Fachhochschule Köln
University of Applied Sciences
Cologne

Abteilung Gummersbach

20 Fachbereich
Informatik

Prof. Dr. Heribert Koch

Physik / Technische Informatik

Ausarbeitung zur Vorlesung

Theoretische Informatik

WS ´00/01

0 Einleitung

0.1 Teilnehmerkreis

Diese einführende Veranstaltung *Theoretische Informatik* richtet sich an Studierende im ersten Semester der Studiengänge Technische Informatik, Allgemeine Informatik und Medien Informatik.

0.2 Eingangsvoraussetzungen

Der Kurs setzt grundsätzlich nur einfache Kenntnisse aus der naiven Mengenlehre voraus, wie sie in der Schule vermittelt und bei der mathematischen Begriffsbildung verwendet werden. Wesentlich für ein aktives Verständnis dieser Einführung in die Theoretische Informatik ist eine gewisse Erfahrung mit mathematischem Denken und mathematischen Formeln.

0.3 Übersicht: Themengebiete und Aufbau des Kurses

Im Rahmen der Veranstaltung werden wesentliche Basisbausteine der Theoretischen Informatik thematisiert und durch anwendungsorientierte Problemstellungen präzisiert. Die Begriffsbildungen der Theoretischen Informatik werden hier als grundlegende Hilfsmittel für die Lösung praktischer Aufgaben dargestellt und durchgängig über konkrete Anwendungen motiviert. Selbstverständlich ist es nicht möglich, mit einer wöchentlich zweistündigen Vorlesung alle Teilgebiete der Theoretischen Informatik angemessen anzusprechen oder die in dieser Einführung ausgewählten Gebiete umfassend zu behandeln; dies bleibt weiterführenden Veranstaltungen höherer Semester vorbehalten.
Als grundlegende Gebiete der Theoretischen Informatik für eine fundierte Informatik - Ausbildung haben sich die folgenden Themen, die die Gliederung und Auswahl des Kurses widerspiegeln, bewährt:

1. Zahlensysteme, Zahlendarstellung, Numerische Aspekte,
2. Codierung, Informationstheorie,
3. Aussagenlogik,
4. Prädikatenlogik,
5. Boolesche Algebra,
6. Schaltnetze und Schaltwerke,
7. Endliche Automaten,
8. Reguläre Sprachen.

Der Kurs ist zweigliedrig als Vorlesung und Übung aufgebaut, wobei die Übungen für Studierende im ersten Semester selbstredend eine herausragende Bedeutung besitzen.

0.4 Groblehrziele

Grundsätzliches Ziel des Kurses ist eine Einführung in die Begriffe, Methoden, Modelle und Arbeitsweise der Theoretischen Informatik anhand der ausgewählten Teilgebiete. Gleichzeitig bilden diese Themengebiete eine wesentliche Basis und Vorbereitung für Veranstaltungen in höheren Semestern des Studiums.

0.5 Literaturliste

Die nachfolgende Liste stellt eine subjektive Auswahl deutschsprachiger Titel zu den Themengebieten des Kurses dar: 'Stöbern' Sie bitte in der Bibliothek und Sie werden überrascht sein, wie gut diese ausgestattet ist und was sie Ihnen alles zu bieten hat!

Wilhelm, R. (1996): *Informatik*. C. H. Beck, München.

Blieberger, J. et al. (1996): *Informatik.* Springer-Verlag, Wien.

Rembold, U. et al. (1991): *Einführung in die Informatik.* 2. Aufl. Hanser, München.

Rembold, U. et al. (1990): *Aufgaben zur Informatik.* Hanser, München.

Goos, G. (1995): *Vorlesungen über Informatik , Bd. 1.* Springer, Heidelberg.

Sedgewick, R. (1992): *Algorithmen in C++*. Addison-Wesley, Bonn.

Bauer, F.L. und Goos, G. (1991): *Informatik 1.* 4.Aufl. Springer, Heidelberg.

Merzenich, W., Zeidler, H. C. (1997): *Informatik für Ingenieure.* B. G. Teubner, Stuttgart.

Ehrig, H. et al. (1999): *Mathematisch-strukturelle Grundlagen der Informatik.* Springer,

Heidelberg.

Brauch, W.,Dreyer, H. und Haacke, W. (1990): *Mathematik für Ingenieure.* B.G. Teubner,

Stuttgart.

Böhme, G. (1992): *Algebra.* 7.Aufl. Springer, Berlin.

Böhme, G. (1993): *Fuzzy-Logik.* Springer, Berlin, Heidelberg.

Schöning, U. (1992): *LOGIK FÜR INFORMATIKER.* 3. Aufl. BI-Wiss.- Verlag, Mannheim.

Matthiessen, G. (1991): *Logik für Software-Ingenieure.* de Gruyter, Berlin.

Schiffmann, W. und Schmitz, R. (1993): *Technische Informatik 1.* 2.Aufl. Springer, Heidelberg.

Urbanski, K. und Woitowitz, R. (1993): *Digitaltechnik.* BI-Wiss.- Verlag, Mannheim.

Morgenstern, B. (1992): Elektronik III, *Digitale Schaltungen und Systeme.* Vieweg&Sohn, Braunschweig.

Tietze, U. und Schenk, C. (1990): *Halbleiter-Schaltungstechnik.* 9.Aufl. Springer, Berlin.

Beuth, K. (1992): *Digitaltechnik.* 9.Aufl.Vogel, Würzburg.

Schöning, U. (1997): *Theoretische Informatik - kurzgefaßt.* 3. Aufl. Spektrum Akademischer Verlag, Heidelberg.

Vossen, G., Witt K. (2000): *Grundlagen der Theoretischen Informatik mit Anwendungen.* Vieweg & Sohn, Braunschweig.

Albert, J., Ottmann Th. (1987): *Automaten, Sprachen und Maschinen für Anwender*. Bibliographisches Institut, Mannheim.

0.6 Studienhinweise

Der Kurs wird als zweistündige Vorlesung mit zweistündigen Übungen angeboten. Die Vorlesungen sind derart angelegt, dass Sie diesen leicht folgen und den behandelten Stoff ohne große Mühe nachvollziehen können, vorausgesetzt Sie beachten den fortschreitenden und aufeinander aufbauenden Charakter der Vorlesungen. Eine Sache begreifen heisst Begriffe geben, d.h. Sie werden in diesen - wie in den meisten anderen - Veranstaltungen mit vielen neuen Begriffen konfrontiert und Sie sollten immer die Beispiele nacharbeiten und möglichst anhand kleiner Veränderungen neue Beispiele konstruieren und hiermit die Begriffe vertiefen. In Ihrem ersten Studiensemester ist ein Nacharbeiten der Vorlesungen unabdingbar, weil Sie hierbei Ihre Kenntnisse festigen und überprüfen, insbesondere aber nicht verstandene Argumente und Verständnisschwierigkeiten leicht erkennen können - letztere lassen sich dann in den Übungen effektiv beheben. Neben dem eigentlichen Stoff, der die tragenden Gesetzmässigkeiten umfasst, beinhaltet der Kurs selbstverständlich unumgängliche technische und methodische Abschnitte in denen Arbeitstechniken im Mittelpunkt stehen; in Diskussionsteilen und Zusammenfassungen werden Sie aber stets darauf hingewiesen, worauf es wirklich ankommt! Allerdings kann alle Hilfestellung erst den gewünschten Erfolg bieten, wenn Sie sich mit dem Kurs intensiv beschäftigen.
Das vorlesungsbegleitende Skript bietet Ihnen die Möglichkeit, die Ausführungen wiederholt nachzuvollziehen und die wesentlichen Gedanken insbesondere mit Hilfe der Beispiele einzuüben; dieses Skript kann und möchte aber keineswegs das Studium der angegebenen Bücher ersetzen - das Lesen und Bearbeiten unterschiedlicher Darstellungen eines Themas schärft Ihren Blick für das Wesentliche und fördert neben dem Abstraktionsvermögen besonders Ihre Fähigkeit zum logischen Denken.

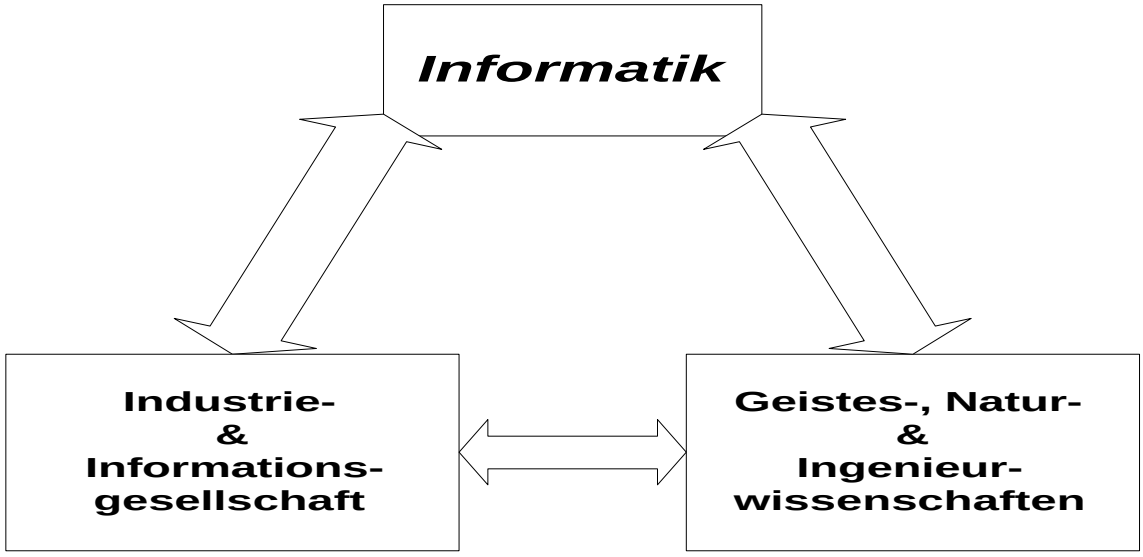
0.7 Übungen

Wie bereits betont bilden Übungen wesentliche Säulen des Kurses, die für Sie von herausragender Bedeutung sind. Hier werden wir handlungsorientiert und teilnehmerzentriert arbeiten und relativ leicht zu lösende Übungsaufgaben bearbeiten, Arbeitstechniken einüben und Problemlösungen diskutieren. Nutzen Sie insbesondere auch die Übungen extensiv als Frageforum! Dass die Übungen Ihnen eine ständige Überprüfung Ihres Kenntnisstandes sowie eine adäquate Klausurvorbereitung liefern, ist Ihnen wahrscheinlich klar und ebenso, dass Sie hier durch aktives Engagement Ihr Studium erfolgreich bereichern können.

0.8 Einführung

Was ist Informatik ?

Alles, das mit der elektronischen Datenverarbeitung zu tun hat, wird unter den Begriff Informatik subsumiert: Informatik ist die Wissenschaft von der systematischen Darstellung, Verarbeitung, Speicherung und Übertragung von Information.



Die Informatik besitzt Eigenschaften einer Grundlagenwissenschaft - vergleichbar mit der Mathematik, der Physik oder der Philosophie - , da sie grundlegende Begriffe wie Information, Berechnung und Algorithmus, Sprache, Komplexität, Zufall, etc. begründet und mit ihren Formalismen in fast alle Wissensgebiete hineinwirkt.

Insbesondere besitzt die Informatik Eigenschaften einer Ingenieurwissenschaft, denn sie befaßt sich mit dem Entwurf, der Konstruktion und der Realisierung von komplexen Systemen.

Gleichfalls zeigt die Informatik wesentliche Eigenschaften einer Naturwissenschaft, indem sie modellierte "Welten" analysiert und simuliert.

Gliederung der Informatik

Theoretische Informatik

- Grundsätzliche Fragen
- Theorie der formalen Sprachen
- Automatentheorie
- Algorithmen
- Komplexitätstheorie
- Berechenbarkeit
- .....

Praktische Informatik

- Systemanalyse
- Modellbildung
- Verteilte Systeme
- Bildverarbeitung
- Softwaretechnik, Softwarewerkzeuge
- Künstliche Intelligenz
- Computeralgebra
- .....

Technische Informatik

- Hardwareorientierte Fragestellungen
- hochintegrierte Schaltungen
- Rechnervernetzung
- Prozeßdatenverarbeitung
- Robotik
- Rechnerorganisation
- .....

Als Ingenieurwissenschaft zeichnet sich die Informatik durch einen starken Bezug zur Praxis aus. In der heutigen Zeit mit stetig steigenden Anforderungen bildet insbesondere ein fundiertes Wissen der theoretischen Grundlagen die unverzichtbare Basis für ein erfolgreiche Tätigkeit - einige der unentbehrlichen Grundlagen stehen im Mittelpunkt dieser Veranstaltung.

1 Zahlensysteme

1 Zahlensysteme

1.1 Polyadische Zahlensysteme

Die Zahlensysteme der Informatik und Digitaltechnik Dual-, Oktal-, Dezimal- und Sedezimalsystem sind polyadische Zahlensysteme (auch B-adische oder Radixsysteme genannt).

> *In einem polyadischen Zahlensystem mit der Basis B wird eine Zahl Z als Potenzreihenentwicklung von B dargestellt*:
>
> $$Z = \sum_{i=-\infty}^{n} b_i \, B^i \ ,$$
>
> $B \in \mathbb{N}$ *heißt Basis des Zahlensystems* ($B \geq 2$) ;
> *die* b_i *werden als Ziffern der Zahl bezeichnet*
> – *Eindeutigkeit der Darstellung erfordert* $0 \leq b_i < B$.

Ganze Zahlen ($b_i = 0 \ , \ i < 0$)

Potenzschreibweise: $Z = \sum_{i=0}^{n} b_i \, B^i$

Stellenschreibweise: $Z = b_n b_{n-1} b_{n-2} \ldots\ldots b_3 b_2 b_1 b_0$

Gebrochene Zahlen
Zwischen die Ziffern b_0 und b_{-1} wird ein Trennzeichen eingefügt.

Potenzschreibweise: $Z = \sum_{i=-m}^{n} b_i \, B^i$

Stellenschreibweise: $Z = b_n b_{n-1} \ldots b_2 b_1 b_0 \ , \ b_{-1} b_{-2} b_{-3} \ldots$

Beispiel: "Zweihunderteins"

B = 2 (dual)

$Z = 1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$
$Z = (11001001)_2$

B = 8 (oktal)

$Z = 3 \cdot 8^2 + 1 \cdot 8^1 + 1 \cdot 8^0$
$Z = (311)_8$

B = 10 (dezimal)

$Z = 2 \cdot 10^2 + 0 \cdot 10^1 + 1 \cdot 10^0$
$Z = (201)_{10}$

B = 12 (duodezimal)

$Z = 1 \cdot 12^2 + 4 \cdot 12^1 + 9 \cdot 12^0$
$Z = (149)_{12}$

B = 16 (sedezimal)

$Z = 12 \cdot 16^1 + 9 \cdot 16^0$
$Z = (C9)_{16}$ [Z = C9H] #

Eine Ziffer ist ein Zeichen aus einem Zeichenvorrat von N Zeichen, denen als Zahlenwert die ganzen Zahlen 0, 1, 2, ... N-1 eindeutig zugeordnet sind.
Bezeichnung:
Dualziffern (N=2), Oktalziffern (N=8), Dezimalziffern (N=10), Duodezimalziffern (N=12), Sedezimalziffern (N=16).

Besitzt ein Zahlensystem eine Basis B > 10, so werden neben den gewohnten Ziffern 0, 1, 2, ..., 9 zusätzlich weitere Ziffern benötigt.

B = 2	B = 5	B = 8	B = 10	B = 12	B = 16
0	0	0	0	0	0
1	1	1	1	1	1
10	2	2	2	2	2
11	3	3	3	3	3
100	4	4	4	4	4
101	10	5	5	5	5
110	11	6	6	6	6
111	12	7	7	7	7
1000	13	10	8	8	8
1001	14	11	9	9	9
1010	20	12	10	A	A
1011	21	13	11	B	B
1100	22	14	12	10	C
1101	23	15	13	11	D
1110	24	16	14	12	E
1111	30	17	15	13	F
10000	31	20	16	14	10
10001	32	21	17	15	11

Die Anzahl der erforderlichen Ziffern zur Zahlendarstellung wächst bei kleiner Basis relativ schnell an und vermindert die Lesbarkeit.

Beispiel: Darstellung der Zahl Z = $(123)_{10}$ in den polyadischen Systemen mit B = 2, 8, 12, 16

n	2^n	8^n	12^n	16^n
0	1	1	1	1
1	2	8	12	16
2	4	64		
3	8			
4	16			
5	32			
6	64			

B = 2: $Z = 1\cdot2^6 + 1\cdot2^5 + 1\cdot2^4 + 1\cdot2^3 + 0\cdot2^2 + 1\cdot2^1 + 1\cdot2^0$
$Z = (1111011)_2$

B = 8: $Z = 1\cdot8^2 + 7\cdot8^1 + 3\cdot8^0$
$Z = (173)_8$

B = 12: $Z = 10\cdot12^1 + 3\cdot12^0$
$Z = (A3)_{12}$

B = 16: $Z = 7\cdot16^1 + 11\cdot16^0$
$Z = (7B)_{16}$ #

Ganze Zahlen lassen sich in einem B-adischen System einfach darstellen; für beliebige reelle Zahlen gilt:

Jede reelle Zahl läßt sich in einem B-adischen System durch einen unendlichen B-adischen Bruch darstellen. Rationale Zahlen besitzen im B-adischen System einen endlichen oder einen periodischen B-adischen Bruch.

Irrationale Zahlen sind in polyadischen Systemen nur näherungsweise darstellbar - z. B. ist die vollständige Dezimaldarstellung der Zahl B = 3,141592... unbekannt.

Beispiel: B-adische Brüche

Dezimalsystem

$7/5 = (1,4)_{10}$ $2/3 = (0,\overline{6})_{10}$ $3/4 = (0,75)_{10}$

Oktalsystem

$7/5 = (1,\overline{3146})_8$ $4/7 = (0,\overline{4})_8$ $5/4 = (1,2)_8$

Dualsystem

$15/8 = (1,111)_2$ $7/5 = (1,\overline{0110})_2$ $3/4 = (0,11)_2$ #

Der im Dezimalsystem endliche Bruch 7/5 ist im Dualsystem ein periodischer Bruch; 3/4 dagegen ist in beiden Systemen als endlicher B-adischer Bruch darstellbar.

Ein endlicher B-adischer Bruch entsteht aus dem echten Bruch p/q, wenn q nur solche Primfaktoren besitzt wie sie auch in der Basis B des Zahlensystems auftreten.

Negative reelle Zahlen werden in einem polyadischen Zahlensystem mit Hilfe des vorangestellten Vorzeichens "-" gekennzeichnet.

Beispiel: Größte darstellbare Zahl

Die größte darstellbare Zahl für eine fixierte Stellenzahl n in einem polyadischen Zahlensystem mit der Basis B ergibt sich zu:

$$Z = (B-1)\cdot B^0 + (B-1)\cdot B^1 + \ldots + (B-1)\cdot B^{n-1} =$$
$$= B^1 - B^0 + B^2 - B^1 + \ldots + B^n - B^{n-1} = B^n - B^0 = B^n - 1 ;$$

sei B = 2 , n = 8: $Z = 2^8 - 1 = (255)_{10}$. #

1.2 Dualsystem

Das Dualsystem ist das polyadische Zahlensystem mit der Basis 2.
Darstellung einer Zahl:

Potenzschreibweise: $Z = \sum_{i=-\infty}^{n} b_i \cdot 2^i \quad , \quad 0 \le b_i < 2$

Stellenschreibweise: $Z = b_n.....b_3\ b_2\ b_1\ b_0\ ,\ b_{-1}\ b_{-2}\ b_{-3}.....$

2^n
2^3
2^2
2^1
2^0

2^{-3}
2^{-2}
2^{-1}

2^n	n	2^{-n}
1	0	1,0
2	1	0,5
4	2	0,25
8	3	0,125
16	4	0,062 5
32	5	0,031 25
64	6	0,015 625
128	7	0,007 812 5
256	8	0,003 906 25
512	9	0,001 953 125
1 024	10	0,000 976 562 5
2 048	11	0,000 488 281 25
4 096	12	0,000 244 140 625
8 192	13	0,000 122 070 312 5
16 384	14	0,000 061 035 156 25
32 768	15	0,000 030 517 578 125

Das Dualsystem stellt mit den Ziffern $b_i \in \{0,1\}$ ein spezielles Binärsystem dar. Binär heißt ganz allgemein zweier Werte fähig sein und ein Binärzeichen ist jedes beliebige Zeichen aus einem zweielementigen Zeichenvorrat. Als Binärzahl wird jede Zahl bezeichnet, die aus einer Folge der binären Informationseinheit 1 bit (binary digit) aufgebaut ist - z. B. binärverschlüsselte Dezimalzahlen (BCD).

Arithmetik im Dualsystem

□ Addition

Augend	+	Addend	=	Summe	Übertrag (Carry)
0	+	0	=	0	
0	+	1	=	1	
1	+	0	=	1	
1	+	1	=	0	1

Der Übertrag wird analog zur dezimalen Addition auf die folgende höherwertige Stelle addiert.

dual		dezimal	
0010,1		2,5	
+ 1010,1		+ 10,5	
--------		------	
1 1		*1*	*Übertrag*
--------		------	
1101,0		13,0	

□ Subtraktion

Minuend	-	Subtrahend	=	Differenz	Borger (Borrow)
0	-	0	=	0	
0	-	1	=	1	1
1	-	0	=	1	
1	-	1	=	0	

dual		dezimal	
101011		43	
- 100111		- 39	
--------		----	
1		*1*	*Borger*
--------		----	
000100		04	

Diese Art der Subtraktion wird in Mikroprozessoren nicht realisiert; Addition und Subtraktion werden mit einem Addierwerk durchgeführt indem die Subtraktion durch Komplementbildung auf eine Addition zurückgeführt wird - s. u. .

□ **Multiplikation**

Multiplikand	·	Multiplikator	=	Produkt
0	·	0	=	0
0	·	1	=	0
1	·	0	=	0
1	·	1	=	1

```
10001  · 1100
10001
 10001
  00000
   00000
11001100
```

$(17)_{10} \cdot (12)_{10} = (204)_{10}$

Die Multiplikation korrespondiert zur fortgesetzten Addition.

