | $0100_2$ | EOT | DC4 | $ | 4 | D | T | d | t |
| **5** | $0101_2$ | ENQ | NAK | % | 5 | E | U | e | u |
| **6** | $0110_2$ | ACK | SYN | & | 6 | F | V | f | v |
| **7** | $0111_2$ | BEL | ETB | ’ | 7 | G | W | g | w |
| **8** | $1000_2$ | BS | CAN | ( | 8 | H | X | h | x |
| **9** | $1001_2$ | TAB | EM | ) | 9 | I | Y | i | y |
| **A** | $1010_2$ | LF | SUB | * | : | J | Z | j | z |
| **B** | $1011_2$ | VT | ESC | + | ; | K | [ | k | { |
| **C** | $1100_2$ | FF | FS | , | < | L | \ | l | \| |
| **D** | $1101_2$ | CR | GS | - | = | M | ] | m | } |
| **E** | $1110_2$ | SO | RS | . | > | N | ^ | n | - |
| **F** | $1111_2$ | SI | US | / | ? | O | _ | o | DEL |

Beispiel Zeichen „j“ = 6A

## ISO 8859

8 Bit Erweiterung von ASCII. ISO 8859-1 = LATIN-1 (Europa).
Aufbau:
Position 0 ($00_{16}$) – 127 ($7F_{16}$) ASCII
Position 128 ($80_{16}$) – 159 ($9F_{16}$) Steuerzeichen
Position 160 ($A0_{16}$) – 255 ($FF_{16}$) regionale Sonderzeichen

## Unicode (1991)

Zeichensatz bestehend aus 17 Bereiche zu je 65 536 Zeichen.

### UTF-8

UTF-8 ist eine Codierungsform für Unicode, die 1 bis 4 Bytes verwendet:
Form 0xxx xxxx 1 Byte, codiert ist der ASCII Zeichensatz (128)
Form 110x xxxx 2 Byte, häufige Sonderzeichen ($2^{5+6} = 2\,048$)
Form 1110 xxxx 3 Byte, weniger häufige Zeichen($2^{4+6+6} = 65\,536$)
Form 1111 0xxx 4 Byte, selten Zeichen ($2^{3+6+6+6} = 2\,097\,152$)
Form 10xx xxxx Folgebyte
Jedes Zeichen, das mit mehr als einem Byte codiert ist, wird hat so viele Folgebytes, wie es selbst lang ist. (Bsp.
1110 xxxx 10xx xxxx 10xx xxxx)
UTF-16 und UTF-32 verwenden für jedes Zeichen 2 respektive 4 Bytes.

## Kompression

| | |
|---|---|
| Präfixfreiheit | Kein Wort ist der Anfang eines anderen Wortes (Fano-Bedingung) |
| Zyklisch | Wenn aus einem Code-Wort alle anderen gültigen Code-Wörter abgeleitet werden können. (Bsp: abc, bca, cab) |
| Hamming-Abstand | Anzahl unterschiedlicher Zeichen zweier gleichlanger Wörter |
| Hamming-Distanz | Minimum des Hamming-Abstandes aller Wörter eines Codes |
| Gray-Code | Benachbarte Wörter haben den Hamming-Abstand 1 |
| Gleichverteilung | Alle Wörter gleich $n$ Bits lang: $\lceil n \rceil = \log_2 \lvert Z \rvert$ |

Umwandlung einer Dezimalzahl ($d$) in Graycode:

1. Setze $s$ = Rechts-Shift von dezimal um 1 Bit
2. Setze $g$ = XOR($d$, $s$)
3. Gebe $g$ zurück

## Shannon-Fano-Algorithmus

Betrachte alle Zeichen als eine Gruppe.
Erstelle einen neuen Knoten auf der aktuellen Ebene.
Solange (Die Gruppe aus mehr als einem Zeichen besteht):

1. Sortiere die Zeichen nach ihrer Häufigkeit.
2. Teile die Zeichen entlang dieser Reihenfolge so in 2 Gruppen, dass die Summe der Häufigkeiten in den beiden Gruppen möglichst gleich ist.
3. Führe diesen Algorithmus mit jeder Gruppe aus.