Verschieben einer Dualzahl um eine Stelle nach links/rechts liefert deren Verdoppelung/Halbierung!

□ **Division**

Dividend	:	Divisor	=	Quotient	+	Rest
0	:	0	=	nicht definiert		
0	:	1	=	0		
1	:	0	=	nicht definiert		
1	:	1	=	1		

$(299)_{10} : (13)_{10} = (23)_{10}$

```
 100101011  :  1101  =  10111
-1101
  10110
 -1101
   10011
  -1101
    1101
   -1101
       0
```

Die Division korrespondiert zur fortgesetzten Subtraktion.

Die Zahlendarstellung im Dualsystem ist relativ umständlich und nicht sehr praktikabel; eine Dualzahl besitzt durchschnittlich 3,3-mal soviele Stellen wie die entsprechende Dezimalzahl.

$(1000000111110)_2 = (4158)_{10}$

1.3 Oktalsystem und Sedezimalsystem

Zahlendarstellungen im Oktal- bzw. Sedezimalsystem sind eng mit der Darstellung im Dualsystem verknüpft; in diesen Systemen werden mehrere Dualstellen (3-bit bzw. 4-bit) zu einer Ziffer zusammengefaßt.

Das Oktalsystem ist das polyadische Zahlensystem mit der Basis B=8:

$$Z = \sum_{i=-\infty}^{n} b_i \cdot 8^i \quad , \quad b_i \in \{0,1,2,...,7\} \quad .$$

1	=	$\mathbf{8^0}$	$= 2^0$
8	=	$\mathbf{8^1}$	$= 2^3$
64	=	$\mathbf{8^2}$	$= 2^6$
512	=	$\mathbf{8^3}$	$= 2^9$
4 096	=	$\mathbf{8^4}$	$= 2^{12}$
32 768	=	$\mathbf{8^5}$	$= 2^{15}$
262 144	=	$\mathbf{8^6}$	$= 2^{18}$
2 097 152	=	$\mathbf{8^7}$	$= 2^{21}$

Das Sedezimalsystem ist das b-adische Zahlensystem mit der Basis B=16:

$$Z = \sum_{i=-\infty}^{n} b_i \cdot 16^i \quad , \quad b_i \in \{0,1,2,...,9,A,B,C,D,E,F\} \quad .$$

Anstelle der normierten Bezeichnung (DIN 44300) Sedezimalsystem wird häufig auch Hexadezimalsystem verwendet und die Zahldarstellung durch ein angefügtes H gekennzeichnet.

Umwandlung einer Oktal- in eine Dualzahl:
Ausgehend von der Oktaldarstellung der Zahl ergibt sich deren Darstellung im Dualsystem, indem jede einzelne Oktalziffer durch drei aufeinanderfolgende Dualziffern repräsentiert wird.

$(523)_8 = (101\ 010\ 011)_2$, $(7,41)_8 = (111\ ,\ 100\ 001)_2$

Entsprechend läßt sich jede Dualzahl in eine Oktalzahl umwandeln, indem jeweils drei Dualziffern ausgehend vom "," zu einer Oktalziffer - ggf. durch Einfügen von Nullen - zusammengefaßt werden.

$(10\ 111\ ,\ 01)_2 = (010\ 111\ ,\ 010)_2 = (27\ ,\ 2)_8$

Analog erfolgt die Umrechnung zwischen Sedezimal- und Dualsystem, wobei hier jeweils vier Dualstellen zu einer Sedezimalziffer korespondieren.

1 Oktalstelle	**3 Dualstellen**
1 Sedezimalstelle	**4 Dualstellen**

Für die arithmetischen Operationen im Oktal- und Sedezimalsystem gelten analoge Regeln wie für das Rechnen mit Dezimalzahlen; praktikabel sind oftmals sogenannte Additions- und Multiplikationstafeln.

Additionstafel für Oktalzahlen:

+	1	2	3	4	5	6	7
1	2	3	4	5	6	7	10
2	3	4	5	6	7	10	11
3	4	5	6	7	10	11	12
4	5	6	7	10	11	12	13
5	6	7	10	11	12	13	14
6	7	10	11	12	13	14	15
7	10	11	12	13	14	15	16

Multiplikationstafel für Oktalzahlen:

·	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7
2	2	4	6	10	12	14	16
3	3	6	11	14	17	22	25
4	4	10	14	20	24	30	34
5	5	12	17	24	31	36	43
6	6	14	22	30	36	44	52
7	7	16	25	34	43	52	61

1.4 Zahlenkonvertierung

Als Zahlenkonvertierung wird die Umwandlung einer Zahlendarstellung zwischen verschiedenen B-adischen Systemen bezeichnet.

Im weiteren wird aufgrund der hervorragenden Bedeutung hauptsächlich die Umwandlung zwischen Dual- und Dezimalsystem thematisiert.

Praktische Relevanz besitzt die Konvertierung ganzer Zahlen und endlicher B-adischer Brüche, denn Digitalrechner verwenden für die Speicherung der unterschiedlichen Zahlentypen natürlich eine fixierte Stellenzahl.

Die Identität

$$Z = \sum_{i=-m'}^{n'} b'_i \cdot B'^{i} = \sum_{i=-m}^{n} b_i \cdot B^{i} \quad ,$$

die die Darstellung der Zahl Z im B'-adischen System mit der Basis B' und im B-adischen System mit der Basis B beschreibt, ist nur für ganze Zahlen - $b'_i = 0$, $i < 0$ - stets erfüllbar.

Für echte B'-adische Brüche ist die exakte Konvertierung dann möglich, wenn in B' nur Primfaktoren auftreten, die auch in B enthalten sind - vgl. Oktal- und Sedezimalsystem. Ist diese Voraussetzung nicht gegeben, so ist für eine endliche Darstellung in B erforderlich, daß im gekürzten Bruch

$$\sum_{i=-m'}^{-1} b'_i \cdot B'^{i} = \frac{b'_{-1}\, b'_{-2} \ldots b'_{-m'}}{B'^{m'}} =: \frac{p}{q}$$

der Divisor q nur solche Primfaktoren besitzt, die auch in der Basis B auftreten.

Konvertierungsfehler

Eine endlich-stellige Zahl kann bei der Konvertierung in ein anderes B-adisches Zahlensystem eine nichtendliche Ziffernfolge liefern. Hieraus resultiert aufgrund der beschränkten Stellenzahl bei Digitalrechnern der sogenannte Konvertierungsfehler, der die näherungsweise Darstellung eines Dezimalbruchs im Dualsystem

charakterisiert. $\sum_{i=-m'}^{n'} b'_i \cdot 10^{i} \quad \sum_{i=-m}^{n} b_i \cdot 2^{i}$

$(0\ ,\ 01)_{10} = (?...?)_2$

Beispiel: " 100 Pfennige 1 DMark "

```
# include<stdio.h>
int i;
float x = 0.0;
for(i = 0; i < 100; ++i)
    x += 0.01;
x -= 1.0;
printf("100 * .01 - 1 = %g\n",x);
```

100 * .01 - 1 = -6.55651 e-07 #

Die Konvertierung eines Dualbruchs in einen Dezimalbruch liefert dagegen immer ein exaktes Ergebnis, denn 2 ist Primfaktor von 10!

Die Standardmethoden zur Zahlenkonvertierung sind:

(i) **Konvertierung mittels sukzessiver Division mit Rest**,

(ii) **Konvertierung mittels sukzessiver Multiplikation und Addition**;

beide Verfahren basieren auf dem Horner-Schema (W.G. Horner, 1774-1834, englischer Mathematiker).

Einschub: Auswertung von Polynomen

* Horner-Schema

Die numerische Auswertung des Polynoms

$$p(x) = \sum_{i=0}^{n} a_i \, x^i$$

für einen vorgegebenen Wert x_0 von x läßt sich sehr effektiv durchführen, wenn

$$p(x_0) = a_n x_0^{\,n} + a_{n-1} x_0^{\,n-1} + a_{n-2} x_0^{\,n-2} + \ldots + a_1 x_0 + a_0$$

in der Form

$$p(x_0) = (\ldots \{ [(a_n x_0 + a_{n-1}) x_0 + a_{n-2}] x_0 + a_{n-3} \} x_0 + \ldots + a_1) x_0 + a_0$$

dargestellt wird. Die Auswertung erfolgt dann von innen nach außen, indem rekursiv die n Zahlen a_{01}, a_{11}, ... ,$a_{n-1,1}$, a_{n1} nach der Vorschrift

$$a_{n1} = a_n \,,$$
$$a_{j1} = x_0 a_{j+1,1} + a_j \qquad \text{für } j = n\text{-}1(\text{-}1)0$$

berechnet werden, so daß gilt:

$$a_{01} = p(x_0) \,.$$

Das Ergebnis kann auch folgendermaßen abgeleitet werden:

$$a_{01} + (x - x_0)(a_{11} + a_{21}x + \ldots + a_{n1}x^{n-1}) =$$

$$= a_{01} - x_0a_{11} + (a_{11} - x_0a_{21})x + \ldots + (a_{n-1,1} - x_0a_{n1})x^{n-1} + a_{n1}x^n =$$

$$= a_0 + a_1x + \ldots + a_{n-1}x^{n-1} + a_nx^n = p(x) \ .$$

Für die manuelle Auswertung wird das Horner-Schema in Form einer Tabelle dargestellt:

	a_n	a_{n-1}	a_{n-2}	....	a_1	a_0
x_0		x_0a_{n1}	$x_0a_{n-1,1}$	....	x_0a_{21}	x_0a_{11}
	a_{n1}	$a_{n-1,1}$	$a_{n-2,1}$	....	a_{11}	$a_{01}=p(x_0)$

Die Division eines Polynoms durch $(x - x_0)$ ergibt sich direkt mit dem Horner-Schema für $x = x_0$: In der unteren Zeile stehen die Koeffizienten des abgespaltenen Polynoms und der Divisionsrest, d.h. $p(x_0)$ ist der bei Division durch $(x - x_0)$ verbleibende Rest:

$$\frac{p(x)}{x - x_0} = (a_{11} + a_{21}\,x + \ldots + a_{n1}\,x^{n-1}) + \frac{p(x_0)}{x - x_0} \quad .$$

Beispiel:
Auswertung des Polynoms $p(x) = x^4 + 2x^3 - 3x + 1$ für $x_0 = 1.4$:

	1	2	0	-3	1
1.4		1.4	4.76	6.664	5.1296
	1	3.4	4.76	3.664	6.1296

$p(1.4) = \underline{6.1296}$;

$(x^4 + 2.0x^3 - 3x + 1) : (x - 1.4) = \quad x^3 + 3.4x^2 + 4.76x + 3.664 +$

$x^4 - 1.4x^3 \qquad\qquad + 6.1296/(x - 1.4)$

$3.4x^3 - 3x$
$3.4x^3 - 4.76x^2$

$4.76x^2 - 3x$
$4.76x^2 - 6.664x$

$3.664x + 1$
$\underline{3.664x - 5.1296}$
6.1296

$p(x) = (x^3 + 3.4x^2 + 4.76x + 3.664)\cdot(x - 1.4) + 6.1296$ #

Die Anzahl der erforderlichen Multiplikationen und Additionen ist beim Horner-Schema gleich n - entsprechend dem Grad des Polynoms: Das Horner-Schema ist bezüglich der Berechnungskomplexität das günstigste Verfahren zur Polynomauswertung!

Konvertierung mittels sukzessiver Division mit Rest

Bei diesem Verfahren wird die Konvertierung immer im Ursprungszahlensystem durchgeführt. Die Basis B des Zielsystems muß dazu im Ursprungssystem dargestellt werden und der bei den einzelnen Divisionen auftretende Rest wird jeweils in das Zielsystem umgewandelt.
Die im polyadischen Zahlensystem mit der Basis B gegebene ganze Zahl Z

$$Z = \sum_{i=0}^{n} b_i \cdot B^i$$

besitzt die Horner-Darstellung:

$$Z = (\ldots (((b_n B + b_{n-1}) B + b_{n-2}) B + b_{n-3}) B + b_{n-4}) B + \ldots + b_1) B + b_0$$

Die Ziffer b_0 ergibt sich als Rest, wenn Z durch B dividiert wird; Abtrennen des Restes und erneute Division durch B liefert b_1 usw.; die Division wird fortgesetzt bis der Quotient verschwindet.

Sukzessive Division durch B:

Z	:	B	=	q_0	Rest	b_0 (LSB)
q_0	:	B	=	q_1	Rest	b_1
q_1	:	B	=	q_2	Rest	b_2
.				.	.	.
.				.	.	.
.				.	.	.
q_{n-1}	:	B	=	0	Rest	b_n (MSB)

Beispiel:

(i) Konvertierung der Zahl $(21)_{10}$ ins Dualsystem, durchgeführt im Dezimalsystem ($B = (2)_{10}$):

21	:	2	=	10	Rest	1	$b_0 = 1$ (LSB)
10	:	2	=	5	Rest	0	$b_1 = 0$
5	:	2	=	2	Rest	1	$b_2 = 1$
2	:	2	=	1	Rest	0	$b_3 = 0$
1	:	2	=	0	Rest	1	$b_4 = 1$ (MSB)

$(21)_{10} = (10101)_2$ $\quad [= 2^4 + 2^2 + 2^0]$

(ii) Konvertierung der Zahl $(10101001)_2$ ins Dezimalsystem, durchgeführt im Dualsystem (B = $(1010)_2$):

10101001	:	1010	=	10000	Rest	1001	$b_0 = (9)_{10}$
10000	:	1010	=	1	Rest	110	$b_1 = (6)_{10}$
1	:	1010	=	0	Rest	1	$b_2 = (1)_{10}$

$(10101001)_2 = (169)_{10}$ [128 + 32 + 8 + 1 = 169] #

Für echt gebrochene Zahlen Z

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

besitzt die Horner-Darstellung die Form:

$$Z = \frac{1}{B} \cdot (b_{-1} + \frac{1}{B} \cdot (b_{-2} + ... + \frac{1}{B} \cdot (b_{-m+1} + \frac{1}{B} \cdot b_{-m}) ...) \quad .$$

Die Konvertierung erfolgt mit sukzessiver Division durch 1/B, d.h. Multiplikation mit B; die Ziffern b_i ergeben sich als ganzzahliger Anteil (Überlauf) nach jeder Multiplikation in der Reihenfolge b_{-1}, b_{-2}, ... , b_{-m}; nach Abspalten des ganzzahligen Anteils wird mit dem Rest die Multiplikation fortgesetzt.

Beispiel:
Konvertierung der Zahl $(0,69)_{10}$ ins Dualsystem, durchgeführt im Dezimalsystem (sukzessive Division durch 1/2):

0,69	·	2	=	1,38	Überlauf 1	$b_{-1} = 1$
0,38	·	2	=	0,76	Überlauf 0	$b_{-2} = 0$
0,76	·	2	=	1,52	Überlauf 1	$b_{-3} = 1$
0,52	·	2	=	1,04	Überlauf 1	$b_{-4} = 1$
0,04	·	2	=	0,08	Überlauf 0	$b_{-5} = 0$
0,08	·	2	=	0,16	Überlauf 0	$b_{-6} = 0$
.	.	.		.	" .	. .
.	.	.		.	" .	. .
.	.	.		.	" .	. .

$(0,69)_{10}$ $(0,101100)_2$ [1/2 + 1/8 + 1/16 = 0,6875 0,69]

Konvertierung der Zahl $(0,11)_2$ ins Dezimalsystem, durchgeführt im Dualsystem (Multiplikation mit $(1010)_2$):

$0,11 \cdot 1010 = 111,10$ Überlauf 111 $b_{-1} = (7)_{10}$

$0,10 \cdot 1010 = 101,00$ Überlauf 101 $b_{-2} = (5)_{10}$

$0,00 \cdot 1010 = 0,00$ Überlauf 0 $b_{-3} = (0)_{10}$

$(0,11)_2 = (0,75)_{10}$ #

Eine gebrochene Zahl Z

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

wird konvertiert indem der ganze und der echt gebrochene Teil mit

$$Z = \sum_{i=0}^{n} b_i \cdot B^i + \sum_{i=-m}^{-1} b_i \cdot B^i$$

separat umgewandelt wird bzw. indem Z mit Hilfe einer Normierung als ganze oder echt gebrochene Zahl dargestellt wird:

$$Z = \left(\sum_{i=0}^{n+m} b_i \cdot B^i \right) \cdot \frac{1}{B^m} = \left(\sum_{i=-(m+n+1)}^{-1} b_i \cdot B^i \right) \cdot B^{n+1}$$

Selbstverständlich muß das Resultat anschließend mit $1/B^m$ bzw. B^{n+1} renormiert werden.

Beispiel:
Konvertierung der Zahl $(3,75)_{10}$ ins Dualsystem:

(i) $(3,75)_{10} = (3)_{10} + (0,75)_{10}$

3	:	2	=	1	Rest	1		0,75	·	2	=	1,5	Überlauf	1
1	:	2	=	0	Rest	1		0,50	·	2	=	1,0	Überlauf	1

$(3)_{10} = (11)_2$ $(0,75)_{10} = (0,11)_2$

$\underline{(3,75)_{10} = (11,11)_2}$

(ii) $(3,75)_{10} = (0,375)_{10} \cdot (10)_{10}$

0,375	·	2	=	0,75	Überlauf 0
0,75	·	2	=	1,50	Überlauf 1
0,50	·	2	=	1,00	Überlauf 1

$(0,375)_{10} = (0,011)_2$, Multiplikation mit $(10)_{10} = (1010)_2$,
$\underline{(3,75)_{10} = (11,11)_2}$. #

Konvertierung mittels sukzessiver Multiplikation und Addition

Diese Methode fußt auf der Horner-Darstellung der ganzen Zahl Z

$$Z = \sum_{i=0}^{n'} b'_i \cdot B'^i = (\ldots ((b'_n \cdot B' + b'_{n'-1}) \cdot B' + b'_{n'-2}) \cdot B' + \ldots$$

$$\ldots + b'_1) \cdot B' + b'_0 \ ,$$

wobei die höchstwertigste Stelle $b'_{n'}$ mit der Basis B' multipliziert und $b'_{n'-1}$ addiert wird, anschließend wird das Ergebnis wieder mit B' multipliziert etc. Die Konvertierung resultiert indem die Ziffern b'_i genau so wie die Basis B' bei der sukzessiven Multiplikation und Addition im Zielsystem mit der Basis B dargestellt werden.

Sukzessive Multiplikation und Addition:

$b'_{n'} \cdot B' = a_1$, $a_1 + b'_{n'-1} = a_2$

$a_2 \cdot B' = a_3$, $a_3 + b'_{n'-2} = a_4$

. . . , . . .

. . . , $a_{2n'-1} + b'_0 = (Z)_B$

Die Berechnung erfolgt im Zielsystem mit der Basis B.

Horner-Schema:

	$b'_{n'}$	$b'_{n'-1}$	$b'_{n'-2}$	...	b'_0
B'		a_1	a_3	...	$a_{2n'-1}$
	$a_0 = b'_{n'}$	a_2	a_4	...	$a_{2n'}=Z$

Beispiel:
Konvertierung der Zahl $(10101000)_2$ ins Dezimalsystem, durchgeführt im Dezimalsystem:

	b'_7 1	b'_6 0	b'_5 1	b'_4 0	b'_3 1	b'_2 0	b'_1 0	b'_0 0
B'= 2		2	4	10	20	42	84	168
	1	2	5	10	21	42	84	168

$(10101000)_2 = (168)_{10}$

Konvertierung der Zahl $(2314)_{10}$ ins Dualsystem, durchgeführt im Dualsystem:

Darstellung im Zielsystem: $B' = (10)_{10} = (1010)_2$

$(2)_{10} = (10)_2$, $(3)_{10} = (11)_2$, $(1)_{10} = (1)_2$, $(4)_{10} = (100)_2$

	10	11	1	100
B'=1010		10100	11100110	100100000110
	10	10111	11100111	100100001010

$(2314)_{10} = (100100001010)_2$ #

Die Konvertierung einer echt gebrochenen Zahl ergibt sich über die sukzessive Multiplikation mit der inversen Basis 1/B', wobei entsprechend der Horner-Darstellung mit der letzten Stelle begonnen und die Rechnung im Zielsystem ausgeführt wird.

Schema:

$b'_{-m'} \cdot 1/B' = a_1$, $a_1 + b'_{-(m'-1)} = a_2$

$a_2 \cdot 1/B' = a_3$, $a_3 + b'_{-(m'-2)} = a_4$

. . . , . . .

. . . , $a_{2m'-3} + b'_{-1} = a_{2(m'-1)}$

$a_{2(m'-1)} \cdot 1/B' = a_{2m'-1} = (Z)_B$

Beispiel:
Konvertierung der Zahl $(0,1011)_2$ ins Dezimalsystem, durchgeführt im Dezimalsystem:

	$1 = b'_{-4}$	1	0	1	0
1/B'= 0,5		0,5	0,75	0,375	0,6875
	1	1,5	0,75	1,375	

$(0,1011)_2 = (0,6875)_{10}$ #

Konvertierung mit Hilfe eines Zwischensystems

Die direkte Konvertierung zwischen Dual- und Dezimalsystem und vice versa läßt sich häufig leichter durchführen, wenn ein Zwischensystem - Oktal- oder Sedezimalsystem - benutzt wird.

Konvertierungsabfolge:

Dualsystem	Oktalsystem	Dezimalsystem
Dualsystem	Sedezimalsystem	Dezimalsystem

Hierbei erleichtern Tabellen die Umwandlung zwischen Oktal-/Sedezimal- und Dezimalsystem ganz erheblich.

Konvertierungstabelle (sedezimal/dezimal):

4. Stelle		3. Stelle		2. Stelle		1. Stelle	
B=16	B=10	B=16	B=10	B=16	B=10	B=16	B=10
0	0	0	0	0	0	0	0
1	4 096	1	256	1	16	1	1
2	8 192	2	512	2	32	2	2
3	12 288	3	768	3	48	3	3
4	16 384	4	1 024	4	64	4	4
5	20 480	5	1 280	5	80	5	5
6	24 576	6	1 536	6	96	6	6
7	28 672	7	1 792	7	112	7	7
8	32 768	8	2 048	8	128	8	8
9	36 864	9	2 304	9	144	9	9
A	40 960	A	2 560	A	160	A	10
B	45 056	B	2 816	B	176	B	11
C	49 152	C	3 072	C	192	C	12
D	53 248	D	3 328	D	208	D	13
E	57 344	E	3 584	E	224	E	14
F	61 440	F	3 840	F	240	F	15

Beispiel:

Konvertieren der Zahl $(10110101101)_2$ ins Dezimalsystem via Sedezimaldarstellung:

$$(10110101101)_2 = (0101\ 1010\ 1101)_2 = (5\ \ A\ \ D)_{16}$$

$$(5\ \ A\ \ D)_{16} = (1280)_{10} + (160)_{10} + (13)_{10} = (1453)_{10}$$

Konvertieren der Zahl $(37383)_{10}$ ins Dualsystem via Sedezimalsystem:

1. Umwandlung ins Sedezimalsystem (sukzessive Division mit Rest)

37383	:	16	=	2336	Rest 7
2336	:	16	=	146	Rest 0
146	:	16	=	9	Rest 2
9	:	16	=	0	Rest 9

$$(37383)_{10} = (9207)_{16}$$

2. Umwandlung ins Dualsystem (1 Sedezimalstelle ⇔ 4 Dualstellen)

$$(37383)_{10} = (9207)_{16} = (1001\ 0010\ 0000\ 0111)_2$$

#

1.5 Komplementdarstellungen

Das Rechenwerk der ALU (Arithmetisch Logische Einheit, arithmetic logic unit) in Digitalrechnern führt die arithmetischen Grundoperationen - Addition, Subtraktion, Multiplikation und Division - zumeist über eine einfache Addition aus, wobei die Subtraktion als Komplementaddition und die Multiplikation bzw. Division als wiederholte Addition bzw. Subtraktion durchgeführt werden.
Negative Zahlen werden häufig im B- bzw. (B-1)-Komplement dargestellt.

Komplementaddition:

$$a - b = a + \overline{b} - K \qquad , \quad a,b,K \in \mathbb{N} \; ,$$

$$\overline{b} := K - b \quad , \quad \textit{Komplement von } b \textit{ zu } K \; .$$

Die positive Zahl b besitze im polyadischen Zahlensystem mit der Basis B eine n-stellige Darstellung, dh. in Radixschreibweise gilt:

$$b = b_{n-1}b_{n-2}b_{n-3} \; ... \; b_0 \; .$$

<u>B-Komplement (B^n-Komplement):</u>

$$\overline{b} = B^n - b$$

Die Summe aus Zahl und B^n-Komplement ist identisch zur nächsthöheren Basispotenz:

$$B^n = b + \overline{b} \quad .$$

<u>(B-1)-Komplement ((B^n-1)-Komplement):</u>

$$\overline{b} = (B^n - 1) - b$$

Die Summe aus Zahl und (B^n-1)-Komplement liefert die größte darstellbare n-stellige Zahl.

Addition von 1 zum (B^n-1)-Komplement ergibt das B^n-Komplement:

$$\overline{b}^{(B^n)} = \overline{b}^{(B^n-1)} + 1 \quad .$$

Die Berechnung des B-Komplements einer Zahl in Radixdarstellung erfolgt, indem jede Ziffer von der um Eins verkleinerten Basis subtrahiert und anschließend eine Eins zur niedrigsten Stelle - unter Beachtung aller Überträge - addiert wird.

Die Berechnung des (B-1)-Komplements einer Zahl in Radixdarstellung erfolgt durch Subtraktion jeder einzelnen Ziffer von der größten Ziffer (B-1) der Darstellung.

Beispiel:

(i) B = 10

n	b	(B-1)-Kompl.	B-Kompl.
1	3	6	7
2	44	55	56
4	6121	3878	3879

(B-1)-Komplement Neuner-Komplement 9-Komplement,
B-Komplement Zehner-Komplement 10-Komplement;

(ii) B = 2

n	b	(B-1)-K.	B-K.	B^n-1	B^n
1	1	0	1	1	10
1	0	1	*10*	1	10
2	10	01	10	11	100
4	0011	1100	1101	1111	1 0000
8	1100110	0011001	0011010	1111 1111	1 0000 0000

(B-1)-Komplement Einer-Komplement 1-Komplement,
B-Komplement Zweier-Komplement 2-Komplement;

Bei Dualzahlen liefert das Invertieren aller Ziffern (0 1) das (B^n-1)-Komplement; das B^n-Komplement ergibt sich aus dem (B^n-1)-Komplement durch Addition einer 1!

#

Subtraktion im Dezimalsystem mittels Komplementbildung:

24 - 8 = 16 (n = 2)

9-Komplement von 08: 99 - 08 = 91

```
  24
+ 91
 115
+  1   Einerrücklauf korrespondiert zur Subtraktion von 99
  16
```

10-Komplement von 08: 100 - 08 = 92

```
  24
+ 92
 116  Übertrag streichen
  16
```

Subtraktion im Dualsystem mittels (B-1)-Komplement

$$B^n - 1 = 2^n - 1 = 111 \ldots 11$$

(n *Stellen*)

Zahlenbereich für n-stellige ganze Dualzahlen:

$b \geq 0 \qquad 0 \leq b \leq 2^{n-1} - 1$

$b \leq 0 \qquad -2^{n-1} + 1 \leq b \leq 0$

n = 4:

dezimal	-7	-6	-5	-4	-3	-2	-1	-0	+0	+1	+2	+3	+4	+5	+6	+7
dual	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111

Charakteristisch für die Darstellung ist das Auftreten zweier Nullen, hieraus resultiert eine symmetrische Aufteilung der positiven und negativen Zahlenbereiche; die höchstwertigste Stelle (MSB) läßt sich als Vorzeichenbit ("0" - positiv, "1" - negativ) interpretieren.

$$\mathbf{a - b = a + b - (B^n - 1)}$$

- Komplementbildung: $b = (B^n - 1) - b$
- Komplementaddition: $a + b$
- Subtraktion: $a + b - (B^n - 1)$

Fallunterscheidung:

(i) a > b

In der fiktiven (n+1)-Stelle tritt eine Eins auf, Vernachlässigung dieser Stelle korrespondiert zur Subtraktion von B^n, d.h. die zuviel subtrahierte Eins [$(B^n - 1)$-Komplement] muß anschließend wieder addiert werden (Einerrücklauf).

(ii) $a \leq b$

Es tritt keine Eins in der (n+1)-Stelle auf, nach der Addition des Komplements ergibt sich ein negatives Ergebnis in Komplementdarstellung und der Subtraktionsschritt (Rekomplementierung) entfällt.

Einerrücklauf
Tritt nach der Komplementaddition in der fiktiven (n+1)-Stelle eine Eins auf, so wird diese Stelle vernachlässigt und das Ergebnis durch Addition einer Eins korrigiert.

Beispiel:

13 - 7 = 6 , n = 5

- Komplementbildung: $(7)_{10}$ = 00111
  $(-7)_{10}$ = 11000

- Komplementaddition: $(13)_{10}$ = 01101
  $+(-7)_{10}$ = 11000

  100101

- Einerrücklauf: 1

  00110 = $(6)_{10}$

7 - 13 = -6 , n = 5

- Komplementbildung: $(13)_{10}$ = 01101
  $(-13)_{10}$ = 10010

- Komplementaddition: $(7)_{10}$ = 00111
  $+(-13)_{10}$ = 10010

  011001 = $(-6)_{10}$ #

Subtraktion im Dualsystem mittels B-Komplement

$$B^n = 2^n = 100 \ldots 00 \quad (n \text{ Nullen})$$

Zahlenbereich für n-stellige ganze Dualzahlen:

$b \geq 0 \qquad 0 \leq b \leq 2^{n-1} - 1$

$b < 0 \qquad -2^{n-1} \leq b \leq -1$

n = 4:

dezimal	-8	-7	-6	-5	-4	-3	-2	-1	0	+1	+2	+3	+4	+5	+6	+7
dual	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111

Charakteristisch für die Darstellung ist eine unsymmetrische Aufteilung der positiven und negativen Zahlenbereiche; die höchstwertigste Stelle (MSB) läßt sich als Vorzeichenbit ("0" - positiv, "1" - negativ) interpretieren.

$$\mathbf{a - b = a + b - B^n}$$

- Komplementbildung: $b = B^n - b$
- Komplementaddition: $a + b$
- Subtraktion: ~~$a + b - B^n$~~
  Subtraktion entfällt, denn Vernachlässigung des Überlaufs in der (n+1)-Stelle, die aus dem betrachteten Zahlenbereich hinausführt, entspricht gerade der Subtraktion von B^n!

Beispiel:

13 - 7 = 6 , n = 5

- Komplementbildung:
  $(7)_{10}$ = 00111
  $(-7)_{10}$ = 11000
  \+ 1
  11001
- Komplementaddition:
  $(13)_{10}$ = 01101
  $+(-7)_{10}$ = 11001
  100110
- Überlauf streichen: 00110 = $(6)_{10}$

7 - 13 = -6 , n = 5

- Komplementbildung:

$(13)_{10}$ = 01101
$(-13)_{10}$ = 10010
+ 1
10011

- Komplementaddition:

$(7)_{10}$ = 00111
$+(-13)_{10}$ = 10011

11010 = $(-6)_{10}$ #

Bemerkung: "Bereichsüberschreitungen"

Es ergeben sich falsche Resultate, wenn der zulässige Zahlenbereich überschritten wird, z.B.:

13 + 7 = 20 , n = 5

$(13)_{10}$ = 01101
$+(7)_{10}$ = 00111

10100 = $(-12)_{10}$ ☹

Abhilfe:
Einführung einer Schutzstelle derart, daß alle gültigen positiven/ negativen Zahlen in den gleichbewerteten Stellen n und n+1 eine 00/ 11 aufweisen.

Besitzt ein Ergebnis nun eine der Kombinationen 01 oder 10 in den führenden Stellen, so liegt eine Bereichsüberschreitung vor.

13 + 7 = 20 , n = 5

$(13)_{10}$ = 001101
$+(7)_{10}$ = 000111

010100

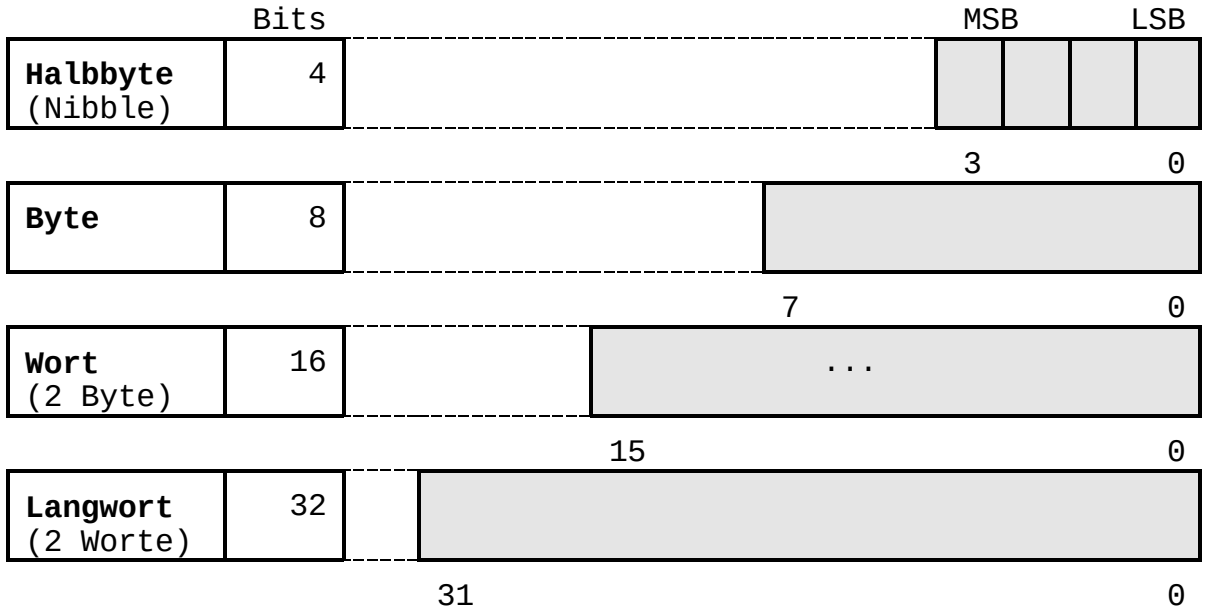
1.6 Zahlendarstellungen

Ein Wort stellt allgemein eine geordnete (als Einheit adressierbare) Sequenz von N Bits dar - die Zahl N kennzeichnet die Wortlänge.

Die binäre Darstellung von N Symbolen erfordert ld N := $\log_2$ N Bits, z.B.:

Dualziffer	ld 2 = 1 Bit ,
Oktalziffer	ld 8 = 3 Bits ,
Dezimalziffer	ld 10 = 3,32 Bits also 4 Bits.

Wortlängen in Bussystemen:



In der Regel wird eine Zahl durch ein Wort repräsentiert, d.h. der resultierende Zahlenbereich ist infolge der endlichen Wortlänge beschränkt.

Im folgenden wird die Darstellung im Dualsystem betrachtet; einige Rechnersysteme (IBM 360/370, Siemens) verwenden B=16 als Basis.

Positive ganze Zahlen

Mit der Wortlänge 16 Bit lassen sich im Dualsystem die natürlichen Zahlen des Bereichs

$$0 \; - \; (\; 2^{16} \; - 1 \;) = 65535$$

darstellen.

Die nicht benötigten Stellen höherer Wertigkeit werden durch führende Nullen dargestellt:

$$(867)_{10} = 0000\ 0011\ 0110\ 0011 \quad .$$
MSB LSB

Ganze Zahlen

Es gibt grundsätzlich drei Möglichkeiten für die Darstellung negativer Zahlen:

Darstellung nach Betrag und Vorzeichen,

Darstellung im B-Komplement,

Darstellung im (B-1)-Komplement,

wobei die feste Wortlänge bewirkt, daß der Bereich der verfügbaren positiven Zahlen eingeschränkt wird.

* Darstellung nach Betrag und Vorzeichen

Hierbei wird das MSB als Vorzeichenbit s verwendet,

s = 0 positive
kennzeichnet eine Zahl.
s = 1 negative

Die feste Wortlänge sichert die Eindeutigkeit der Darstellung:

$$(118)_{10} = 0\ 1\ 1\ 1 \;\; 0\ 1\ 1\ 0 \;\; ,$$

$$(-118)_{10} = 1\ 1\ 1\ 1 \;\; 0\ 1\ 1\ 0 \;\; .$$
$(-1)^s \; 2^6 2^5 \; 2^4 \;\; 2^3 \; 2^2 \; 2^1 \; 2^0$

Beispiel:
Wortlänge 16 Bit (= 1 Vorzeichenbit + 15 Bit für den Betrag)
Darstellbarer Zahlenbereich:

$$1111\ 1111\ 1111\ 1111 \leq Z \leq 0111\ 1111\ 1111\ 1111$$
$$\{ \; -32\ 768 \leq Z \leq 32\ 767 \; \} \quad . \qquad \#$$

* Darstellung im B-Komplement (2-Komplement)

Zurückführung der Subtraktion auf die Komplementaddition, d.h. Addition und Subtraktion lassen sich mit einem Addierwerk ausführen; das höchste Bit liefert ebenfalls das Vorzeichen der Zahl:

$$(\ 118)_{10} = \mathbf{0}\ 1\ 1\ 1\ \ 0\ 1\ 1\ 0\ \ ,$$

$$(-118)_{10} = \mathbf{1}\ 0\ 0\ 0\ \ 1\ 0\ 1\ 0\ \ \text{(2-Komplement)}\ .$$

Zahlenbereich für eine Wortlänge von 16 Bit:

$$-32\ 768 = -2^{15} \le Z \le 2^{15} - 1 = 32\ 767\ .$$

Die Umwandlung einer Zahl in 2-Komplementdarstellung der Wortlänge N in eine mit der Wortlänge N' (N' > N) geschieht durch eine Vorzeichenergänzung (Vorzeichenerweiterung), d.h. die führenden Bits werden mit dem Vorzeichenbit aufgefüllt:

$$(\ 118)_{10} = \mathbf{0}\ 1\ 1\ 1\ \ 0\ 1\ 1\ 0\ = \mathbf{0\ 0\ 0\ 0\ \ 0\ 0\ 0\ 0}\ \mathbf{0}\ 1\ 1\ 1\ \ 0\ 1\ 1\ 0,$$

$$(-118)_{10} = \mathbf{1}\ 0\ 0\ 0\ \ 1\ 0\ 1\ 0\ = \mathbf{1\ 1\ 1\ 1\ \ 1\ 1\ 1\ 1}\ \mathbf{1}\ 0\ 0\ 0\ \ 1\ 0\ 1\ 0.$$

Einschub: Offset-Dual-Darstellung

Die Transformation in die Offset-Dual-Darstellung erfolgt über ein Verschieben des Zahlenbereichs durch Addition eines "Offsets" derart, daß die größte negative Zahl nach Null verschoben wird.

Beispiel:
Die 2-Komplementdarstellung mit einer Wortlänge von 8 Bit umfaßt den Zahlenbereich von -128 bis +127; Addition von 128 liefert die Offset-Dual-Darstellung, die den Bereich von 0 bis 255 überdeckt, d.h. Zahlen größer/kleiner als 128 sind positiv/negativ und die Bereichsmitte korrespondiert zur Null.

B = 10	8Bit 2-Komplement-D.	8Bit Offset-Dual-D.
127	0 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1
.	.	.
.	.	.
.	.	.
1	0 0 0 0 0 0 0 1	1 0 0 0 0 0 0 1
0	0 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0
-1	1 1 1 1 1 1 1 1	0 1 1 1 1 1 1 1
.	.	.
.	.	.
.	.	.
-127	1 0 0 0 0 0 0 1	0 0 0 0 0 0 0 1
-128	1 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0

Beachte: Der Übergang zwischen den beiden Darstellungen ergibt sich mit Hilfe der Invertierung des MSB! #

Festkommadarstellung

Bei der Festkommadarstellung ist die Anzahl der Stellen vor bzw. nach dem Komma (Dezimal-, Dualpunkt) fixiert.
Für eine gegebene Wortlänge ergibt die Division mit der wertniedrigsten Zweierpotenz ganze Zahlen; nach Ausführung der arithmetischen Operationen liefert eine entsprechende Multiplikation das Ergebnis.

Gleitkomma- (Gleitpunkt-) Darstellung

Eine wesentliche Erweiterung des Zahlenbereichs insbesondere mit Blick auf technisch-wissenschaftliche Anwendungen liefert die halbexponentielle (halblogarithmische) Darstellung in der Form:

$$Z = M \cdot B^E ,$$

M - Mantisse, E - Exponent, B - Basis des Zahlensystems.

Gleitkomma - Dezimalzahl: $Z = M \cdot 10^E$

$522{,}345 = 5{,}22345 \cdot 10^2 = 522345 \cdot 10^{-3} = 522345\ E\text{-}3$

Gleitkomma - Dualzahl: $Z = M \cdot 2^E$

1110 0001,1101 = 1,1100 0011 101 E 0111

Floating - Point - Standard IEEE - P754
(IEEE - Institution of Electrical and Electronic Engineers)

Als Basis wird hier B = 2 verwendet und die Eindeutigkeit der Darstellung wird dadurch sichergestellt, daß die Mantisse

$$M = m_0 , m_1 m_2 m_3 m_4 ...$$

durch Modifikation des Exponenten auf $m_0 = 1$ normiert wird (**normalisierte Gleitpunktdarstellung**), d.h. explizit besitzt M die Darstellung

$$M = 1 + m_1 2^{-1} + m_2 2^{-2} + ... = 1 + \sum_{i=1}^{k} m_i 2^{-i}$$

mit $1 \leq M < 2$.

Um beim Exponenten E die Angabe eines Vorzeichens einzusparen, wird E im IEEE-Format als Offset-Dualzahl aufgeführt, indem abhängig von der Wortlänge eine Konstante e,

$e = 2^7 - 1 = 127$ (32 Bits, einfache Länge),
$e = 2^{10} - 1 = 1023$ (64 Bits, doppelte Länge),
$e = 2^{14} - 1 = 16383$ (80 Bits, erweiterte doppelte Länge)

zum Exponenten E addiert wird; der resultierende vorzeichenfreie Exponent wird als Charakteristik bezeichnet:

Charakteristik = Exponent + Konstante

C = E + e .

$$Z = M \cdot 2^E$$

IEEE Format	Wort-länge	Vor-zeichen s	Exponent Länge E	Exponent Bereich		Mantisse Länge M	Mantisse Dezimalen
Einfach	32bit	1bit	8bit	$2^{\pm 127}$	$10^{\pm 38}$	23bit	7
Doppelt	64bit	1bit	11bit	$2^{\pm 1023}$	$10^{\pm 308}$	52bit	16
Intern	80bit	1bit	15bit	$2^{\pm 16383}$	$10^{\pm 4932}$	64bit	19

32-bit-Single-Precision-Format:

s	$C_7C_6C_5C_4C_3C_2C_1C_0$	$m_1\ m_2\ m_3\ m_4\ .....\ m_{22}\ m_{23}$

64-bit-Double-Precision-Format:

s	$C_{10}C_9\ ...\ C_2C_1C_0$	$m_1\ m_2\ m_3\ m_4\ .....\ m_{51}\ m_{52}$

Für die interne Verarbeitung wird die 80-bit-Darstellung empfohlen.

Das Bit s ist das Vorzeichen der Zahl und der Mantisse, die im 2-Komplement mit Vorzeichen s und Betrag gespeichert wird.