Das Ergebnis weisst eine unnötig hohe Redundanz auf ($R = \frac{\text{mittlere Länge}}{\text{Entropie}} - 1$). Die Anzahl der Ebenen unterhalb der Wurzel, gibt die Anzahl zu verwendender Bits an.

## Huffmann-Algorithmus

Erstelle eine Wald mit einem Baum für jedes Zeichen.
Jeder dieser Bäume enthält nur einen einzigen Knoten: das Zeichen.
Schreibe die Häufigkeit an die Kante.
Wiederhole:

1. Suche die beiden Bäume, die die geringste Häufigkeit haben.
2. Fasse beide Bäume zusammen, indem sie die Teilbäume einer neuen Wurzel werden.
3. Benutze die Summe der Häufigkeiten dieses neuen Baumes zur weiteren Analyse.

bis nur noch ein Baum übrig ist.
Die Anzahl der Ebenen unterhalb der Wurzel, gibt die Anzahl zu verwendender Bits an.

## Arithmetische Codierung

Kodierer und Dekodierer einigen sich auf ein Intervall. Normalerweise wird hier das halboffene Intervall $[0, 1)$ verwendet. Kodierer und Dekodierer bei der De- bzw. Kodierung eines Zeichens immer identische Tabellen Wahrscheinlichkeitstabellen aller dekodierbaren Zeichen. Der Algorithmus für die Codierung sieht dann wie folgt aus:

1. Initialisiere das aktuelle Intervall mit dem vereinbarten Startintervall
2. Zerlege das aktuelle Intervall auf die identische Art wie der Dekodierer in Subintervalle, und weise jedem Subintervall ein Zeichen zu.
3. Das Subintervall, das dem nächsten Zeichen der Eingabe entspricht, wird zum aktuellen Intervall.
4. Sind noch weitere Zeichen zu kodieren, dann weiter bei Punkt 2. Sonst weiter beim nächsten Punkt.
5. Gib eine beliebige Zahl aus dem aktuellen Intervall und zusätzlich die Anzahl der kodierten Zeichen aus. Dieses ist die Zahl $x$, die vom Dekodierer wie oben beschrieben entschlüsselt werden kann. Diese Zahl wird so gewählt, dass sie möglichst wenig Nachkommastellen hat, also möglichst „rund“ ist und sich daher mit relativ wenigen Bits darstellen lässt.

Beispiel: Gegeben sei das Wort „bad“.
$p(a) = 0{,}65$ $p(b) = 0{,}15$ $p(c) = 0{,}10$
$p(d) = 0{,}04$ $p(e) = 0{,}04$ $p(f) = 0{,}02$

1. Startintervall sei $I_0 = [0, 1)$.
2. Aufgrund der Häufigkeiten gelten folgende Teil-Intervalle:
   $I_a = [0, 0{,}65)$ $I_b = [0{,}65, 0{,}80)$ $I_c = [0{,}80, 0{,}90)$
   $I_d = [0{,}90, 0{,}94)$ $I_e = [0{,}94, 0{,}98)$ $I_f = [0{,}98, 1{,}0)$
3. Erstes Zeichen $= b \rightarrow$ neues Startintervall $= I_0 = [0{,}65, 0{,}80)$
4. Aufgrund der Häufigkeiten gelten folgende Teil-Intervalle:
   $I_a = [0{,}65, 0{,}7475)$ $I_b = [0{,}7475, 0{,}77)$ $I_c = [0{,}77, 0{,}785)$
   $I_d = [0{,}785, 0{,}791)$ $I_e = [0{,}791, 0{,}797)$ $I_f = [0{,}797, 0{,}80)$
5. Zweites Zeichen $= a \rightarrow$ neues Startintervall $=$ $I_0 = [0{,}65, 0{,}7475)$
6. Aufgrund der Häufigkeiten gelten folgende Teil-Intervalle:
   $I_a = [0{,}65, 0{,}713375)$ $I_b = [0{,}713375, 0{,}728)$
   $I_c = [0{,}728, 0{,}73775)$ $I_d = [0{,}73775, 0{,}74165)$
   $I_e = [0{,}74165, 0{,}74555)$ $I_f = [0{,}74555, 0{,}7475)$
7. Drittes Zeichen $= d \rightarrow$ neues Startintervall $=$ $[0{,}73775, 0{,}74165)$
8. Da keine weiteren Zeichen vorhanden sind, eine möglichst kleine Zahl aus $[0{,}73775, 0{,}74165)$ wählen: $0{,}74 = \frac{74}{100}$