Beispiel: 32bit Gleitkommadarstellung

(normierte Darstellung, d.h. m_0 wird nicht aufgeführt; das Komma steht vor m_1)

	s	**E+e**	**M**
- 1,375	1	01111111 ,	$\overline{0110...0}$
	{ -	127	0,375 }
+ 10	0	10000010 ,	0100...0
	{ +	130	0,25 }

#

Spezialfälle und Ausnahmen am Beispiel des 32-Bit-Formats:

Die Charakteristik C = E + e umfasst 8 Bits, d.h.

$$0 \leq C \leq 255 .$$

* C=0, M=0: Darstellung der Null

s 0000 0000 , 000 ... 0 ,

* C=0, M 0: Darstellung einer nicht normalisierten Gleitkomma-Zahl (Underflow) ,

* C=255, M=0: Overflow ($\pm \infty$)

0 1111 1111 , 000 ... 0 $+ \infty$,

* C=255, M 0: keine gültige Gleitkomma-Zahl (NaN: Not a Number)

Zahlendarstellung \ { 0, $\pm \infty$ }:

$1 \leq C \leq 254$ $-126 \leq E \leq 127$ (E = C - e)

- größte positive Zahl

0 1111 1110 , 111 ... 1 $2^{127} \cdot (1 + (1 - 2^{-23}))$,

\+ 254 (1 - 2^{-23}),

$2^{127} \cdot (2 - 2^{-23}) = 3,4 \cdot 10^{38}$.

- Differenz zwischen der größten und zweitgrößten Zahl

$2^{127} \cdot (2 - 2^{-23}) - 2^{127} \cdot (2 - 2^{-22}) = 2^{105} - 2^{104} \approx 2 \cdot 10^{31}$.

- kleinste positive Zahl

0 0000 0001 , 000 ... 0 $\hat{=} 2^{-126} \cdot (1 + 0)$,

$2^{-126} \approx 1{,}2 \cdot 10^{-38} \cdot$

- Differenz zwischen der kleinsten und zweitkleinsten Zahl

$2^{-126} \cdot (1 + 2^{-23}) - 2^{-126} \approx 1{,}4 \cdot 10^{-45}$.

Der Exponent ermöglicht es unterschiedliche Wertebereiche auszuwählen, wobei innerhalb eines Wertebereichs (Intervall) ungefähr $1{,}6 \cdot 10^{7}$ Einzelwerte aufgrund der Mantissenlänge von 24 Bits gegeben sind - für große Exponenten ist die Auflösung am geringsten!

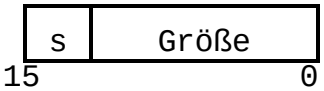
Die Genauigkeit der Zahlendarstellung ergibt sich über die Länge der Mantisse; bei einer Mantissenlänge von n Bits können

$$n/(\mathrm{ld}\ 10) = 0{,}3 \cdot n$$

Dezimalstellen dargestellt werden.

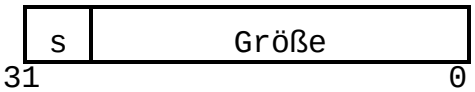
Zahlendarstellungen bei Intel 80X86 Mikroprozessoren:
(Borland C++, Programmierhandbuch)

int



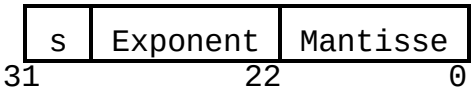
(2-Komplement)

long int

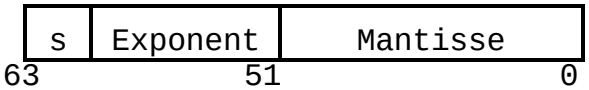


(2-Komplement)

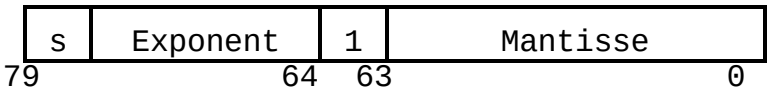
float 1/



double 1/



long double /



s : Vorzeichenbit (0=positiv, 1=negativ)
/ : Position des implizierten Dualpunkts
1 : Integerbit in Mantisse (wird nur für long double benötigt; bei den Datentypen float und double ist dieses Bit immer gesetzt!)

Exponent (normalisierte Werte):	float:	127 (7FH)
	double:	1023 (3FFH)
	long double:	16383 (3FFFH)

1.7 Numerische Aspekte

Reelle Zahlen werden durch eine endliche Gleitkommadarstellung im allgemeinen nicht exakt, sondern in gerundeter Form als Maschinenzahlen gespeichert; dies gilt insbesondere für alle irrationalen Zahlen.

Grundsätzlich legt die Wahl der internen Gleitkommadarstellung den Rechenbereich, den Speicherbedarf, die Maschinengenauigkeit und die Verarbeitungsgeschwindigkeit fest. Programmiersprachen liefern zumeist die Option, den Speicherumfang und damit die Genauigkeit der numerischen Berechnungen durch unterschiedliche Darstellungen - z.B. Float, double, long double - zu beeinflussen.

In jedem Fall bilden die Maschinenzahlen nur eine endliche Teilmenge der reellen Zahlen, d.h. bildlich gesprochen stellen die Maschinen-Zahlen eine Menge diskreter und unterschiedlich dicht liegender Punkte auf der reellen Zahlengeraden dar.

Als Folge der endlichen Zahlendarstellung resultiert beispielsweise eine Verletzung des Assoziativgesetzes

$$x + (y + z) = (x + y) + z$$

der Addition bei der Gleitkommaarithmetik, wobei die Addition zweier Maschinenzahlen jeweils durch folgende Teilschritte gekennzeichnet wird:

* Vergleich der beiden Exponenten,
* Angleichen der Exponenten,
* Anpassen der Mantisse,
* Addition der Mantissen,
* Normierung des Ergebnisses.

Beispiel:

$$0.740 \cdot 10^5 + 0.806 \cdot 10^3 = ?$$

Mantisse: 3 Dezimalstellen; führende Null vor dem Trennzeichen,

Vergleich der Exponenten: 5 - 3 = 2,

Angleichen der Exponenten und Anpassung der Mantisse der kleineren Zahl:

$0.740 \cdot 10^5 + 0.008 \cdot 10^5 = ?$ (-> Informationsverlust),

Addition: $0.748 \cdot 10^5$, Ergebnis ist bereits normiert!

#

Die im Beispiel erläuterte Auslöschung signifikanter Stellen spielt insbesondere bei der näherungsweisen Berechnung unendlicher Summen eine wesentliche Rolle. Betrachten wir die folgende Näherung:

$$\frac{B^2}{6} = \sum_{i=1}^{\infty} \frac{1}{i^2} \qquad \sum_{i=1}^{N} \frac{1}{i^2} \quad ,$$

so ergibt sich für hinreichend großes N folgende Situation: mit wachsendem i werden die Zwischensummen immer größer und die hinzukommenden Summanden immer kleiner; Auslöschung führt dann dazu, dass sich die Summe für hinreichend große Werte von i überhaupt nicht mehr ändert und das Ergebnis der Näherung unbrauchbar ist. Als wirkungsvoller Ausweg bietet sich hier an, die Summe absteigend - d.h. mit i = N, N-1, N-2, . . . , 1 - zu berechnen und zuerst die kleineren Summanden zusammen zu fassen. Das Ergebnis ist in jedem Fall genauer, als wenn eine relativ große und eine relativ kleine Zahl addiert werden.

Die Basis für die Beurteilung numerischer Berechnungen bildet die Maschinengenauigkeit. Als Maschinengenauigkeit , bezeichnet man die kleinste Zahl, die addiert zu 1.0 ein von 1.0 abweichendes Resultat liefert:

$$1 + \varepsilon > 1 \ .$$

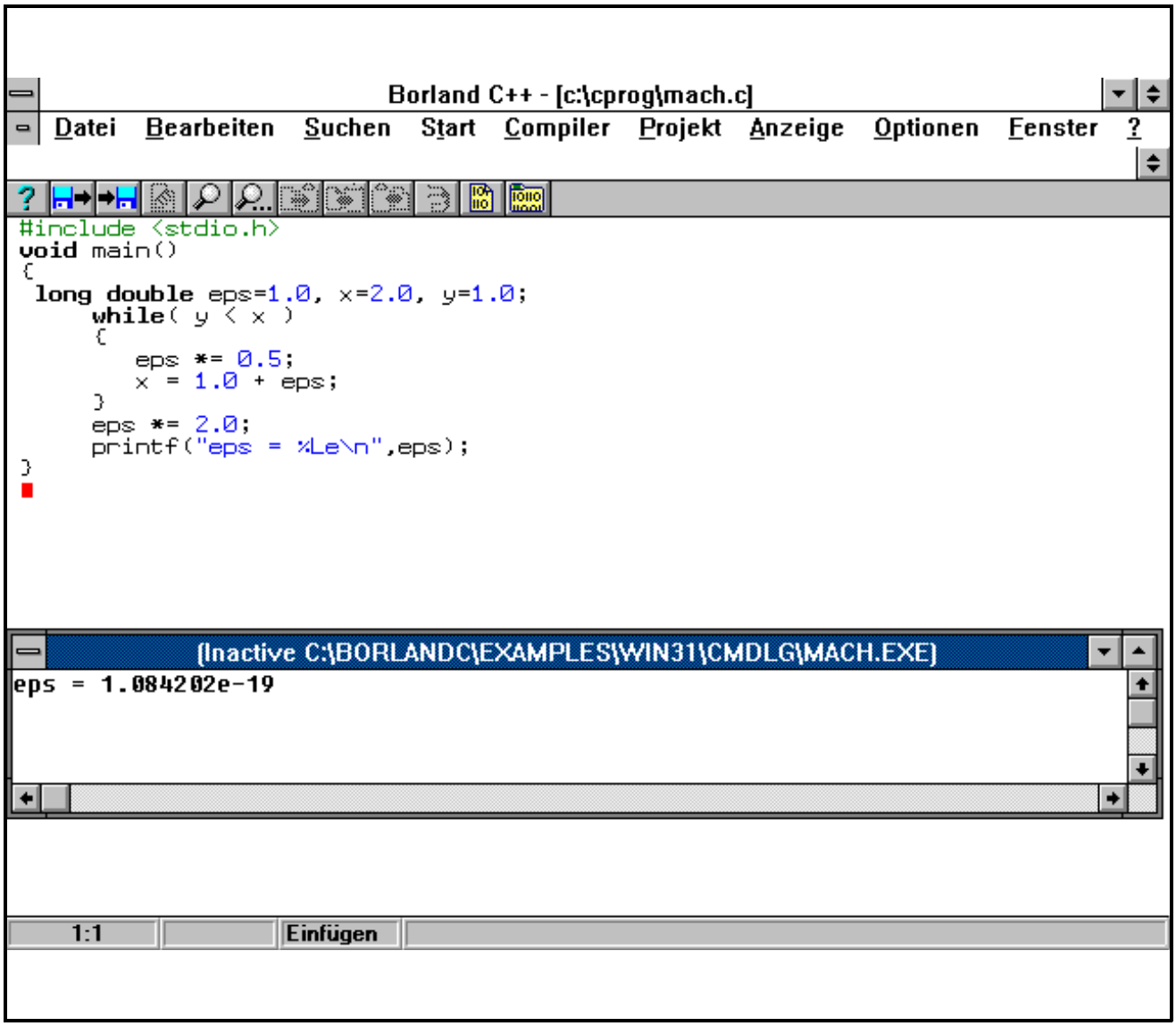
, ist nicht die kleinste Zahl, die auf der Maschine dargestellt werden kann, sondern charakterisiert die Mantisse:

ε, 10^{-7} für B = 2 und 32-bit Wortlänge,
ε, 10^{-16} für B = 2 und 64-bit Wortlänge,
ε, 10^{-19} für B = 2 und 80-bit Wortlänge.

, ist ein Maß für den relativen Fehler, der bei allen arithmetischen Operationen zwischen Gleitpunktgrößen auftritt.

Aufgrund dieser Tatsache macht es keinen Sinn Gleitpunktzahlen auf exakt Null abzufragen, sondern bei der Nullstellenbestimmung einer Funktion mit der Regula Falsi oder bei der Berechnung des Schnittpunktes zweier Geraden oder ähnlichen Berechnungen verwendet man eine darstellbare Zahl, die etwas größer als die Maschinengenauigkeit , ist, als Abbruchkriterium.

Die Maschinengenauigkeit , lässt sich leicht anhand eines kleinen Programmes ermitteln; dies erläutert die nachfolgende Abbildung.



Numerische Verfahren werden generell durch Rundungsfehler und Verfahrensfehler beeinflusst; das anschließende Beispiel der Approximation der Ableitung einer Funktion durch den zentralen Differenzenquotienten:

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h} \quad ,$$

zeigt die Wechselwirkung der beiden Fehlerkomponenten exemplarisch für die numerische Berechnung der ersten Ableitung der e-Funktion.

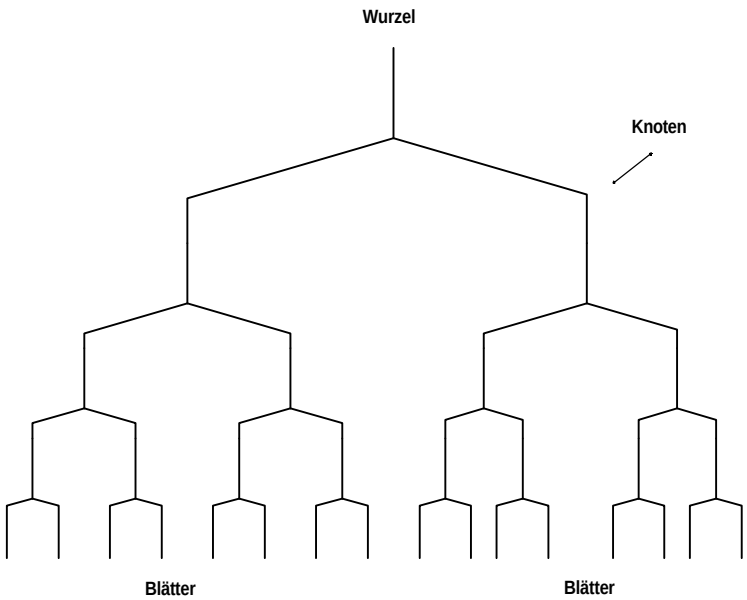
Beispiel: $f(x) = e^x,\ f'(0) = e^0 = 1$

```
#include <stdio.h>
#include <math.h>
double f(double x)
{
  return (exp (x));
}
void main()
{
double h=1, x=0;
int i;
FILE *fp;
fp=fopen("OUT.DAT","w");
for (i=0; i <=20; i++)
   { h = h/10;
     if (h != 0)
     fprintf(fp,"h = %10.2e    f'(0) = %18.15lf\n",h,
                          (f(x+h) - f(x-h))/2/h);
   };
fclose(fp);
}
```

```
h =   1.00e-01    f'(0) =  1.001667500198440
h =   1.00e-02    f'(0) =  1.000016666750000
h =   1.00e-03    f'(0) =  1.000000166666675
h =   1.00e-04    f'(0) =  1.000000001666667
h =   1.00e-05    f'(0) =  1.000000000016669
h =   1.00e-06    f'(0) =  1.000000000000133
h =   1.00e-07    f'(0) =  0.999999999999753
h =   1.00e-08    f'(0) =  1.000000000002192
h =   1.00e-09    f'(0) =  0.999999999961535
h =   1.00e-10    f'(0) =  1.000000000341006
h =   1.00e-11    f'(0) =  0.999999996004197
h =   1.00e-12    f'(0) =  0.999999996004197
h =   1.00e-13    f'(0) =  0.999999779163763
h =   1.00e-14    f'(0) =  1.000003031770280
h =   1.00e-15    f'(0) =  0.999959663683381
h =   1.00e-16    f'(0) =  0.999634403031635
h =   1.00e-17    f'(0) =  0.997465998686664
h =   1.00e-18    f'(0) =  0.975781955236954
h =   1.00e-19    f'(0) =  1.084202172485504
h =   1.00e-20    f'(0) =  0.000000000000000
h =   1.00e-21    f'(0) =  0.000000000000000
```

#

2 Codierung



2 Codierung

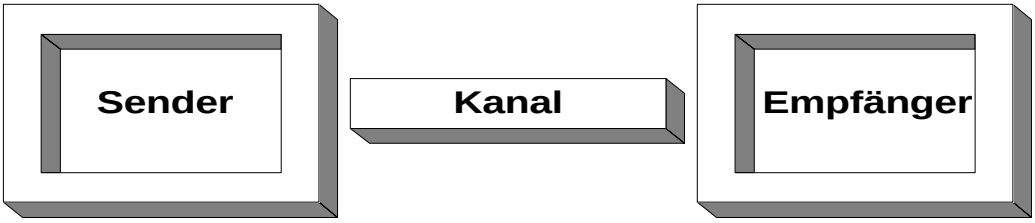
2.1 Einführung

Informatik ist die Wissenschaft von der systematischen Darstellung, Verarbeitung, Speicherung und Übertragung von Information.

Was ist Information?

Die Frage " Was ist Information? " läßt sich heute nicht definitiv und abschließend beantworten. Der Begriff *Information* wurde jedoch von Claude E. Shannon - dem Begründer der Informationstheorie - 1948 in einer grundlegenden Arbeit richtungsweisend präzisiert, insbesondere ist hier die qualitative Beschreibung über die Größen Informationsgehalt und Entropie (vgl. Thermodynamik) hervorzuheben. Die Informationstheorie stellt das mathematische Fundament für Fragestellungen im Zusammenhang mit der Speicherung, Übertragung, Codierung und Kompression von Information dar.

Die Übertragung von Information wird hierbei mit Hilfe eines Nachrichtenkanals, der zwischen Sender und Empfänger vermittelt, modelliert.



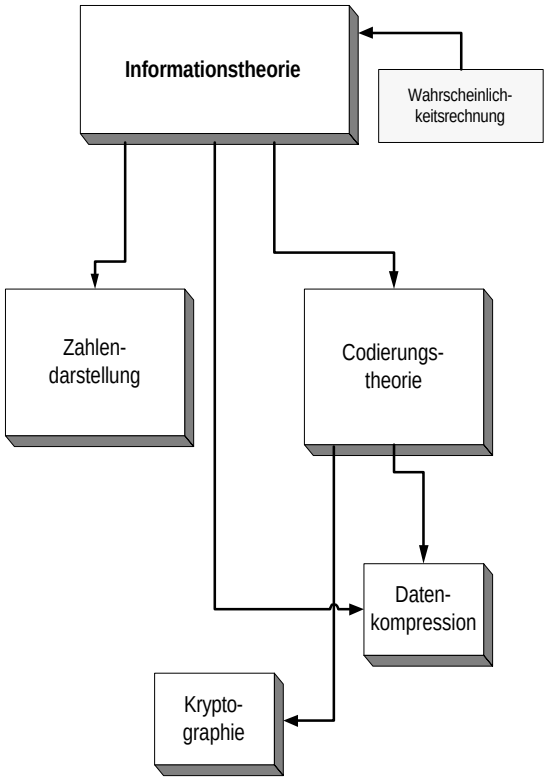
Exemplarisch läßt sich mit diesem Modell das Gespräch zweier Personen, der Satellitenempfang, die Bilddatenkompression ins JPEG-Format sowie eine Internetverbindung beschreiben; der Nachrichtenkanal kann natürlich auch gestört sein, so dass der Empfänger fehlerbehaftete Daten registriert.

Im Mittelpunkt der nachfolgenden Ausführungen steht die Codierung von Information. Beispielsweise kann die Information *Köln* durch das Wort Köln, den Buchstaben K (Autokennzeichen) oder auch CGN (Flughafen) codiert werden - Kölsch und FC sind vielleicht auch gültige Codierungen.

Die Codierung irgendwelcher Zeichen und die dafür verwendeten Codes bilden eine der wesentlichen Säulen der Informatik. Mit Hilfe unterschiedlichster Codes werden digitale Zeichen (Ziffern, Zahlen, Buchstaben, Sonderzeichen, etc.) technisch durch physikalische Zustandsgrößen -z.B. Spannungen, Ströme, Frequenzen, Schalterstellungen, Magnetisierungszustände, Lichtsignale, etc.- dargestellt.

Vorwiegend verwendet man in technischen Systemen hierfür binäre Zustände, d.h. Zustände, die durch zwei Werte eindeutig festgelegt werden. Bei Vorliegen einer zweielementigen Bildmenge spricht man allgemein von einer Binärcodierung. Eine Übersicht sowie Einordnung

der Codierungstheorie liefert die folgende Abbildung.



2.2 Grundlegende Begriffe und Definitionen

Im folgenden verstehen wir unter Codierung die Zuordnung binärer Sequenzen zu Elementen eines endlichen Alphabets.

Die Menge der binären Sequenzen wird als *Code* und die Elemente dieser Menge werden als *Codeworte* bezeichnet.

Die Menge der zu codierenden Zeichen heißt *Alphabet* und ihre Elemente heißen *Symbole* oder *Buchstaben*. Beispiele für Alphabete sind:

- die Ziffern des Dezimalsystems

  $G = \{0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9\}$,

- die Menge der kleinen lateinischen Buchstaben

  $G = \{a , b , c , ... , x , y , z\}$,

- der Zeichenvorrat der Programmiersprache C.

Die endlich langen Zeichenfolgen, die über einem Alphabet E gebildet werden können, bezeichnet man als *Wörter* über E. Wörter werden aufgebaut, indem Symbole oder bereits erzeugte Wörter aneinandergereiht, d.h. miteinander verkettet (*konkateniert*) werden; Konkatenation bedeutet Verkettung.

Definitionsgemäß gilt für die Menge E^* aller Wörter, die über einem Alphabet E gebildet werden können:

- jeder Buchstabe des Alphabets E ist auch Wort über E
  $a \in E \Rightarrow a \in E^*$,

- die Verkettung vorhandener Wörter liefert neue Wörter
  $v,w \in E^* \Rightarrow vw \in E^*$,

- das *leere Wort* g ist ein Wort über jedem Alphabet E
  $g \in E^*$ und $gw = wg = w$ für alle $w \in E^*$.

Im weiteren bezeichnet E^+ die Menge aller Wörter über E ohne das leere Wort:

$$E^+ = E^* - \{g\} .$$

Normalerweise ist das betrachtete Alphabet nicht leer, so dass E^* abzählbar unendlich viele Wörter als Elemente besitzt; falls $E = \varnothing$ resultiert $E^* = \{g\}$.

Beispiel:

Sei $E = \{a,b\}$, dann folgt: $E^* = \{g,a,b,aa,ab,ba,bb,aaa,aab,aba,...\}$
#

Für u,v,w $\in E^*$ liefert die Konkatenation ein Wort z = uvw $\in E^*$, wobei dann u als *Präfix*, v als *Infix* und w als *Postfix* bezeichnet wird.

Die Anzahl der Buchstaben eines Wortes legt die *Länge* des Wortes fest; die Länge eines Wortes läßt sich rekursiv anhand der Funktion

$$l: E^* \rightarrow \mathbb{N}$$

mit

$$l(g) = 0$$

und

$$l(wa) = l(w) + 1 \text{ für } w \in E^* , a \in E$$

berechnen. Die Länge des Wortes aba über E = {a,b} ist 3; die rekursive Berechnung stellt sich wie folgt dar:

$$l(aba)=l(ab) + 1=l(a) + 1 + 1=l(g) + 1 + 1 + 1 = 0 + 1 + 1 + 1 = 3.$$

Häufig wird die Länge l(w) eines Wortes w auch durch die Bezeichnung |w| dargestellt; diese Notation erfolgt in Anlehnung an die Bezeichnung für die Mächtigkeit (Kardinalität) einer Menge: Die Mächtigkeit |M| einer Menge M ist die Anzahl der Elemente, die zu M gehören.

Beispiel:

$$M = \{1,3,5,7,9\} , \quad |M| = 5;$$

$$A = \{a_1,a_2,a_3, \ldots ,a_{n+1}\} , \quad |A| = n+1 .$$

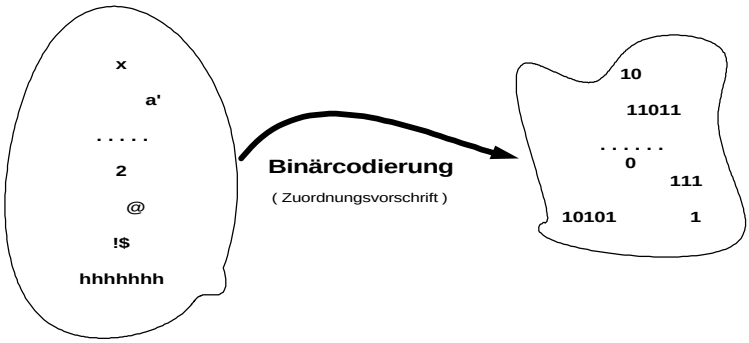
#

Def:

Eine binäre Codierung ist eine eindeutige Abbildungsvorschrift

$$c: A \rightarrow E^* ,$$

die jedem Zeichen eines Alphabets A (Urbildmenge) eine Bitkombination aus der Bildmenge $E^* = \{0,1\}^*$ zuordnet.



Beispiel:

Der ASCII-Code für den Buchstaben a ist 1000011, c wird mit 1100011 codiert und der *Buchstabe* , wird über die Bitsequenz 0011010 dargestellt, d.h. die Codeworte des ASCII-Codes besitzen alle die gleiche Länge, so dass hier eine *Codierung mit fester Länge* vorliegt. Im Gegensatz hierzu bieten *Codierungen mit variabler Länge* die Möglichkeit, den Aufwand zu verringern und eine effiziente Datenkomprimierung durchzuführen - s.u. .

#

Wesentlich ist der Aspekt der eindeutigen Decodierbarkeit eines Codes. Ein Code ist umkehrbar eindeutig, wenn zwei verschiedenen Zeichenfolgen der Urbildmenge stets zwei verschiedene Bitsequenzen der Bildmenge zugeordnet werden!
Sei

$$\mathbf{f: A \rightarrow B}$$

die Codierungsvorschrift mit der Urbildmenge A und Bildmenge B, so bezeichnet man jedes $\mathbf{b \in f(A)}$ **B** als *Codewort des Codes f*; f ist genau dann umkehrbar eindeutig, wenn für die beiden verschiedenen Zeichensequenzen $a_1a_2...a_n$ und $b_1b_2...b_m$ ($a_i \in A$, $b_j \in A$, i=1(1)n, j=1(1)m) die codierten Zeichenfolgen $f(a_1)f(a_2)...f(a_n)$ und $f(b_1)f(b_2)...f(b_m)$ ebenfalls verschieden sind.

Fano- oder Präfixbedingung:

Ein Code ist eindeutig umkehrbar -decodierbar-, falls keines der Codewörter den Anfang eines anderen Wortes des Codes bildet (hinreichende Bedingung), d.h. kein Wort w des Codes ist Anfang eines anderen Wortes w´:

$$\mathbf{w'' \ \ mit \ \ w' = w\ w''} \ ,$$

(steht für ‘*es existiert kein*’).

Jeder Code der diese Eigenschaft besitzt, heißt **Präfixcode**.

Beispiel:

Binärcodierung von 4 Zeichen,

C = { 0 , 10 , 110 , 111 }

ist eindeutig decodierbar,

C = { 0 , 10 , 100 , 111 }

dagegen nicht!

#

Besitzen alle Codewörter die gleiche Länge, d.h. jedes einzelne Codewort wird durch eine gleiche, feste Anzahl von Bits repräsentiert, so liegt ein *Code mit fester Wortlänge* vor. In der Praxis werden selbstverständlich auch *Codes mit variabler Wortlänge* gewinnbringend eingesetzt, da beispielsweise häufig auftretende Zeichen bzw. Zeichenkombinationen auf kurze Bildsequenzen abgebildet werden können und dies zu einer Datenkomprimierung ausgenutzt werden kann - vgl. Huffman-Code.

Grobklassifizierung binärer Codes:

Binäre Codes

Alphanumerische Codes

Buchstaben, Ziffern und Sonderzeichen werden binär verschlüsselt;
z.B. ASCII-Code,
EBCDI-Code, etc..

Numerische Codes

Verschlüsselung von Zahlen und Ziffern;
z.B. Dualcode,
BCD-Code,
etc..

[**ASCII**: American Standard Code for Information Interchange,
EBCDI: Extended Binary Coded Decimal Interchange,
BCD: Binary Coded Decimal (Binär codierte Dezimalziffer)]

Numerische Codes

Wortcodes

z.B.
- *Dualcode*:
Konvertierung der Dezimalzahl als Ganzes in die Dualdarstellung,

$(23)_{10}$ $(10111)_2$;

- *Gray-Code*:
.
.

Zifferncodes

z.B.
- *BCD-Code (8421)*:
Konvertierung der einzelnen Ziffern der Dezimalzahl in die Dualdarstellung,

$(23)_{10}$ $(0010\ 0011)_{BCD}$;

- *3-Exzeß-Code*:
.
.

Die kleinste Informationseinheit (binäre Einheit) ist 1 bit (binary digit); diese erlaubt die Darstellung zweier Zustände (0,1 / hell, dunkel / ...).

Im folgenden wird die Abbildung (Zuordnung) einer Nachrichtenmenge M auf die binäre Nachrichtenmenge N betrachtet:
Für eine fixierte Stellenzahl k [bit] ist in diesem Fall die Nachrichtenmenge

$$\mathbf{|M| = 2^k}$$

binär darstellbar. Umgekehrt bedeutet dies für ein gegebenes M (keine Prüfbits!)

$$\mathbf{k = ld\ |M|} \quad (\text{ ld - } \textit{logarithmus dualis}\ );$$

ist |M| keine ganze Potenz der Zahl 2, so werden mindestens

$$\mathbf{k = [ld\ |M|]}$$

Binärstellen zur Darstellung der Menge M benötigt, wobei [.] die *Gauss-Klammer* bezeichnet,

$$[r] := \min\{n \in \mathbb{N} : n \quad r\}\ ,\ r \in \quad ,$$

und jeweils die nächstgrößere natürliche Zahl liefert - technisch ist nur $k \in \mathbb{N}$ möglich ([1,20] = 2, [10,004] = 11). In der Informationstheorie (Shannon) bezeichnet man k als Informationsgehalt.

Beispiel:

M = {0,1,2,...,9}, denäres Alphabet (Alphabet der 10 Dezimalziffern),
$k = [ld\ 10]bit = [3,3]bit = 4bit$,

d.h. zur Darstellung der 10 Ziffern werden 4 Binärstellen benötigt bzw. der Informationsgehalt einer Dezimalziffer entspricht 3,3bit.

M = {a,b,c,...,z}, lateinisches alphaisches Alphabet mit 26 Buchstaben,
$k = [ld\ 26]bit = [4,7]bit = 5bit$,

d.h. die Darstellung erfordert 5bit.

#

Für $2^k > |M|$ ist von den 2^k möglichen Bitkombinationen die Anzahl

$$M_R = 2^k - |M|$$

überflüssig, d.h. M_R wird für die Codierung nicht benutzt.

Def.:
Als Maß für die nicht verwendeten Bitkombinationen eines Codes dient die *Redundanz* R:

$$R := k - ld\,|M|\ ,\ [R] = bit\ .$$

Beispiel:
Denäres Alphabet: $R = (ld\ 2^4 - ld\ 10)bit = 4bit - 3{,}3bit = 0{,}7bit$;

Alphaisches Alphabet: $R = 5bit - 4{,}7bit = 0{,}3bit$. #

Def.:
Das *Gewicht* g eines binären Codewortes der Länge k ist gleich der Anzahl der mit "1" belegten Stellen; g besitzt den Wertebereich
$0 \quad g \quad k$.

Speziell ergibt sich für das
0-Wort g = 0.
1-Wort g = k.

(0-Wort und 1-Wort sollten nach Möglichkeit nicht als gültige Codeworte verwendet werden, da beispielsweise bei einem Stromausfall die Fehlererkennung erschwert wird.)

Beispiel: *"Wortlänge und Gewicht"*

Wort:	0100	01011	000000	1111
k:	4	5	6	4
g:	1	3	0	4

#

Def.:
Ein Code heißt *gleichmäßig*, wenn alle Codeworte die gleiche Länge k besitzen; er heißt *vollständig*, wenn für die Redundanz R = 0 gilt.

Zwei nicht identische Worte eines gleichmäßigen Codes variieren in mindestens einer Binärstelle voneinander.

Def.:
Die *Hamming-Distanz* D ist identisch mit der Anzahl der (in zwei Worten eines gleichmäßigen Codes) unterschiedlichen Binärstellen.

Bemerkung: Die Hamming-Distanz D korrespondiert zur Abstandsmessung auf der Menge der Codeworte.

Beispiel:

Wort 1:	001	01101	000000	10101
Wort 2:	011	10001	111111	10101
D:	1	3	6	0

#

Eigenschaften der Hamming-Distanz D:

(i) Die Hamming-Distanz eines Wortes W_a von sich selbst ist gleich Null:

$$\mathbf{D(W_a,W_a) = 0} \quad ;$$

(ii) Die Hamming-Distanz eines Wortes W_a vom 0-Wort W_0 ist gleich dem Gewicht g_a von W_a:

$$\mathbf{D(W_a,W_0) = g_a} \quad ;$$

(iii) Die Hamming-Distanz eines Wortes W_a vom 1-Wort W_1 ist gleich der Differenz aus Wortlänge k und Gewicht g_a:

$$\mathbf{D(W_a,W_1) = k - g_a} \ ;$$

(iv) Die Hamming-Distanz eines Wortes W_a zum Wort W_b ist gleich der Hamming-Distanz des Wortes W_b zum Wort W_a (Symmetrie):

$$\mathbf{D(W_a,W_b) = D(W_b,W_a)} \ .$$

Def.:
Die *Minimaldistanz* d_{min} eines Codes ist die kleinste Hamming-Distanz aller Codewörter des Codes;
die *Maximaldistanz* d_{max} eines Codes ist die größte Hamming-Distanz bzgl. aller Codewörter des Codes.

Insbesondere ist $d_{min} = 1$ bei vollständigen / $d_{min} \geq 1$ bei redundanten Codes!

Beispiel: "*Hamming-Distanz und Fehlererkennung*"

Gegeben sei ein Alphabet mit den vier Zeichen {a,e,i,o} und folgender Codierung:

a → 00
e → 01
i → 10
o → 11 ,

d.h. bei Störung eines einzigen Bits erfolgt eine fehlerhafte Decodierung der übermittelten Information; beispielsweise liefert eine fehlerhafte Übertragung des ersten Bits des Codewortes für a als Ergebnis ein i, usw..
Wird dagegen eine redundante Codierung mit

a → 000
e → 011
i → 101
o → 110 ,

verwendet, resultiert die Möglichkeit einer *Fehlererkennung*: Störung des ersten Bits für a liefert z.B. 100 für den Empfänger, der dann -

da es kein Codewort 100 gibt - direkt die fehlerhafte Übertragung registriert; eine *Fehlerkorrektur* ist allerdings in diesem Fall nicht möglich.

Notiert man matrixförmig die Hamming-Distanzen zwischen den einzelnen Codewörten,i.e.

2-Bit-Codierung

	a	**e**	**i**	**o**
a	0	1	1	2
e	1	0	2	1
i	1	2	0	1
o	2	1	1	0

3-Bit-Codierung

	a	**e**	**i**	**o**
a	0	2	2	2
e		0	2	2
i			0	2
o				0

, wobei die Elemente ober- und unterhalb der Hauptdiagonalen aufgrund der Symmetrie natürlich identisch sind. Ein Vergleich der Minimaldistanzen beider Codes eröffnet die folgende Aussage: Bei einer Codierung mit d_{min} 2 lassen sich Übertragungsfehler, die d_{min} - 1 Bits umfassen, registrieren; werden jedoch d_{min} Bits gestört, dann ist eine Fehlererkennung nicht mehr möglich.

#

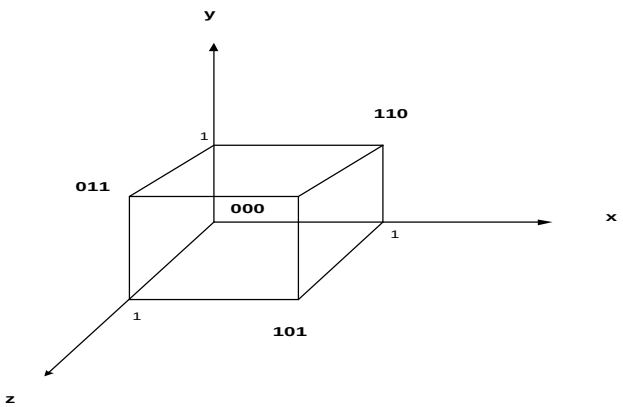
Def.:
Ein Code heißt *stetig*, wenn die Hamming-Distanz zwischen allen benachbarten Codeworten konstant ist.

Beispiel: *"Stetiger Code"*

k=3bit, M = {000 , 011 , 101 , 110} ⇒ ld |M| = (ld 4)bit =2bit ,

Redundanz: R = k - ld |M| = 1bit ,
Hamming-Distanz: D = 2 = konst. , $d_{min} = d_{max} = 2$,

Graphische Darstellung der Menge M: D korrespondiert zur Anzahl der Würfelkanten zwischen zwei Codeworten.



2.3 Numerische Codes

Im folgenden werden - zwangsläufig unvollständig - einige für die Codierung von Ziffern und Zahlen gebräuchliche numerische Codes aufgelistet.

2.3. 1 Zählcodes

Zählcodes sind die einfachsten Binärcodes und lassen sich grob in Wort- und Zifferncodes einteilen.

* Wortcode

Die Wortlänge richtet sich nach der höchsten darzustellenden Dezimalzahl; das Gewicht eines Wortes entspricht dem dezimalen Wert.

Beispiel:

Wortlänge k = 100bit, darstellbarer Zahlenbereich 0,1,...,100

Dezimalzahl	Binärwort	Gewicht g
0	00 ... 0000	0
1	00 ... 0001	1
2	00 ... 0011	2
3	00 ... 0111	3
. . .	. . .	. . .
99	01 ... 1111	99
100	11 ... 1111	100

100bit #

* Zifferncode

Jede einzelne Ziffer wird über eine entsprechende Anzahl von 1-Bits binär codiert.

Beispielsweise wird hier für die Darstellung der Dezimalzahl

$$(Z)_{10} = b_n b_{n-1} \ldots b_1 b_0$$

eine Wortlänge von (n+1)·9bit benötigt.

Beispiel:

Dezimalziffer	Binärcodierte Ziffer
0	000 000 000
1	000 000 001
2	000 000 011
.	.
.	.
.	.
8	011 111 111
9	111 111 111

(Fernsprech-Vermittlungstechnik, Wählimpulse der Nummernscheibe)
#

2.3.2 Positionscodes

Die Position der gesetzten Bits innerhalb eines Wortes und nicht die Anzahl liefert die Verschlüsselung.

Positionscodes gliedern sich in bewertbare Codes und Anordnungscodes.

Ein *bewertbarer Code* liegt vor, wenn jeder Binärstelle i - analog zu polyadischen Zahlensystemen - eine definierte Wertigkeit w_i zugeordnet ist ($w_{i+1} > w_i$, d.h. aufsteigende Wertigkeit muß allerdings nicht vorliegen).

Anordnungscodes basieren auf Zuordnungsvorschriften, die sich mit Hilfe der zugehörigen Codetabellen einfach beschreiben lassen; die mathematischen Beziehungen sind teilweise recht kompliziert.

Beispiel: "*Dualcode, Gray-Code*"

	Dualcode					**Gray-Code**				
Stellengewicht	16	8	4	2	1	-	-	-	-	-
Dezimalzahl	D1	D2	D3	D4	D5	G1	G2	G3	G4	G5
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	1
2	0	0	0	1	0	0	0	0	1	1
3	0	0	0	1	1	0	0	0	1	0
4	0	0	1	0	0	0	0	1	1	0
5	0	0	1	0	1	0	0	1	1	1
6	0	0	1	1	0	0	0	1	0	1
7	0	0	1	1	1	0	0	1	0	0
8	0	1	0	0	0	0	1	1	0	0
9	0	1	0	0	1	0	1	1	0	1
10	0	1	0	1	0	0	1	1	1	1
11	0	1	0	1	1	0	1	1	1	0
12	0	1	1	0	0	0	1	0	1	0
13	0	1	1	0	1	0	1	0	1	1
14	0	1	1	1	0	0	1	0	0	1
.			.					.		
.			.					.		

Dualcode: Wichtung (Wertigkeit) der Stellen korrespondiert zur Darstellung im Dualsystem.

Gray-Code: nicht bewerteter Code, stetiger Code, Hamming-Distanz zweier benachbarter Codeworte ist konstant gleich 1,d.h. es liegt ein einschrittiger Code vor - es ändert sich jeweils nur ein Bit!

#

**** Natürlicher Binärcode (NBC)***

Der NBC - vgl. Dualcode - fußt ausschließlich auf der Zahlendarstellung im Dualsystem, das Codewort ist jeweils identisch mit der Stellenschreibweise der entsprechenden k-stelligen Dualzahl:

$$(Z)_2 = b_{k-1}\cdot 2^{k-1} + b_{k-2}\cdot 2^{k-2} + \ldots + b_0\cdot 2^0 = b_{k-1}b_{k-2}\ldots b_0 \ .$$

NBC

Dezimalzahl	Wertigkeit 8	4	2	1	Codelineal			
0	0	0	0	0				
1	0	0	0	1				■
2	0	0	1	0			■	
3	0	0	1	1			■	■
4	0	1	0	0		■		
5	0	1	0	1		■		■
6	0	1	1	0		■	■	
7	0	1	1	1		■	■	■
8	1	0	0	0	■			
9	1	0	0	1	■			■
10	1	0	1	0	■		■	
11	1	0	1	1	■		■	■
12	1	1	0	0	■	■		
13	1	1	0	1	■	■		■
14	1	1	1	0	■	■	■	
15	1	1	1	1	■	■	■	■

Der NBC wird für einfache Zähl- und Rechenoperationen eingesetzt; eine wesentliche Einschränkung resultiert aus der gleichzeitigen Änderung mehrerer Bits beim Übergang zwischen benachbarten Codeworten, so daß bei einer technischen Abtastung beispielsweise unter Verwendung von Photozellen kurzzeitig eine Fehlinformation auftritt.

**** Einschrittige Codes***

Bei *einschrittigen Codes* ändert sich beim Übergang von einem Codewort zum nächstfolgenden immer nur *ein Bit*.

Einschrittige Codes:

Gray-Code , BCD-Codes (4-Bit-Codes) , 5-Bit-Codes , Dekadische Codes , ...

Jeder Code besitzt anwendungsspezifische Vorteile, z.B. bzgl. der Addition, Subtraktion, Komplementbildung, Fehlererkennung, etc..

**** Gray-Code , Binärer Reflexcode (BRC)***

Das wesentliche Charakteristikum des Gray-Codes ist seine Einschrittigkeit; diese bestimmt den Einsatz im Rahmen der Meßtechnik beispielsweise zur Längenerfassung oder Winkelmessung mittels Codierscheibe.

Beim Übergang zur benachbarten Ziffer ändert sich jeweils nur ein Bit, d.h. die Hamming-Distanz ist konstant: D = 1 .

Das Bildungsgesetz des Gray-Codes basiert auf der fortgesetzten Reflexion bzw. Spiegelung vorhandener Worte:
Beginnend mit dem Anfangswert(0000) werden die vorhandenen Ziffern des letzten Codewortes an die nächsthöhere -auf 1 gesetzte- Bit-stelle angefügt, die Ziffern des vorletzten Codewortes bilden die letzten neuen Ziffern des folgenden Codewortes, usw. .

Gray-Code								
Dezimalzahl	Bit x_3	x_2	x_1	x_0	Codelineal			
0	0	0	0	0				
1	0	0	0	1				
2	0	0	1	1				
3	0	0	1	0				
4	0	1	1	0				
5	0	1	1	1				
6	0	1	0	1				
7	0	1	0	0				
8	1	1	0	0				
9	1	1	0	1				
10	1	1	1	1				
11	1	1	1	0				
12	1	0	1	0				
13	1	0	1	1				
14	1	0	0	1				
15	1	0	0	0				

**** 4-Bit-BCD-Codes (Tetradencodes)***

4-Bit-Codes werden allgemein als *tetradische* Codes bezeichnet; die für die Darstellung der Ziffern 0,1,...,9 nicht genutzten sechs Codeworte heißen *Pseudotetraden* (Pseudodezimalen).
Die Anzahl der möglichen 4-Bit-BCD-Codes beträgt $2.9 \cdot 10^{10} = 16!/6!$; allerdings besizt nur eine geringe Anzahl eine praktische Relevanz.

Lexikographische BCD-Codes ergeben sich direkt aus dem natürlichen Binärcode (NBC), indem die Plazierung der Pseudotetraden variiert wird und die natürliche Sequenz der Ziffern erhalten bleibt.