Dekodierung: Beispiel: Gegeben sei die Zahl 0,74 für ein Wort bestehend aus drei Zeichen. Welches Wort verbirgt sich darin?

1. Startintervall sei $I_0 = [0, 1)$.
2. Aufgrund der Häufigkeiten gelten folgende Teil-Intervalle:
   $I_a = [0, 0{,}65)$ $I_b = [0{,}65, 0{,}80)$ $I_c = [0{,}80, 0{,}90)$
   $I_d = [0{,}90, 0{,}94)$ $I_e = [0{,}94, 0{,}98)$ $I_f = [0{,}98, 1{,}0)$
3. 0,74 liegt im Teil-Intervall von $I_b = [0{,}65, 0{,}80) \rightarrow$ das erste Zeichen ist ein $b$.
4. Aufgrund der Häufigkeiten gelten folgende Teil-Intervalle:
   $I_a = [0{,}65, 0{,}7475)$ $I_b = [0{,}7475, 0{,}77)$ $I_c = [0{,}77, 0{,}785)$
   $I_d = [0{,}785, 0{,}791)$ $I_e = [0{,}791, 0{,}797)$ $I_f = [0{,}797, 0{,}80)$
5. 0,74 liegt im Teil-Intervall von $I_a = [0{,}65, 0{,}7475) \rightarrow$ das zweite Zeichen ist ein $a$.
6. Aufgrund der Häufigkeiten gelten folgende Teil-Intervalle:
   $I_a = [0{,}65, 0{,}713375)$ $I_b = [0{,}713375, 0{,}728)$
   $I_c = [0{,}728, 0{,}73775)$ $I_d = [0{,}73775, 0{,}74165)$
   $I_e = [0{,}74165, 0{,}74555)$ $I_f = [0{,}74555, 0{,}7475)$
7. 0,74 liegt im Teil-Intervall von $I_d = [0{,}73775, 0{,}74165) \rightarrow$ das dritte Zeichen ist ein $d$.
8. Das gesuchte Wort ist „bad“

Die Länge des gesuchten Wortes muss bekannt sein!

# Fehler

| | |
|---|---|
| Fehler | Ein Fehler liegt vor, wenn sich die gesendeten und empfangen bzw. abgespeicherte und ausgelesene $n$-Bits an mindestens einer Stelle unterscheiden (Bitweises XOR $\neq 0$) |
| Mehrfachfehler | Ein Mehrfachfehler von $x$-Bit liegt vor, wenn sich die gesendeten und empfangenen bzw. diesem abgespeicherten und ausgelesenen $n$-Bits an mindestens $x, x > 1$ Stellen unterscheiden. |
| Einzelbitfehler | statistisch voneinander unabhängig – das Auftreten eines Fehlers hängt nicht von anderen Fehlern ab. |
| Bündelfehler | (Blockfehler, Bursts) mehrere direkt aufeinanderfolgende Bits fehlerhaft sind – das Auftreten hängt von anderen/früheren Fehlern ab. |
| Synchronistationsfehler | lange Bündelfehler |

## Fehlerursachen

Typische physikalische Ursachen für Fehler sind:

- Rauschen (Wärme- oder Impulsrauschen) verursacht langfristig gleichmässig verteilte Einzelbitfehler.
- Dämpfung, Dispersion (Aufgrund von Leitungswiderstand, Dämpfung, Reflexion) verusacht Einzelbitfehler oder Bündelfehler.
- Strahlung (kosmische, radioaktive) verusacht langfristig gleichmässig verteilte Einzelbitfehler.
- Störungen (elektr. Funken, Kratzer) führen zu Bündelfehlern. Die Fehler sind ungleichmässig verteilt.
- Nebensprechen (kapazitive Kopplung benachbarter Datenleitungen) verursacht ungleichmässig verteilte Bündelfehler.

## Einfache Parität

Die einfache Parität wird auch einfach gebildet:

1. Die Code-Wörter werden um ein zusätzliches Bit, das Paritäts-Bit erweitert.
2. Der Wert dieses Bits wird bei:
   **gerader Parität (even parity)** so gesetzt, dass die Summe der Einsen im Code-Wort inkl. des Paritäts-Bits gerade ist. (Beispiel: Das Code-Wort sei 0100.1110 $\rightarrow$ 4 Einsen sind gerade $\implies$ Paritätsbit = 0)
   **ungerader Parität (ungerade)** so gesetzt, dass die Summe der Einsen im Code-Wort inkl. des Paritäts-Bits ungerade ist. (Beispiel: Das Code-Wort sei 0100.1110 $\rightarrow$ 4 Einsen sind gerade $\implies$ Paritätsbit = 1)

- Alle ungeraden-Bit-Fehler können erkannt werden
- Gerade-Bit-Fehler können nicht erkannt werden
- Bündelfehler können nur zu 50% erkannt werden
- Fehler können nicht korrigiert werden.

## Zweidimensionale Parität

Ansatz:

- Ein Block-Code aus $n$-Bits wird in Wörter gleicher Länge $l$ aufgeteilt.
- Jedes Wort wird um ein Paritätsbit ergänzt
- Jedes $i$-te Bit (inkl. Paritätsbit) jedes Wortes wird ebenfalls um ein Paritätsbit ergänzt.

Im Ergebnis:

- 1-Bit-Fehler werden doppelt erkannt (Zeile und Spalte) und können korrigiert werden
- ungerade-Bit-Fehler werden erkannt, können aber nicht korrigiert werden (Maskierung)
- 2, 6, 10, ...-Bit-Fehler werden erkannt (Zeile oder Spalte), können aber nicht korrigiert werden
- Vielfache von 4-Bit-Fehler werden in aller Regel erkannt. Es sei denn, sie maskieren sich doppelt, was nur extrem selten (Bsp. bei einer Anordnung im Viereck) der Fall ist
- Bündelfehler werden nur Ausnahmsweise nicht erkannt.

## Blockcode - CRC

$(n, k)$-Block Code bedeutet, dass für $k$ übertragene Nutzdatenbits $n - k$ Prüfbits und somit insgesamt $n$-Bits übertragen werden.

### Algorithmus

Eine Bitfolge von $n$-Bits wirt als Nachrichtenpolynom vom Grad $n - 1$ dargestellt:

$$I(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1x^1 + a_0x^0 = \sum_{i=0}^{n-1} a_i \cdot x^i$$

Ein Generatorpolynom $C(x)$ mit Grad $n - k, n > k, n \in \mathbb{N}$ wird zwischen Sender und Empfänger vereinbart.
Der genaue Algorithmus zur Übermittlung einer Nachricht $I(x)$ und dem Generatorpolynom $C(x)$ vom Grad $n - k$ liesst sich folgendermassen:

1. Sender erzeugt $N(x)$:
   a) $M(x) = I(x) \cdot x^{n-k}$
   b) $R_s(x) = I(x) \mod C(x)$
   c) $N(x) = (M(x) + R_s(x))$
2. Sender sendet $N(x)$
3. Empfänger berechnet: $R_e = N(x) \mod C(x)$ und entscheidet:
   - Falls $R_e = 0$: Kein Fehler, also $I(x) = M(x) + R_s(x)$
   - Falls $R_e \neq 0$: Ein Fehler ist aufgetreten, also nochmal senden

**Ungerade-Bit-Fehler** können sicher erkannt werden, wenn das Generatorpolynom den Term $x - 1$ enthält

**Gerade-Bit-Fehler** werden erkannt, deren Polynomdarstellung einen kleineren Grad als das Generatorpolynom hat, Sie werden nur dann nicht erkannt, wenn sie ein Vielfaches des Generatorpolynoms sind

**Bündelfehler** werden erkannt, wenn ihre Länge $l \leq n - k$

# Automaten

Informale Spezifikation können unvollständig, redundant und Mehrdeutig sein. Besser: Spezifikation über Zustandstabelle oder UML-Diagramm (Zustandsübergangsdiagramm) für endliche Automaten.