Gebräuchliche BCD-Codes:

	8-4-2-1-Code				**BCD-Gray-Code**				**Aiken-Code**				**3-Exzeß-Code**			
Wertigkeit	8	4	2	1	-	-	-	-	2	4	2	1	-	-	-	-
Dezimalziffer	x_3	x_2	x_1	x_0	x_3	x_2	x_1	x_0	x_3	x_2	x_1	x_0	x_3	x_2	x_1	x_0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	0	0	1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	0	1	1	0	0	1	0	0	1	0	1
3	0	0	1	1	0	0	1	0	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	0	0	1	0	0	0	1	1	1
5	0	1	0	1	0	1	1	1	1	0	1	1	1	0	0	0
6	0	1	1	0	0	1	0	1	1	1	0	0	1	0	0	1
7	0	1	1	1	0	1	0	0	1	1	0	1	1	0	1	0
8	1	0	0	0	1	1	0	0	1	1	1	0	1	0	1	1
9	1	0	0	1	1	1	0	1	1	1	1	1	1	1	0	0
Pseudotetrade	1	0	1	0	1	1	1	1	0	1	0	1	0	0	0	0
"	1	0	1	1	1	1	1	0	0	1	1	0	0	0	0	1
"	1	1	0	0	1	0	1	0	0	1	1	1	0	0	1	0
"	1	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1
"	1	1	1	0	1	0	0	1	1	0	0	1	1	1	1	0
"	1	1	1	1	1	0	0	0	1	0	1	0	1	1	1	1

**** 8-4-2-1 - Code***

Der bewertbare 8-4-2-1 - Code stellt die Restriktion des NBC-Codes auf eine Dekade dar.

Eigenschaften:

+ leichte Identifizierung gerader und ungerader Zahlen
  (1 in der letzten Bitstelle kennzeichnet eine ungerade Zahl),

+ einfaches Bildungsgesetz (leichte Speicherung),

+ einfache Addition - solange kein Übertrag in die nächste Dekade auftritt,

- 0-Wort ist vorhanden,

- der Code ist unsymmetrisch (ungünstig für die Komplementbildung), deshalb für Rechenschaltungen ungeeignet.

**** Exzeß-3-Code (Stibitz-Code)***

Symmetrischer (nicht bewerteter) Anordnungscode wobei die Pseudodezimalen in zwei Dreierblöcken angeordnet sind.

Bildungsgesetz:

$$(Z)_{10} = a_3 \cdot 2^3 + a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0 - 3 =$$
$$= a_3 \cdot 8 + a_2 \cdot 4 + a_1 \cdot 2 + a_0 \cdot 1 - 3$$

$[(4)_{10} = (0111)_{3\text{-Exze}\ss} = 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 - 3 = 4 + 2 + 1 - 3\ ]$

Eigenschaften:

\+ gerade und ungerade Zahlen sind leicht unterscheidbar
(1 in der letzten Bitposition liefert eine gerade Zahl),

\+ einfache Rundungsanzeige
(1 in der werthöchsten Stelle heißt (Zahl 5),

\+ 0- und 1-Wort fehlen,

\+ Komplementbildung durch Vertauschen von 0 und 1.

**** Aiken-Code (2-4-2-1 - Code)***

Die Tetradenstellen dieses symmetrischen Codes besitzen die Wertigkeit 2-4-2-1, d.h. der Code ist nicht eindeutig umkehrbar (die Wichtung 2 tritt zweimal auf).

Eigenschaften:

\+ gerade und ungerade Zahlen sind leicht erkennbar,

\+ einfache Neuner-Komplementbildung (Invertierung 0 1),

\- 0- und 1-Wort sind vorhanden.

**** 5-Bit-BCD-Codes (Pentadische Codes)***

Codierung der Dezimalziffern mit 5-stelligen Binärworten (Pentaden); die Anzahl der Codeworte beträgt $2^5 = 32$, so daß insgesamt 22 Pseudopentaden auftreten und die Redundanz den Wert $R = ld\ 32 - ld\ 10 = 1{,}7$ bit besitzt.

**** Gleichgewichtete Codes (g aus k - Codes)***

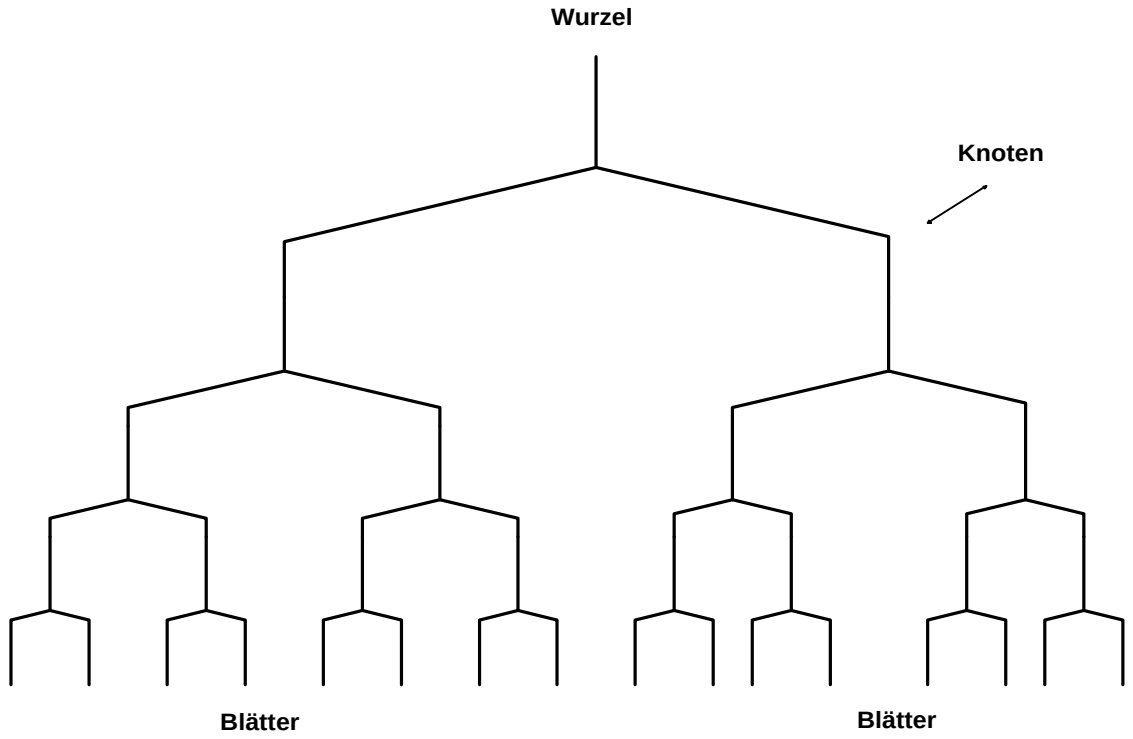
Bei gleichgewichteten Codes haben alle Codeworte konstantes Gewicht, d.h. von den k Stellen sind durchgängig g Stellen mit 1 belegt; die Anzahl der Codeworte folgt hier mit dem Binomialkoeffizienten:

$$\binom{k}{g} = \frac{k!}{g!\ (k - g)!} \quad .$$

Bemerkung: **‘ Codebäume ‘**

Neben den Codelinealen bilden Codebäume ein beliebtes Hilfsmittel, um die Struktur eines Codes geometrisch zu veranschaulichen.

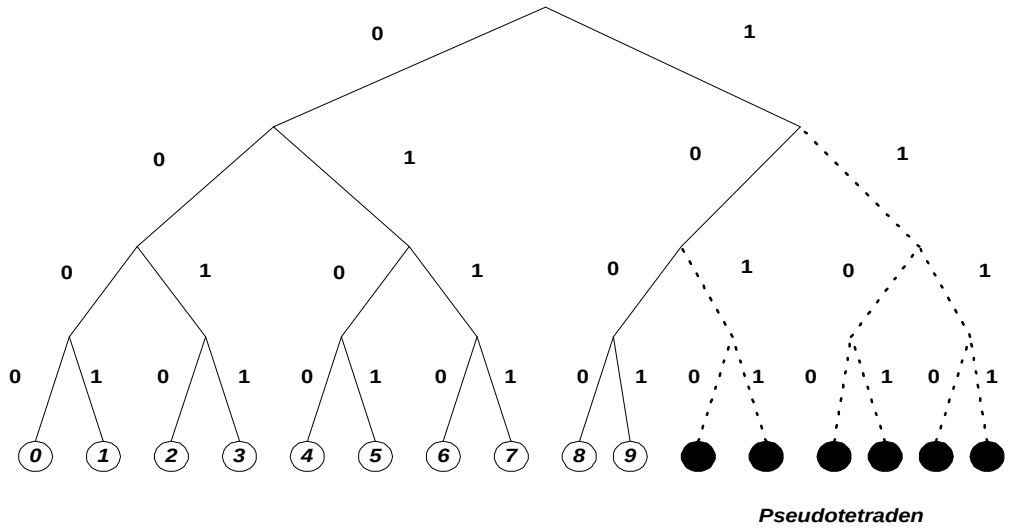
Ein Codebaum besteht aus einer *Wurzel* - die in der Informatik häufig oben gezeichnet wird -, den Verzweigungen, die auch als *Knoten* bezeichnet werden, den Endknoten oder *Blättern* und den *Kanten*, das sind die Verbindungen zwischen den Knoten; diese Art von Baum wird in der Graphentheorie als gerichteter azyklischer Graph bezeichnet.



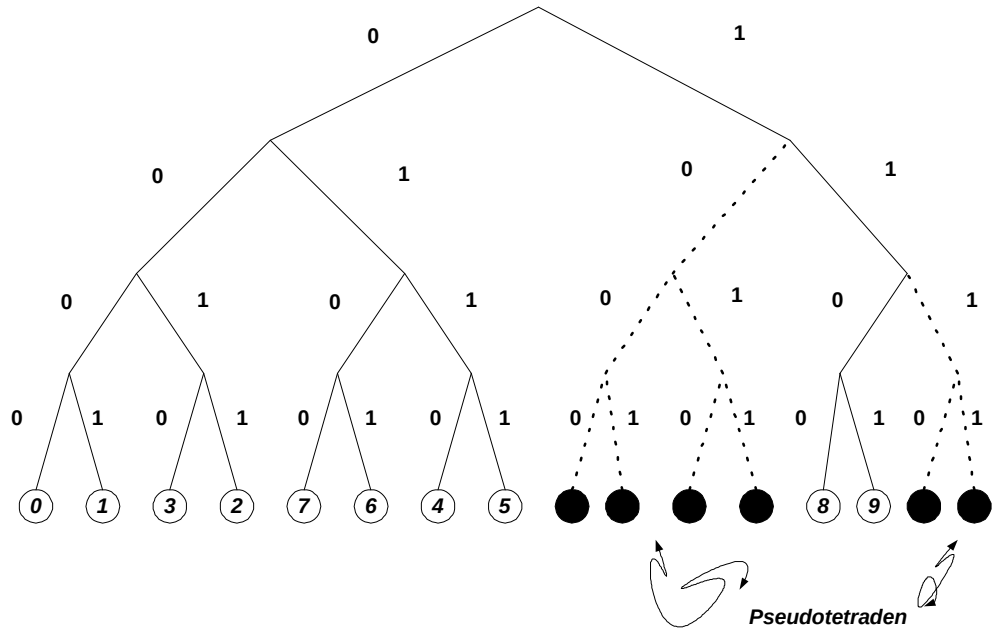
Die Codierung bzw. Decodierung eines Zeichens folgt, indem ausgehend von der Wurzel bis zu dem Blatt für das betreffende Zeichen der Baum durchlaufen wird und entlang des Weges beispielsweise die Binärzeichen (mit denen die Kanten beschriftet sind) als Codesequenz notiert werden.

Beispiele:

BCD - Tetradencode



Gray - Code



#

2.4 ASCII - Code

Der ASCII - Code [ASCII: American Standard Code for Information Interchange] bildet den alphanumerischen Standardcode für die binäre Verschlüsselung von Buchstaben, Ziffern und - der Steuerung dienenden - Sonderzeichen.

Die deutsche Version mit den Zeichen §, Ä, Ö, Ü, ä, ü, ö, ß ist in der DIN-Vorschrift 66003 genormt - vgl. auch ISO-7-Bit-Code.
[ISO: International Standardization Organisation]

		Spaltenverschlüsselung							
	H	0	1	2	3	4	5	6	7
Zeilenverschlüsselung	A6	0	0	0	0	1	1	1	1
	A5	0	0	1	1	0	0	1	1
	A4	0	1	0	1	0	1	0	1
H	A3 A2 A1 A0								
0	0 0 0 0		NUL	DLE	SP	0	@ §	P	`	p
1	0 0 0 1		SOH	DC1	!	1	A	Q	a	q
2	0 0 1 0		STX	DC2	"	2	B	R	b	r
3	0 0 1 1		ETX	DC3	#	3	C	S	c	s
4	0 1 0 0		EOT	DC4	$	4	D	T	d	t
5	0 1 0 1		ENQ	NAK	%	5	E	U	e	u
6	0 1 1 0		ACK	SYN	&	6	F	V	f	v
7	0 1 1 1		BEL	ETB	'	7	G	W	g	w
8	1 0 0 0		BS	CAN	(	8	H	X	h	x
9	1 0 0 1		HT	EM	)	9	I	Y	i	y
A	1 0 1 0		LF	SUB	*	:	J	Z	j	z
B	1 0 1 1		VT	ESC	+	;	K	[Ä	k	{ ä
C	1 1 0 0		FF	FS	´	<	L	\ Ö	l	| ö
D	1 1 0 1		CR	GS	-	=	M	] Ü	m	} ü
E	1 1 1 0		SO	RS	.	>	N	^	n	~ ß
F	1 1 1 1		SIX	US2	/	?	O	_	o	DEL

Bei der tabellarischen (matrixanalogen) Darstellung liefern die Bits A4, A5, A6 die Spaltenverschlüsselung und die Bits A0, A1, A2, A3 die Zeilenverschlüsselung:

A0	A1	A2	A3	A4	A5	A6		
1	1	1	1	0	1	1	o	$6FH = (6F)_{16}$.
Zifferntteil				*Zonenteil*				

Mit 7 Bit lassen sich 128 Zeichen verschlüsseln; ASCII-Zeichen werden im Rechner zumeist rechtsbündig in einem Byte gespeichert, wobei das freie Bit (MSB) dann für die Datensicherung (Paritätsbit) bzw. für die Erweiterung des Zeichensatzes auf 256 Codewörter (IBM-ASCII) verwendet wird.

Die Tastatureingabe der ASCII-Zeichen erfolgt mit Hilfe der Alternate -<Alt>- Taste in Verbindung mit der Dezimalverschlüsselung:

H <Alt> 72

$$(64 + 8)_{10} = (48)_{16}$$

bzw. für die in den beiden ersten Spalten plazierten nicht-darstellbaren Sonderzeichen mittels Control -<CTRL>- Taste und Buchstabentaste aus den Spalten 4 oder 5:

BEL $(07)_{16}$ <CTRL> G .

Eine Zusammenstellung und Erläuterung der Sonderzeichen des ASCII-Codes liefert die folgende Tabelle:

Dezimal-Äquivalent	Sedezimal-Äquivalent	ASCII-Zeichen	Bedeutung	
00	00	NUL	Null	Füllzeichen
01	01	SOH	Start of Heading	Anfang des Kopfes
02	02	STX	Start of Text	Anfang des Textes
03	03	ETX	End of Text	Ende des Textes
04	04	EOT	End of Transmission	Ende der Übertragung
05	05	ENQ	Enquiry	Stationsaufforderung
06	06	ACK	Acknowledge	Positive Rückmeldung
07	07	BEL	Bell	Klingel
08	08	BS	Backspace	Rückwärtsschritt
09	09	HT	Horizontal Tabulation	Horizontal-Tabulator
10	0A	LF	Line Feed	Zeilenvorschub
11	0B	VT	Vertical Tabulation	Vertikal-Tabulator
12	0C	FF	Form Feed	Formularvorschub
13	0D	CR	Carriage Return	Wagenrücklauf
14	0E	SO	Shift Out	Dauerumschaltung
15	0F	SI	Shift In	Rückschaltung
16	10	DLE	Data Link Escape	Datenübertragung-Umschalt.
17	11	DC1	Device Control 1	Gerätesteuerung 1
18	12	DC2	Device Control 2	Gerätesteuerung 2
19	13	DC3	Device Control 3	Gerätesteuerung 3
20	14	DC4	Device Control 4	Gerätesteuerung 4
21	15	NAK	Negative Acknowledge	Negative Rückmeldung
22	16	SYN	Synchronous Idle	Synchronisierung
23	17	ETB	End of Transmission Block	Ende des Übertrag.-Blocks
24	18	CAN	Cancel	Ungültig machen
25	19	EM	End of Medium	Ende der Aufzeichnung
26	1A	SUB	Substitute	Substitution
27	1B	ESC	Escape	Umschaltung
28	1C	FS	File Separator	Hauptgruppen-Trennung
29	1D	GS	Group Separator	Gruppen-Trennung
30	1E	RS	Record Separator	Untergruppen-Trennung
31	1F	US	Unit Separator	Teilgruppen-Trennung
32	20	SP	Space	Zwischenraum, Leerschritt
127	7F	DEL	Delete	Löschen

2.5 Fehlererkennung, Fehlerkorrektur, Übertragungssicherung

Die Verwendung von Prüfbits bzw. Prüfworten ermöglicht, Übertragungsfehler zu erkennen und diese anhand von Zusatzinformationen aufgrund einer redundanten Verschlüsselung zu beseitigen.

Die Übertragung redundanter Daten impliziert eine Reduktion des Informationsgehalts bzw. einen erhöhten Aufwand.

Als grundlegendes Maß, das die fehlerbehaftete Datenübertragung charakterisiert, wird die *Bitfehlerwahrscheinlichkeit* verwendet:

$$p_B := \lim_{b \to} \left(\frac{b_f}{b} \right) \quad ,$$

hierbei bezeichnet b_f die Zahl der fehlerhaften Bits und b die Anzahl der gesendeten Bits; näherungsweise gilt:

$$p_B \quad \left\{ \begin{array}{l} 10^{-5} \quad , \textit{Fernsprechleitungen} \\ 10^{-9} \quad , \textit{Koaxialkabel} \\ 10^{-12} \quad , \textit{Lichtwellenleiter} \end{array} \right\} \quad ,$$

d.h. auf 10^9 Bits kommt ein fehlerhaftes bei Verwendung von Koaxialkabeln.

In der Praxis spielen selbstverständlich umfassendere Begriffsbildungen wie Wort- und Blockfehlerwahrscheinlichkeit sowie Restfehlerrate eine hervorragende Rolle - vgl. Literatur.

Beispiel: *Halbleiterspeicher* (ROM, EPROM, RAM)

Auftretende Hardwarefehler (Hard Errors, permanente Fehler) lassen sich im allgemeinen leicht erkennen und beheben.
Softwarefehler (Soft Errors, flüchtige Fehler) können dagegen leicht zu Schwierigkeiten führen: betrachtet man die Invertierung einer Speicherzelle (0 1) durch Störimpulse oder radioaktive Strahlung wie " - Teilchen, die durch radioaktive Atomkerne im Chip entstehen, so beträgt die Wahrscheinlichkeit bezogen auf eine einzelne Zelle 1 zu 1 Millionen Jahre; bei 1 MByte-Speicherkapazität (2^{23} bit = 8 388 608 bit) liegt die mittlere Wahrscheinlichkeit für das Auftreten eines 1-Bit-Softwarefehlers bei circa 45 Tagen, d.h. eine Fehlerkorrektur beispielsweise mittels Prüfbits bietet sich an!

#

Standardverfahren zur Fehlererkennung und -korrektur:

- Querparität (VRC: Vertical Redundancy Check)
- Längsparität (LRC: Longitudinal Redundancy Check)
- Kreuzsicherung (Kombination aus Quer- und Längsparität)
- Zyklische Blocksicherung (CRC: Cyclic Redundancy Check)
- Hamming - Codes

**** Paritätsprüfung***

Das einfachste Verfahren zur Fehlererkennung basiert auf dem Anhängen eines Prüfbits an das Datenwort: dem Datenwort wird ein Paritätsbit p (Parity-Bit, Prüfbit) hinzugefügt und die Quersumme aller Einsen (modulo 2) des Datenworts wird jeweils überprüft.
Grundsätzlich wird zwischen gerader und ungerader (even / odd) Parität unterschieden.

Gerade Parität:
Die Quersumme aller Einsen (modulo 2) des Datenwortes inklusive Prüfbit ist gerade, d.h.

$$p := \begin{cases} 0 \ , & \textit{Gewicht g des Wortes ist gerade} \\ 1 \ , & \textit{Gewicht g des Wortes ist ungerade} \end{cases} .$$

Beispiel: *8-4-2-1-BCD-Code*

Dezimal-zahl	**8 4 2 1**	**p**	**ungerade Parität**
0	0 0 0 0	0	1
1	0 0 0 1	1	0
2	0 0 1 0	1	0
.	.	.	.
.	.	.	.
.	.	.	.

#

Bei gleichgewichtigen Codes erübrigt sich natürlich die Übertragung eines Paritätsbits, jedoch liefert eine Prüfung z.B. 1-Bit-Fehler.

Sensitiv ist die Querparitätsprüfung nur in bezug auf eine ungerade Anzahl von Bitfehlern; eine Fehlerkorrektur ist hier grundsätzlich

wegen der fehlenden Lokalisierbarkeit nicht möglich.

In der PC-Technik wird mit der Paritätsprüfung gearbeitet und jedes Byte mit einem Paritätsbit versehen. Durch eine Paritätsschaltung wird dieses Paritätsbit automatisch erzeugt, wenn ein Byte in einen Schreib-Lese-Speicher - z.B. SRAM-Baustein im 9-Bit-Format (SRAM: statische RAM-Speicherzelle , RAM: Random Access Memory) - eingeschrieben wird. Beim Auslesen eines Bytes überprüft die Paritätsschaltung wiederum automatisch, ob die Überprüfung richtig ausgeführt worden ist. Es werden nur Einfachfehler erkannt; tritt ein Doppelfehler auf, arbeitet das System weiter. IBM hat die Paritätsprüfung mit gerader Quersummenbildung praktisch standardisiert, d.h.
nahezu alle PC-Systeme arbeiten mit dieser Sicherungsmethode.

**** Kreuzsicherung***

Hierbei handelt es sich um eine Kombination aus Quer- und Längsparitätsprüfung, die bei blockweiser (bitparallele - zeichenserielle) Übertragung eingesetzt wird.
Nachfolgend wird eine Übertragungssicherung mittels gerader Längs- und Querparität exemplarisch für einen Einfachfehler erläutert:

	Zeichen	8-Bit	VRC
	1	1 0 1 1 0 1 0 1	1
	2	1 0 1 0 1 1 0 1	1
Block	3	-0-0-0-**1**-1-0-1-0- - - - - - - - - - -	**0**
	4	1 1 1 0 1 0 1 0	1
	LRC	1 1 1 **1** 1 0 0 0	.

**** Zyklische Blocksicherung*** (CRC - Prüfsummen - Verfahren)

Das CRC-Verfahren (CRC: Cyclic Redundancy Check) wird beispielsweise zur Datensicherung bei Disketten und Festplatten eingesetzt. Viele FDC-Einheiten (FDC: Floppy Disk Controller) verfügen über eine CRC-Hardware, die dieses Kontrollverfahren automatisch ausführt.
Für jeden Datenblock wird eine Prüfsumme (CRC-Prüfwort, FCS: Frame Check Sequence) - die 16 oder 32 Bit umfasst - mit Hilfe eines Algorithmus berechnet und gemeinsam mit dem Datenblock übertragen; im Empfänger wird anhand der Daten die Prüfsumme erneut berechnet (Hardware-Realisierung) und bei Übereinstimmung beider Prüfsummen die Übertragung als fehlerfrei betrachtet.

○ Ziel: Verwendung spezieller Prüfsummen derart, daß "alle" Fehler bezogen auf einen Block registriert werden!

○ Lösung: Erzeugende Generatorpolynome (primitive Polynome)

[Beispiele:

CRC - 12 , $G(x) = x^{12} + x^{11} + x^8 + x^3 + x^2 + 1$;

CRC - 16 , $G(x) = x^{16} + x^{15} + x^2 + 1$;

CRC - CCIT , $G(x) = x^{16} + x^{12} + x^5 + 1$;

. .
. .
32-Bit
. .]

(CCIT: Comité Consultatif International Télégraphique et Téléphonique)

Das CCIT-Polynom wird bei den Übertragungsprotokollen von XMODEM, X25, Kermit, etc. eingesetzt; CRC-16 dient zur Sicherung des EBCDI-Codes in der Großrechnertechnik.

○ Basis: Modulo-2-Arithmetik

" + " 0 + 0 = 0 , 1 + 1 = 0 ,
0 + 1 = 1 , 1 + 0 = 1 ,

" - " 0 - 0 = 0 , 1 - 1 = 0 ,
0 - 1 = 1 , 1 - 0 = 1 ,

" * " 0 * 0 = 0 , 1 * 1 = 1 ,
0 * 1 = 0 , 1 * 0 = 0 ,

" / " 0 : 1 = 0 , 1 : 1 = 1 ;

Addition und Subtraktion stellen hierbei identische Operationen dar und lassen sich bequem über ein *exklusives Oder-Schaltglied* realisieren.

Binäre Information kann über Polynome, deren Koeffizienten 0 bzw. 1 sind, dargestellt werden; alternativ lassen sich Polynome durch Bitsequenzen parametrisieren:

Nachricht		Polynom
11 0000 1101	-->	$x^9 + x^8 + x^3 + x^2 + 1$,
CCIT-Polynom	-->	1000 1000 0001 0000 1 (x^{16} x^{12} x^5 1) .

Beispielsweise gilt dann:

$$x^2 + 1 = (x + 1)\cdot(x + 1)$$

$$101 = 11 \cdot 11 = 11$$
$$ \ 11$$
$$ \ ---$$
$$ \ 101 \quad \text{(mod-2-Arithmetik).}$$

CRC - Algorithmus

Gegeben seien ein Generatorpolynom G(x) mit Grad M und eine Sequenz von N Datenbits, die über ein binäres Datenpolynom S(x) dargestellt werden, Grad S N-1 :

1. Multiplikation von S(x) mit x^M , d.h. Anfügen von M 0-Bits (Raum für die CRC-Prüfsumme)

2. Division
$$(S(x)\cdot x^M) : G(x) = Q(x) + R(x)/G(x)$$

⇒ Quotient Q(x) (bedeutungslos),

⇒ Rest R(x) - Polynom vom Grad N-1 - ist die CRC-Prüfsumme und umfaßt M-Bits.

Der gesamte Übertragungsblock setzt sich dann aus N Datenbits und M CRC-Bits zusammen.

Überprüfung beim Empfänger:
Berechnung der CRC-Prüfsumme aus den Datenbits und Vergleich mit der gesendeten Prüfsumme,

oder alternativ

Berechnung der CRC-Prüfsumme für den gesamten Block bestehend aus (N+M)-Bits, wobei hier bei fehlerfreier Übertragung der Rest verschwindet.

{ $S(x)\cdot x^M + R(x)$ ist das Polynom für den gesamten Block (N+M Bits) und nach Konstruktion gilt:
$$S(x)\cdot x^M = Q(x)\cdot G(x) + R(x) \quad ;$$

⇒ $S(x)\cdot x^M + R(x) = Q(x)\cdot G(x) + R(x) + R(x) = Q(x)\cdot G(x)$

ist ein Vielfaches von G(x),

⇒ $(S(x)\cdot x^M + R(x)) : G(x) = Q(x)$ mit Rest Null ! }

Beispiel:

N = 8, M = 8,
$G(x) = x^8 + 1$ (1 0000 0001),
$S(x) = x^7 + x^5 + x$ (1010 0010),

1. $S(x)\cdot x^8 = x^{15} + x^{13} + x^9$,

2. $(S(x)\cdot x^8) : G(x)$

$$
\begin{array}{l}
x^{15} + x^{13} + x^9 \;:\; (x^8 + 1) = x^7 + x^5 + x \\
-x^{15} - x^7 \\
\hline
x^{13} + x^9 - x^7 \\
-x^{13} - x^5 \\
\hline
x^9 - x^7 - x^5 \\
-x^9 - x \\
\hline
-x^7 - x^5 - x
\end{array}
$$

$\Rightarrow$ $R(x) = -x^7 - x^5 - x = x^7 + x^5 + x$ { mod.-2-Arith. }

Übertragen wird:

$$S(x)\cdot x^8 + R(x) = \underbrace{x^{15} + x^{13} + x^9}_{\text{Daten}} + \underbrace{x^7 + x^5 + x}_{\text{CRC-Prüfsumme}}$$

Überprüfung beim Empfänger:

$(S(x)\cdot x^8 + R(x)) : G(x) =! Q(x)$ $R(x) = 0$

$$
\begin{array}{l}
x^{15} + x^{13} + x^9 + x^7 + x^5 + x \;:\; (x^8 + 1) = x^7 + x^5 + x \\
-x^{15} \qquad\qquad -x^7 \\
\hline
x^{13} + x^9 + x^5 + x \\
-x^{13} \qquad -x^5 \\
\hline
x^9 + x \\
-x^9 - x \\
\hline
0
\end{array}
$$

$\Rightarrow$ $R(x) = 0$, d.h. es liegt eine fehlerfreie Übertragung vor! #
Das CRC-16-Verfahren erlaubt, bis zu 16 aufeinanderfolgende Bitfehler eindeutig und längere Bitfehlersequenzen mit einer Sicherheit von 99,997% zu erkennen.

Technisch läßt sich das CRC-Verfahren relativ einfach (Modulo-2-Arithmetik) über Schieberegister (Flipflops) und XOR-Gatter realisieren - Festplattenkontroller verfügen über eine CRC-Hardware, die das Prüfverfahren automatisch ausführt.

Die CRC-Prüfung zwischen zwei Ethernet-Controllern basiert auf einer 32-Bit-Prüfsumme, die mit dem Prüfpolynom

$$G(x) = x^{32} + x^{25} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^6 + x^5 + x^4 + x^2 + x + 1$$

gebildet wird.

**** Hamming - Codes***

Hamming-Codes sind fehlerkorrigierende Codes mit mehreren Prüfbits, die die eindeutige Lokalisierung von Übertragungsfehlern ermöglichen.

Die Voraussetzung für eine Korrektur von n-Bit-Fehlern ist eine Minimaldistanz $d_{min} = 2n + 1$, d.h. n fehlerhafte Bits liefern kein anderes gültiges Codewort.

Hamming-Codes werden über die Wortlänge k (= Summe der Daten- plus Prüfbits) und die Anzahl d der Datenbits klassifiziert:

(k,d) - Hamming - Code ;

z.B.: (7,4)- , (64,57)-Hamming-Code .

Die Anzahl der erforderlichen Prüfbits p, um bei d Datenbits alle Einzelfehler zu korrigieren, wird durch die Ungleichung

$$2^p \quad d + p + 1$$

festgelegt:

Datenbits d	1 - 4	5 - 11	12 - 26	. . .	121 -247
Prüfbits p	3	4	5	. . .	8

.

Verteilung der Prüfbits über das Codewort:

- die Bitpositionen des Codewortes werden von links nach rechts mit 1 beginnend durchnummeriert,
- alle Positionen, deren Nummer eine Potenz von 2 ist (1,2,4,8,...) sind Prüfbits,
- alle anderen Bits (3,5,6,7,...) sind Datenbits.

Wesentlich ist nun, daß jedes einzelne Datenbit für die Berechnung verschiedener Prüfbits benutzt wird, d.h. ein Übertragungsfehler wirkt sich immer auf mehrere Prüfbits aus - tritt ein einzelner Fehler bei einem Prüfbit auf, so muß dieses selbst fehlerhaft sein.

Beispiel: *(7,4) - Hamming - Code*

4 Datenbits (D), 3 Prüfbits (P),

Codewort: 1 2 3 4 5 6 7
P P D P D D D ,

Datenbits: x_3, x_5, x_6, x_7 ,

Prüfbits: p_1, p_2, p_4 ,

Berechnung der Prüfbits:

{Modulo-2-Arithmetik: 0 0 = 0 , 1 1 = 0 , 1 0 = 1 , 0 1 = 1}

$p_1 = x_3 \quad x_5 \quad x_7$,
$p_2 = x_3 \quad x_6 \quad x_7$,
$p_4 = x_5 \quad x_6 \quad x_7$,

Datenwort: x_3 x_5 x_6 x_7
0 1 1 0

⇒ $p_1 = 0 \quad 1 \quad 0 = 1$,
$p_2 = 0 \quad 1 \quad 0 = 1$,
$p_4 = 1 \quad 1 \quad 0 = 0$,

Übertragen wird das Codewort: 1 1 0 0 1 1 0 (gerade Parität!),
Empfangen werde (Bit 5 gestört): 1 1 0 0 **0** 1 0 ,

Paritätsberechnung beim Empfänger:

$p_1 \quad x_3 \quad x_5 \quad x_7 = 1 \quad 0 \quad 0 \quad 0 = 1$,
$p_2 \quad x_3 \quad x_6 \quad x_7 = 1 \quad 0 \quad 1 \quad 0 = 0$,
$p_4 \quad x_5 \quad x_6 \quad x_7 = 0 \quad 0 \quad 1 \quad 0 = 1$,

⇒ die Berechnungen der Prüfbits p_1 und p_4 zeigen einen Fehler des Bits 5 = 1 + 4 an, denn es ist das Datenbit, das von den Prüfbits p_1 und p_4 kontrolliert wird.

#

Bei Hamming-Codes stellt jedes Prüfbit ein Parity-Bit für eine bestimmte Menge von Datenbits dar; ein bestimmtes Datenbit beeinflußt verschiedene Prüfbits. Um festzustellen, zu welchen Prüfbits ein bestimmtes Bit, z.B. das mit der Nummer k, einen Beitrag liefert, zerlegt man k in eine Summe aus Zweierpotenzen (k=11=8+2+1, k=29=16+8+4+1). Ein Bit liefert zu allen Prüfbits einen Beitrag, deren Nummer in dieser Zerlegung auftritt - Bit 11 wird beispielsweise für die Berechnung der Prüfbits 1,2 und 8 eingesetzt.

Für die Korrektur von Mehrfachfehlern sind selbstverständlich weitere Prüfbits erforderlich!

2.6 Informationstheorie und Datenkompression

- Einführende Bemerkungen

Die Kompression digitaler Daten bildet die essentielle Grundlage für Multimedia-Anwendungen, z.B. im Zusammenhang mit dem Internet, der mobilen Kommunikation, Video-Konferenzen, digitalem Fernsehen, etc.; Verfahren wie JPEG und MPEG zur Kompression von Bildern und Videos benutzen effektive Algorithmen zur Datenreduktion und eröffnen eine praktikable Möglichkeit für die Speicherung und Übertragung der Daten.

Die nichtkomprimierte Darstellung eines Audio-Signals über einen Zeitraum von einer Minute in CD-Qualität - d.h. 44100 Abtastwerte/s, wobei jeder Signalwert mit 16 Bit codiert wird - umfasst ca. 42 Millionen Bits, d.h. ohne Datenkompression und der hiermit verknüpften Rekonstruktion der komprimierten Daten ist ein sinnvoller Einsatz nicht möglich.

In Abhängigkeit von den Anforderungen an den Rekonstruktionsprozess unterscheidet man zwischen *verlustfreien* (informationserhaltenden) und *verlustbehafteten* (informationsmindernden) Kompressionstechniken.

Die verlustfreie Kompression liefert eine Datenreduktion ohne jeglichen Informationsverlust, so dass sich die Originaldaten exakt aus den komprimierten Daten rekonstruieren lassen. Bedeutende Anwendungsgebiete verlustfreier Kompressionsverfahren bilden die Kompression von Text und medizinischen Daten.

Die verlustbehaftete Kompression ermöglicht aufgrund des beabsichtigten Informationsverlustes keine exakte Rekonstruktion der Originaldaten. Auf der anderen Seite liefern die verlustbehafteten Verfahren weit größere Kompressionsraten als die verlustfreien. Insbesondere bei der Bilddaten- und Video-Kompression läßt sich der Informationsverlust tolerieren, wenn hiermit keine wesentliche Beeinträchtigung des Rekonstruktionsergebnis einhergeht.

Die Bewertung eines Kompressionsverfahrens unterliegt unterschiedlichen Kriterien. Neben Laufzeit, Speicherplatzbedarf und Komplexität des Algorithmus besitzt das *Kompressionsverhältnis*, das ist das Verhältnis der Anzahl der Bits zur Darstellung der Daten vor und nach der Kompression, eine hervorragende Bedeutung. Beispielsweise sind für die Speicherung eines Bildes mit 256x256-Bildpunkten und 8 Bits für die Graustufenverteilung pro Bildpunkt insgesamt 65536 Bytes erforderlich; beläuft sich der Datenumfang nach der Kompression auf 16384 Bytes, so beträgt das Kompressionsverhältnis 4:1, d.h. die Kompression reduziert den Datenumfang auf 25% der Originaldaten. Naturgemäß ist die Bewertung verlustbehafteter Kompressionsverfahren schwieriger, da aufgrund des Informationsverlustes hier eine Approximation des Originals rekonstruiert wird und bei Anwendungen wie der Bilddatenkompression subjektive Beurteilungsfaktoren eine Rolle spielen.

- Shannons Informationstheorie

Im folgenden werden einige grundlegende Elemente der Informationstheorie eingeführt; diese Theorie wurde durch C. E. Shannon 1948 begründet. Im Mittelpunkt seiner fundamentalen Arbeit steht die quantitative Beschreibung von Information und es war Shannon, der mit Hilfe etablierter Begriffsbildungen aus dem Gebiet der Statistischen Physik erkannte, daß nur ein logarithmisches Maß diese quantitative Beschreibung liefern kann. Shannon definierte eine Größe, die als *Informationsgehalt* bezeichnet und über die Wahrscheinlichkeit für das Auftreten eines Ereignisses festgelegt wird.

Betrachten wir ein Ereignis A als Ergebnis eines Zufallsexperiments und bezeichnet p(A) die Wahrscheinlichkeit, daß das Ereignis A eintritt, so berechnet sich der Informationsgehalt h(A), der mit A verknüpft ist, anhand der Definition:

$$h(A) := ld\ \frac{1}{p(A)} = -ld\ p(A) \quad .$$

$$ld\ x := \log_2 x = \frac{\ln x}{\ln 2}$$

Grundsätzlich ist bei der Berechnung des Informationsgehalts h(A) für die Wahrscheinlichkeiten

$$0 \leq p(A) \leq 1$$

zu beachten; weiterhin ist h(A) auf dem Intervall [0,1] eine streng monoton fallende Funktion.

Beispiele: *Informationsgehalt*

* Würfeln mit einem nichtmanipulierten Würfel

  Ereignismenge {1,2,3,4,5,6}, die Wahrscheinlichkeit für das Ereignis $i \in \{1,2,3,4,5,6\}$ beträgt p(i) = 1/6 und der Informationsgehalt, der mit dem Ereignis i verknüpft ist, berechnet sich zu:

$$h(i) = -ld\ p(i) = \frac{\ln 6}{\ln 2} = 2.58\ bit.$$

Beachte: Der Informationsgehalt ist eine einheitenbehaftete Größe, die Einheit von h ist das bit!

* Werfen einer Münze

  Ereignismenge {Kopf,Zahl}, die Wahrscheinlichkeit für das Ereignis i $\in$ {Kopf,Zahl} beträgt p(i) = 1/2 und der Informationsgehalt, der mit dem Ereignis i verknüpft ist, berechnet sich zu:

$$h(i) = 1 \text{ bit.}$$

* TI-Klausur

  Die Wahrscheinlichkeit für das Bestehen p(ja) der TI-Klausur - Ihre Mitarbeit vorausgesetzt - liegt bei p(ja) = 0.8, Nichtbestehen ist nicht sehr wahrscheinlich und werde mit p(nein) = 0.2 festgelegt; dann folgt:

  $h(ja) = ld\ 1/0.8 = 0.23$ bit und $h(nein) = ld\ 1/0.2 = 2.32$ bit.

#

Die einfachen Beispiele verdeutlichen, daß für gleichwahrscheinliche Ereignisse der Informationsgehalt jeweils den gleichen Wert besitzt und dieser mit zunehmender Wahrscheinlichkeit abnimmt. Ein Vergleich der beiden ja/nein-Modellierungen zeigt insbesondere ein überproportionales Anwachsen des Informationsgehalts für unwahrscheinliche (seltene) Ereignisse. Analog stellt sich die Situation bei der Tagespresse dar: unwahrscheinliche Ereignisse besitzen einen großen Informationsgehalt, Trivialmeldungen sind dagegen wenig informativ.

Im Rahmen des Kanalmodells ist der Informationsgehalt ein Maß dafür, wieviel Information eine diskrete Nachricht enthält, die von einem Sender an einen Empfänger übermittelt wird. Unter Nachricht versteht man hier eine Zeichenfolge, in der die einzelnen Zeichen mit bestimmten Wahrscheinlichkeiten auftreten.

<u>Beispiele:</u>

* Grenzfälle

  - Quelle sendet immer das gleiche Zeichen
    $p = 1 \quad \Rightarrow h = 0$ bit ;

  - ein Zeichen wird nie gesendet (tritt in der Praxis nicht auf)
    $p = 0 \quad \Rightarrow h = \infty$ bit ;

* Quelle sendet Binärzeichen
  $p(0) = ½$, $p(1) = ½ \quad \Rightarrow h(0) = h(1) = 1$ bit;

* Quelle sendet n verschiedene Zeichen mit gleicher Wahrscheinlichkeit
  $p(i) = 1/n$ für $i=1,2,...,n \quad \Rightarrow h(i) = ld\ n$;
  speziell: $n = 2^k \quad \Rightarrow h(i) = k$ bit ;
  Informationsgehalt einer Dezimalziffer: (ld 10)bit = 3.32 bit;
  n-stellige Dezimalzahl: $h = n \cdot 3.32$ bit.

#

Eine wichtige Eigenschaft des Informationsgehalts wird durch die Funktionalgleichung der Logarithmus-Funktion (log (ab) = log a + log b) sichergestellt: Betrachtet man zwei unabhängige Ereignisse A und B, so folgt für den Informationsgehalt, der mit dem Auftreten beider Ereignisse A und B verknüpft ist:

$$h(AB) = -ld\ p(AB),$$

da die beiden Ereignisse unabhängig voneinander sind, gilt:

$$p(AB) = p(A)p(B)$$

und damit

$$h(AB) = -ld\ (p(A)p(B)) = -ld\ p(A) - ld\ p(B) = h(A) + h(B),$$

d.h. der Informationsgehalt für das Auftreten zweier Ereignisse entspricht der Summe der Informationsgehalte für die Einzelereignisse oder besteht eine Nachricht aus einer Folge unabhängiger Zeichen, so ist der Informationsgehalt der Nachricht identisch mit der Summe der Informationsgehalte der einzelnen Zeichen, die die Nachricht aufbauen.

Sendet eine Nachrichtenquelle mit dem Quellalphabet

$$A = \{A_i\ ,\ i=1,...,n\ \}$$

die einzelnen Buchstaben dieses Alphabets mit unterschiedlichen Wahrscheinlichkeiten, so kann der Empfänger den Informationsgehalt nur dann bestimmen, wenn die Auftrittswahrscheinlichkeiten der Buchstaben bekannt sind. In der Praxis ist es wünschenswert, Nachrichtenquellen bezüglich des zu erwartenden Informationsgehalts zu charakterisieren. Dieser Erwartungswert entspricht dem Mittelwert der Informationsgehalte aller Buchstaben des Quellalphabets und wird als *mittlerer Informationsgehalt* oder *Entropie* bezeichnet. Sei $h(A_i)$ der Informationsgehalt und $p(A_i)$ die zugehörige Auftrittswahrscheinlichkeit für das Zeichen A_i, so berechnet sich die Entropie H der Quelle anhand der Formel:

$$H := \sum_{i=1}^{n} p(A_i)\cdot h(A_i) = -\sum_{i=1}^{n} p(A_i)\cdot ld\ p(A_i) \quad ,$$

$$\sum_{i=1}^{n} p(A_i) = 1 \quad .$$

Die Entropie H ist die fundamentale Größe für die Codierung einer Nachrichtenquelle!