Für endliche Automaten gilt:

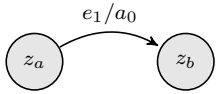
| | |
|---|---|
| Zustände | $Z = \{z_0, z_1, \ldots, z_n\}$ |
| Eingaben | $E = \{e_0, e_1, \ldots, e_o\}$ |
| Ausgaben | $A = \{a_0, a_1, \ldots, a_p\}$ |
| Übergangsfunktionen | $f : Z \times E \to Z \times A$ |
| Anfangszustand | $z_0$ |
| Gültiger Endzustand | $z_0$ |

Ereignis-Tabelle:

| Zustand | Eingabe | Folgezustand | Ausgabe |
|---|---|---|---|
| $z_0$ | $e_0$ | $z_1$ | $a_0$ |
| ... | ... | ... | ... |

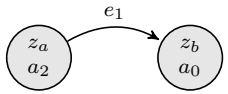
**Mealy-Automat**

Die Ausgabe wird mit der Transition assoziiert, dass heisst, sie hängt vom Zustand und der Eingabe ab.




**Moore-Automat**

Die Ausgabe wird mit dem Zustand assoziiert und hängt somit nur vom Zustand ab.




# Graphen

Ein Graph ist ein Tupel $G = \{V, E\}$ mit der Knotenmenge $V$ und der Kantenmenge $E$, wobei $e_i = \{V_j, V_k\}$. Di-Graphen haben gerichtete Kanten.

| | |
|---|---|
| Vollständigkeit | Von jedem Knoten existiert ein Kante zu jedem anderen Knoten ($\frac{\lvert V\rvert \cdot (\lvert V\rvert - 1)}{2}$) |
| Verbunden | Zwei Knoten sind verbunden, wenn eine Folge von Knoten und Kanten existiert, mit der man von einem Konten zum anderen gelangt |
| Zusammenhängend | Jeder Knoten ist mit jedem anderen Knoten verbunden |
| Schleife | Sind Knoten über mehr als eine nicht-identische Kantenfolge verbunden, liegt eine Schleife vor |

## Bäume

Haben eine Wurzel-Knoten, der nur gerichtete Kanten hat. Jeder handere Knoten hat nur eine einlaufende Kante. Binäre Bäume haben pro Knoten höchstens zwei auslaufende Kanten.

| | |
|---|---|
| Blatt | Knoten ohne auslaufende Kanten |
| Höhe | Maximum der Kanten zwischen Wurzel und Blättern |
| Minimale Höhe | Es gibt keinen anderen Baum mit geringerer Höhe |
| Ausgewogenheit | Es existiert höchstens ein Knoten mit nur einer auslaufenden Kante |

# Logik

## Aussagenlogik

**Aussage** Eine Aussage ist eine sprachliche Äusserung, die wahr ($w$) oder falsch($f$) sein kann.
Bsp: „Es gibt unendlich viele natürliche Zahlen."

### Junktoren (Verknüpfungsoperatoren)

| | |
|---|---|
| $\neg$ | Negation, „nicht ..." |
| $\wedge$ | Konjunktion, „... und ..." |
| $\vee$ | Disjunktion, „... oder ..." |
| $\Rightarrow$ | Implikation, „wenn ..., dann ..." |
| $\Leftrightarrow$ | Äquivalenz, „... genau dann, wenn ..." |
| $\bigoplus$ | Antivalenz, „entweder ... oder ..." |

Für eine einzelne Aussage $A$ gilt:

| $A$ | $\neg A$ |
|---|---|
| $f$ | $w$ |
| $w$ | $f$ |

Für zwei Aussagen $A$ und $B$ gilt:

| $A$ | $B$ | $A \wedge B$ | $A \vee B$ | $A \Rightarrow B$ | $A \Leftrightarrow B$ | $A \bigoplus B$ |
|---|---|---|---|---|---|---|
| $f$ | $f$ | $f$ | $f$ | $w$ | $w$ | $f$ |
| $f$ | $w$ | $f$ | $w$ | $w$ | $f$ | $w$ |
| $w$ | $f$ | $f$ | $w$ | $f$ | $f$ | $w$ |
| $w$ | $w$ | $w$ | $w$ | $w$ | $w$ | $f$ |

Bindungsstärke: $\neg$ vor $\wedge$ vor $\vee$ vor $\Rightarrow$ vor $\Leftrightarrow$.

### Rechenregeln

| | | |
|---|---|---|
| Duplizität | $A \wedge A$ | $\Leftrightarrow A$ |
| | $A \vee A$ | $\Leftrightarrow A$ |
| Doppelte Negation | $\neg\neg A$ | $\Leftrightarrow A$ |
| Kommutativität | $A \wedge B$ | $\Leftrightarrow B \wedge A$ |
| | $A \vee B$ | $\Leftrightarrow B \vee A$ |
| Assoziativität | $(A \wedge B) \wedge C$ | $\Leftrightarrow A \wedge (B \wedge C)$ |
| | $(A \vee B) \vee C$ | $\Leftrightarrow A \vee (B \vee C)$ |
| Distributiviät | $A \wedge (B \vee C)$ | $\Leftrightarrow (A \wedge B) \vee (A \wedge C)$ |
| | $A \vee (B \wedge C)$ | $\Leftrightarrow (A \vee B) \wedge (A \vee C)$ |
| De Morgan Regeln | $\neg(A \wedge B)$ | $\Leftrightarrow \neg A \vee \neg B$ |
| | $\neg(A \vee B)$ | $\Leftrightarrow \neg A \wedge \neg B$ |
| Implikation | $(A \Rightarrow B)$ | $\Leftrightarrow (\neg A \vee B)$ |
| Kontraposition | $(A \Rightarrow B)$ | $\Leftrightarrow (\neg B \Rightarrow \neg A)$ |
| Äquivalenz | $(A \Leftrightarrow B)$ | $\Leftrightarrow (A \Rightarrow B) \wedge (B \Rightarrow A)$ |
| Absorbtion | $A \wedge (A \vee B)$ | $\Leftrightarrow A$ |
| | $A \vee (A \wedge B)$ | $\Leftrightarrow A$ |

Die Rechenregel der Aussagenlogik werden mit Hilfe von Wahrheitstafeln bewiesen.

**Erfüllbarkeit** Es gibt eine Interpretation der Variablen, so dass die Formel wahr wird.

**Allgemeingültigkeit** Ein Formel wird mit jeder Interpretation der Variablen wahr.

**Widerlegbarkeit** Es gibt mindestens eine Interpretation der Variablen, so dass die Formel falsch wird.

**Entscheidbarkeit** Man kann wissen, bei welcher Variablen-Kombination eine Formel wahr wird.

**Konjunktive Normalform (KNF)** Konjunktion von Disjunktionstermen $(a \wedge b) \vee (c \wedge b)$

**Kanonisch konjunktive Normalform (KKNF)** (Minterm) Jede Variable tritt pro KNF-Term genau einmal auf.

**Disjunktive Normalform (DNF)** Disjunktion von Konjunktionstermen $(a \vee b) \wedge (c \vee d)$

**Kanonisch disjunktive Normalform (KDNF)** (Maxterm) Jede Variable tritt pro DNF-Term genau einmal auf.

Normalformen werden über Wahrheitstafeln oder durch Term-Umformung konstruiert.
Für die KDNF alle Belegungen einer Wahrheitstafel, die $w$ ergeben konjunktiv verknüpfen. Einzelterme disjunktiv verknüpfen. Für die KKNF alle Belegungen einer Wahrheitstafel, die $f$ ergeben disjunktiv verknüpfen. Einzelterme konjunktiv verknüpfen.

### Beispiel Formelumwandlung zu DNF/KDNF

Gegeben $F = ((a_1 \Rightarrow a_2) \Rightarrow \neg a_3) \vee \neg a_2$
1. Umrechung in DNF:

$$
\begin{aligned}
F &= ((a_1 \Rightarrow a_2) \Rightarrow \neg a_3) \vee \neg a_2 \\
(a_1 \Rightarrow a_2) &\Leftrightarrow \neg a_1 \vee a_2 && \text{Impl.} \\
((\neg a_1 \vee a_2) \Rightarrow \neg a_3) &\Leftrightarrow (\neg(\neg a_1 \vee a_2)) \vee \neg a_3) && \text{Impl.} \\
&\Leftrightarrow \neg(a_2 \vee \neg a_1) \vee \neg a_3 && \text{Assoz.} \\
\neg(a_2 \vee \neg a_1) &\Leftrightarrow \neg a_2 \wedge \neg\neg a_1 && \text{De Morgan} \\
&\Leftrightarrow \neg a_2 \wedge a_1 && \text{D. Vernein.} \\
((\neg a_2 \wedge a_1) \vee \neg a_3) \vee \neg a_2 &\Leftrightarrow (\neg a_2 \wedge a_1) \vee \neg a_3 \vee \neg a_2 && \text{Absorbtion} \\
&\Leftrightarrow \neg a_2 \wedge \neg a_3
\end{aligned}
$$

2. KDNF erstellen:

| $a_1$ | $a_2$ | $a_3$ | $F_2$ | Ausdruck |
|---|---|---|---|---|
| $w$ | $w$ | $w$ | $f$ | |
| $w$ | $w$ | $f$ | $w$ | $a_1 \wedge a_2 \wedge \neg a_3$ |
| $w$ | $f$ | $w$ | $w$ | $a_1 \wedge \neg a_2 \wedge a_3$ |
| $w$ | $f$ | $f$ | $w$ | $a_1 \wedge \neg a_2 \wedge \neg a_3$ |
| $f$ | $w$ | $w$ | $f$ | |
| $f$ | $w$ | $f$ | $w$ | $\neg a_1 \wedge a_2 \wedge \neg a_3$ |
| $f$ | $f$ | $w$ | $w$ | $\neg a_1 \wedge \neg a_2 \wedge a_3$ |
| $f$ | $f$ | $f$ | $w$ | $\neg a_1 \wedge \neg a_2 \wedge \neg a_3$ |

3. Ergebnis zusammensetzen:

$$
\begin{aligned}
F_2 :&(a_1 \wedge a_2 \wedge \neg a_3) \vee (a_1 \wedge \neg a_2 \wedge a_3) \vee (a_1 \wedge \neg a_2 \wedge \neg a_3) \\
&\vee (\neg a_1 \wedge a_2 \wedge \neg a_3) \vee (\neg a_1 \wedge \neg a_2 \wedge a_3) \vee (\neg a_1 \wedge \neg a_2 \wedge \neg a_3)
\end{aligned}
$$

## Prädikatenlogik

**Aussageform** Eine Aussageform ist eine sprachliche Äusserung, in der Variablen vorkommen und die in Abhängigkeit der Variablenwerte wahr ($w$) oder falsch ($f$) sein kann – Aussageformen sind manchmal wahr, manchmal falsch.
Bsp: „Die Zahl $x$ ist eine gerade Zahl."
Es wird unterschieden zwischen Objekt und Prädikat (Eigenschaft): für oberes Beispiel ist „ist gerade" das Prädikat, während $x$ das Objekt ist.

### Quantoren (Variablenbinder)

$A(x)$ sei eine Aussageform, $M$ eine Menge von Objekten.

| | |
|---|---|
| $\forall x \in M A(x)$ | Für alle $x$ der Menge $M$ gilt $A(x)$ |
| $\exists x \in M A(x)$ | Für mindestens ein $x$ der Menge $M$ gilt $A(x)$ |

### Besondere Ausdrücke

| | |
|---|---|
| Für mindestens zwei gilt | $\exists x \exists y((A(x) \wedge A(y)) \Rightarrow (x \neq y))$ |
| Es gibt höchstens ein | $\forall x \forall y ((A(x) \wedge A(y)) \Rightarrow (x = y))$ |


Revision: 1.0, Datum: 23. Juni 2013