***Shannon* zeigte:**

+ **Die Entropie ist das Maß für die mittlere Anzahl der Binär zeichen, die für die Codierung der Quelle benötigt werden.**

+ **Es gibt kein verlustfreies Kompressionsverfahren für die Codierung der Quelle, mit einer mittleren Anzahl von Bits, die kleiner als der Wert der Entropie ist, d.h. die Entropie stellt die untere Grenze für die realisierbare Kompressionsrate dar.**

Beispiele: *Entropie*

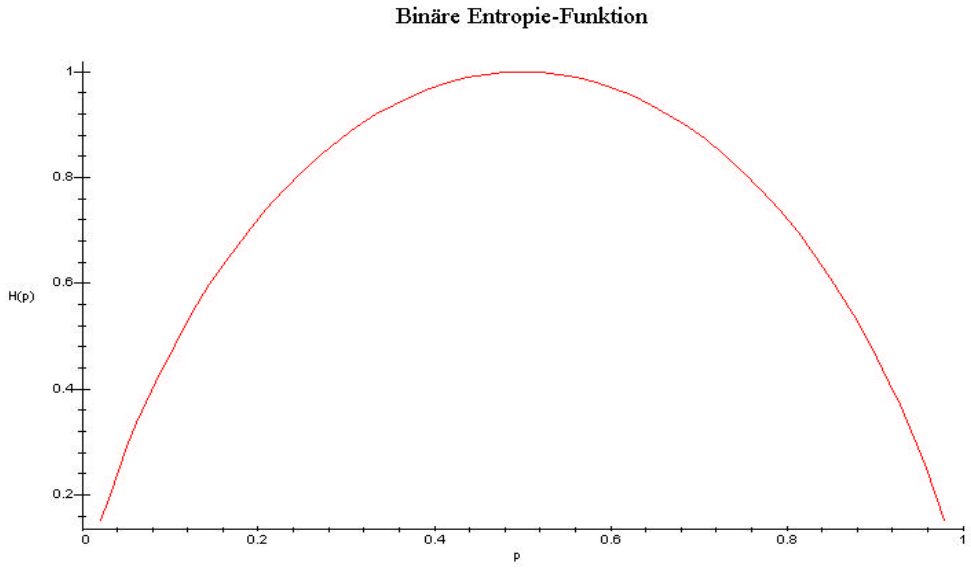
* $H = 0$ falls $p(A_j)=1$ und $p(A_i)=0$ für $i \neq j$;

* H wird maximal für $p(A_i)=1/n$ $(i=1,...,n)$, gleiche Wahrscheinlichkeiten,

$$H = ld\ n\ ;$$

* Quelle sendet zwei Zeichen mit den Wahrscheinlichkeiten p und $q=1-p$ (binäre Entropie-Funktion)

$$H(p) = -p \cdot ld\ p\ - (1-p) \cdot ld\ (1-p) = H(1-p)\ ,$$



* Alphabet A = {x,y,z}

A_i	p_i	h_i
x	0.50	1
y	0.25	2
z	0.25	2

$$H = (0.5 \cdot 1 + 0.25 \cdot 2 + 0.25 \cdot 2)bit = 1.5\ bit$$

#

Eine Codierung für das Alphabet des letzten Beispiels ist:

c(x) = 1 , c(y) = 01 , c(z) = 00;

der Code ist umkehrbar eindeutig, z.B.

011100011 → yxxzyx ,

es liegt ein Präfixcode vor, der die Fano-Bedingung erfüllt.

Die angegebene Codierung ist optimal in dem Sinne, daß die *mittlere Wortlänge* L mit

$$L := \sum_{i=1}^{n} p(A_i) \cdot l(A_i) \quad ,$$

wobei $l(A_i)$ die Länge des dem i-ten Zeichen entsprechenden Codewortes darstellt, in diesem Fall identisch mit dem mittleren Informationsgehalt ist: L = H!

Allgemein definiert man die *Redundanz* R eines Codes als Differenz der mittleren Wortlänge und der Entropie:

$$R := L - H \ .$$

Für alle Binärcodierungen gilt dann:

$$\boldsymbol{R := L - H \geq 0} \ ,$$

d.h. die mittlere Wortlänge eines Binärcodes ist immer größer oder gleich als der mittlere Informationsgehalt!!!

Die Basis für die verlustfreie Kompression bildet ein Schema, das häufig auftretende Zeichen über kurze Binärsequenzen codiert und selten auftretende Zeichen über längere Codeworte darstellt. Die Klassifizierung erfolgt hierbei mit Hilfe der Entropie und mittleren Wortlänge; deshalb bezeichnet man diese Kompressionsverfahren generell als *Entropiecodierungen*.

- Huffman-Codes

Grundsätzlich gibt es für jede Nachrichtenquelle einen optimalen Code variabler Länge (Symbol-Code), der die Präfixbedingung erfüllt.

Huffman-Codes (David A. Huffman, 1951) sind Präfix-Codes und das beste - optimale - Verfahren für die Codierung einzelner Zeichen, denen bestimmte Auftrittswahrscheinlichkeiten zugeordnet sind. Der Huffman-Algorithmus basiert auf den folgenden zwei Eigenschaften zur Charakterisierung optimaler Präfix-Codes:

1. Bei einem optimalen Code besitzen Zeichen, die häufiger auftreten, d.h. eine größere Wahrscheinlichkeit für das Auftreten besitzen, kürzere Codeworte als Zeichen, die weniger häufig auftreten.

2. Bei einem optimalen Code besitzen die beiden am seltensten auftretenden Zeichen, d.h. die beiden Zeichen mit den geringsten Auftrittswahrscheinlichkeiten, Codewörter gleicher Länge.

<u>Konstruktion eines Huffman-Codes</u>

Die Konstruktion eines Huffman-Codes für ein Quellalphabet $A = \{ a_0, a_1, ... a_{n-1} \}$, $|A| = n$, gliedert sich in vier Schritte:

1. Die Wahrscheinlichkeit für das Auftreten des Zeichens a_i sei $p_i = p(a_i)$ und im folgenden sei vorausgesetzt, daß die Zeichen des Alphabets nach fallender Wahrscheinlichkeit angeordnet sind, d.h. $p_0 \geq p_1 \geq p_2 \geq ... \geq p_{n-1}$.

2. Beginnend mit $A_{n-1} := A$ werden nun veränderte Alphabete A_j mit $j=n-2, n-3, ... , 0$ aufgebaut, indem die beiden seltensten Zeichen a_j und a_{j-1} aus A_{j+1} zu einem neuen Zeichen a’ mittels Konkatenation zusammengefaßt werden und $p'=p(a')= p_j + p_{j-1}$ gesetzt wird. Das neue Zeichen a’ wird entsprechend der Wahrscheinlichkeit p’ in die fallende Sequenz $p_0 \geq p_1 \geq ... \geq p_{j-2}$ eingefügt.
   Diese Zusammenfassungen werden parallel als binärer Baum dargestellt; a’, a_j und a_{j-1} bilden die Knoten und die beiden Kanten, die von a_j und a_{j-1} ausgehen enden in a’.

3. Für $j=0$ enthält A_0 nur noch ein einziges Zeichen, nämlich die Konkatenation $a = a_0\ a_1\ a_2 ... a_{n-1}$ aller Buchstaben von A mit $p(a)=1$.
   Hiermit ist der Aufbau des Binärbaumes beendet; es ergibt sich ein Codebaum mit a als Wurzel.

4. Abschließend werden alle Kanten des Codebaums mit 0 für die linken und 1 für die rechten Kanten markiert.
   Durchläuft man den Codebaum von der Wurzel zu den Blättern, so liefert jeweils die Sequenz der Kantenmarkierungen das entsprechende Codewort.

Beispiel: *Huffman-Code*

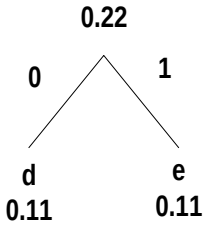
Alphabet A = { a,b,c,d,e }, |A|= 5

A	p
a	0.40
b	0.20
c	0.18
d	0.11
e	0.11

⇒ Entropie: H = 2.14 bit

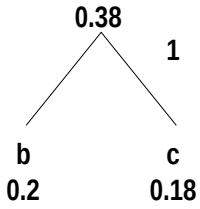
1. Zusammenfassung:

A_3	p
a	0.40
de	0.22
b	0.20
c	0.18



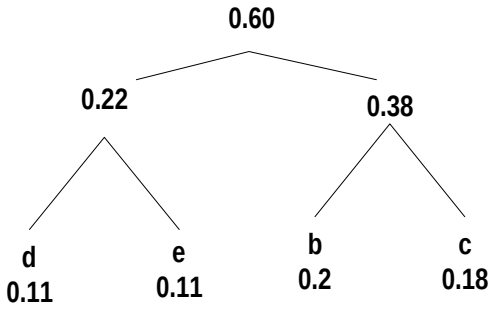
2. Zusammenfassung:

A_2	p
a	0.40
bc	0.38
de	0.22



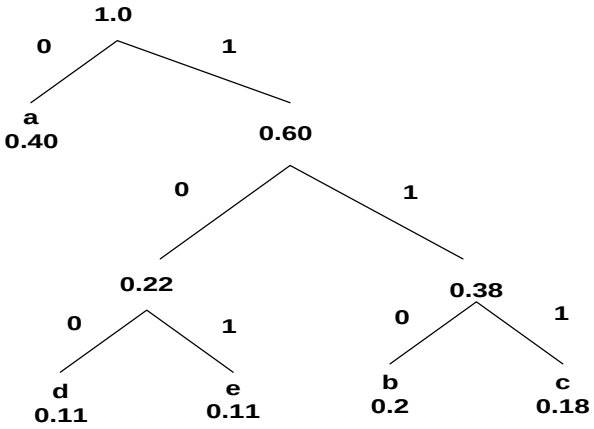
3. Zusammenfassung:

A_1	p
bcde	0.60
a	0.40



4. Zusammenfassung:

A_0	p
abcde	1.00



⇒ Huffman-Code

A	p_i	c_i	l_i	$p_i l_i$
a	0.40	0	1	0.40
b	0.20	110	3	0.60
c	0.18	111	3	0.54
d	0.11	100	3	0.33
e	0.11	101	3	0.33

Die mittlere Wortlänge L ergibt sich zu L = 2.20 bit und die Redundanz beträgt somit R = L - H = 0.06 bit. Die Präfix-Eigenschaft des Huffman-Codes zeigt der zugehörige Codebaum, indem alle Zeichen als Endknoten - Blätter - angeordnet sind.

#

Für die mittlere Wortlänge L von Huffman-Codes, die natürlich vom Umfang des Alphabets A und den Wahrscheinlichkeiten für das Auftreten der einzelnen Zeichen abhängt, läßt sich eine obere und untere Grenze über die Ungleichung

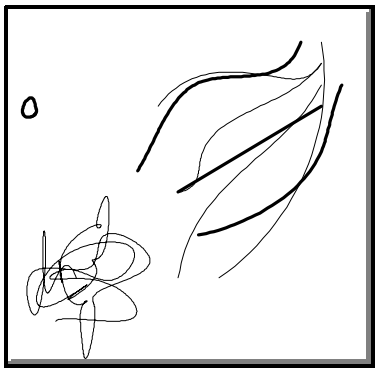
$$H(A) \leq L \leq H(A) + 1 \text{ bit}$$

angeben: L wird nach unten durch die Entropie H(A) begrenzt und nach oben durch die Entropie plus 1 bit.

In der Praxis werden selbstverständlich Erweiterungen des hier vorgestellten Codierungsschemas gewinnbringend eingesetzt. Es bietet sich an, nicht einzelne Zeichen sondern ganze Zeichensequenzen zu codieren, wodurch die Redundanz erheblich vermindert werden kann - vgl. Übung.

Gebräuchlich sind insbesondere auch *adaptive Huffman-Codes*, mit der Eigenschaft, daß Sender und Empfänger die Wahrscheinlichkeiten dynamisch berechnen indem die Häufigkeitsverteilungen der Zeichen in festgelegten Abständen aktualisiert und anschließend für eine der Datensequenz angepasste Codierung eingesetzt werden.

3 Aussagenlogik



3.1 Einführung

° Logik ist die Lehre von der Folgerichtigkeit

° Logik untersucht die Verknüpfung von Aussagen und mögliche Schlußfolgerungen

° Logik abstrahiert und die formale Struktur entscheidet über den Wahrheitsgehalt

° - Aussagenlogik

- Prädikatenlogik (Sprachen der ersten Stufe)

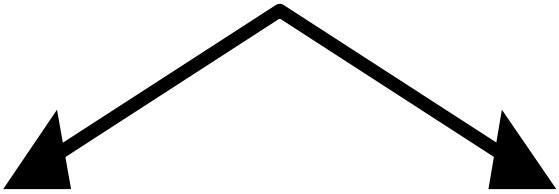
- Fuzzy-Logik

- Modallogik

- . . .

Gegenstand der Logiken sind stets formale Sprachen!

Logik (Sprache)



Syntax **Semantik**

> Regeln > Bedeutung

Syntax

Festlegung der zugelassenen Ausdrücke und formalen Strukturen der Sprache

Semantik

Festlegung der Bedeutung und Interpretation insbesondere auch die Zuordnung von Wahrheitswerten

Aussagenlogik

Aussagen (Sätze):

“ es regnet ” ;
“ die Straße ist naß” ;
“ Köln liegt in den Alpen”;
“ die Eulersche Zahl ist irrational”.

Aussagen können wahr oder falsch sein **!**

Unscharfe Aussagen, die möglicherweise wahr sein können, sind keine Aussagen im Sinne der Aussagenlogik.

Eigenschaften der Aussagenlogik

- Zweiwertigkeitsprinzip

  { zwei Wahrheitswerte: wahr / falsch (w / f) }

- Aussagen werden als Ganzes betrachtet

  { innere Bestandteile (Subjekt, Prädikat, . . .)
  werden nicht untersucht --> vgl. Prädikatenlogik }

- Verknüpfung von Aussagen mittels Junktoren nach formal festgelegten Vorschriften und nicht nach inhaltlichen Aspekten

3.2 Syntax und Semantik

Aussagen dergestalt wie

Aussage a: “ es regnet ” ,

Aussage b: “ die Straße wird naß ”

heißen **atomare Aussagen** oder **atomare Formeln**;

Die Verknüpfung der beiden Aussagen a, b:

Aussage c: “ es regnet und die Straße wird naß ”

beschreibt die **Formel**: $a \wedge b$.

Den Aufbau syntaktisch korrekter Formeln wird durch die Syntax der Aussagenlogik festgelegt.

Def.: *Syntax der Aussagenlogik*
Die Syntax der Aussagenlogik bildet
ein **Alphabet**, bestehend aus der Vereinigung

- der Menge der kleinen lateinischen Buchstaben (gegebenenfalls auch indiziert),

- der Menge der Junktoren { ¬ , ∧ , ∨ , → , ↔ , ⊤ , ⊥ } (Verknüpfungssymbole),

- der Klammermenge { (,) }

und die **Regeln** für die syntaktisch korrekten Formeln:

- kleine lateinische Buchstaben sind atomare Formeln,

- ⊤ und ⊥ sind atomare Formeln,

- alle atomaren Formeln sind Formeln,

- sind A und B Formeln, dann sind auch (A), ¬A, A ∧ B, A ∨ B , A → B und A ↔ B Formeln,

- für die Junktoren gelten die nachfolgenden Prioritäten: ¬ vor ∧ , ∨ vor → vor ↔ .

Bemerkungen:

a) Die Bezeichnung der Junktoren

$\top$ (**verum**, das Wahre) und $\bot$ (**falsum**, das Falsche),
$\neg$ (**nicht**),
$\wedge$ (**und**),
$\vee$ (**oder**),
$\rightarrow$ (**wenn - dann**) und
$\leftrightarrow$ (**genau dann - wenn**)

basiert darauf, daß im allgemeinen diese Zeichen mehrere Aussagen miteinander verbinden; dies ist für $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$ zutreffend, jedoch nicht für $\top$, $\bot$, $\neg$.

Das *verum* $\top$ sowie das *falsum* $\bot$ sind Aussagen, d.h. atomare Formeln: $\top$ steht für die absolut wahre Aussage und $\bot$ steht für die absolut falsche Aussage; beide bezeichnet man als *Aussagekonstanten*.

b) Da $\wedge$ und $\vee$ die gleiche Priorität besitzen werden Klammern verwendet, um die Reihenfolge der Ausführung festzulegen.

c) Die induktive Definition der Formeln ist eine Vorschrift für ein sukzessives Konstruktionsverfahren, das nach und nach alle Objekte erzeugt, die unter den definierten Begriff fallen. Diese Vorgehensweise ist nicht nur für die mathematische Logik sondern insbesondere auch für die Informatik gebräuchlich.

Beispiel:

$$a \wedge b \rightarrow c$$

ist eine syntaktisch korrekte Formel, denn

a, b, $(a \wedge b)$, $(a \wedge b) \rightarrow c$ und schließlich $a \wedge b \rightarrow c$

sind syntaktisch korrekte Formeln;

dagegen verstoßen

$$a \wedge b \vee \neg c, \qquad a \wedge \rightarrow b \vee c, \qquad a(\neg(b \leftrightarrow c))$$

gegen die Regeln der Syntax, bilden mithin keine Formeln!

#

Def.: *Semantik der Aussagenlogik*

Die Elemente der Menge {w , f} heißen *Wahrheitswerte*; w steht für *wahr* und f für den Wahrheitswert *falsch*.

Eine *Belegung* ist eine Funktion A von der Menge der Formeln $\mathcal{F}$ in die Menge der Wahrheitswerte:

$$A: \mathcal{F} \rightarrow \{w , f\} .$$

Die *Interpretation* atomarer Formeln erfolgt über eine Belegung, wodurch diesen Aussagen ein Wahrheitswert zugeordnet wird:

A(a) = w , a wird mit einer wahren Aussage belegt;

A(a) = f , a wird mit einer falschen Aussage belegt .

Die Bedeutung der Junktoren wird wie folgt definiert:

Negation		
Zeichen	Sprechweise	Definition
¬	nicht	a ¬a --- w f f w

Konjunktion		
Zeichen	Sprechweise	Definition
∧	und	a b a ∧ b --- w w w w f f f w f f f f

Disjunktion		
Zeichen	Sprechweise	Definition
$\vee$	oder	a b a $\vee$ b --- w w w w f w f w w f f f

Subjunktion		
Zeichen	Sprechweise	Definition
$\rightarrow$	wenn - dann	a b a $\rightarrow$ b --- w w w w f f f w w f f w

Bijunktion		
Zeichen	Sprechweise	Definition
↔	genau dann-wenn	a b a ↔ b --- w w w w f f f w f f f w

Für eine n-stellige Formel $F(a_1, \ldots, a_n)$ berechnet sich der Wahrheitswert $A(F(a_1, \ldots, a_n))$ für eine gegebene Belegung

$$A(a_1), \ldots, A(a_n) \in \{ w , f \}$$

gemäß:

$$A(F(a_1, \ldots, a_n)) = F(A(a_1), \ldots, A(a_n)).$$

Beispiel:

Für die Formel

$$F(a,b,c) = \neg (\neg b \rightarrow c \wedge a) \wedge (b \leftrightarrow \neg a \vee b)$$

ergibt sich der Wahrheitswert für die Belegung

$$A(a) = w , \quad A(b) = f , \quad A(c) = f$$

folgendermaßen:

$$\begin{aligned} A(F) &= \neg (\neg f \rightarrow f \wedge w) \wedge (f \leftrightarrow \neg w \vee f) \\ &= \neg (w \rightarrow f) \wedge (f \leftrightarrow f \vee f) \\ &= \neg f \wedge (f \leftrightarrow f) \\ &= w \wedge w \\ &= w \end{aligned}$$

#

Die Anzahl der möglichen Belegungen bei einer n-stelligen Formel beträgt 2^n, d.h. in endlich vielen Schritten kann der Wahrheitswerteverlauf einer Formel für alle möglichen Belegungen ermittelt werden.

Praktikabel und bequem sind in diesem Zusammenhang *Wahrheitstafeln*.

Beispiel:

$$F = \neg a \rightarrow (a \rightarrow b)$$

Günstig ist es, wenn für jede in F vorkommende Teilformel eine eigene Spalte angelegt wird.

a	b	¬a	(a → b)	F
w	w	f	w	w
w	f	f	f	w
f	w	w	w	w
f	f	w	w	w

#

Vereinbarung:
Im obigen Beispiel und im weiteren verzichten wir auf die explizite Angabe der Belegungsfunktion, wenn immer dies bequem ist und keine Missverständnisse auftreten können!

Aussageverknüpfungen, die für jede mögliche Belegung den Wahrheitswert w liefern, besitzen im Rahmen der Aussagenlogik eine besondere Bedeutung!

Def.:

Eine Formel F heißt *erfüllbar*, wenn wenigstens eine Belegung existiert mit $A(F) = w$;

eine Formel F ist *allgemeingültig*, eine *Tautologie*, wenn für jede Belegung $A(F) = w$ resultiert;

eine Formel ist *unerfüllbar*, eine *Kontradiktion*, wenn für jede Belegung $A(F) = f$ gilt.

Beispiel:

$$F = (a \rightarrow b) \rightarrow (b \rightarrow a),$$

$$G = (a \rightarrow b) \rightarrow (\neg b \rightarrow \neg a),$$

$$H = (a \rightarrow b) \wedge \neg (a \rightarrow b)$$

Wahrheitswerteverlauf der drei Formeln:

a	b	a→b	b→a	¬(a→b)	¬a	¬b	¬b→¬a	F	G	H
w	w	w	w	f	f	f	w	w	w	f
w	f	f	w	w	f	w	f	w	w	f
f	w	w	f	f	w	f	w	f	w	f
f	f	w	w	f	w	w	w	w	w	f

Somit:

F ist erfüllbar,

G ist allgemeingültig, G ist eine Tautologie,

H ist unerfüllbar, H ist eine Kontradiktion.

Beachte:
Die Tautologie liefert für jede Belegung eine wahre Aussage! #

Bemerkung:

Die Erfüllbarkeit einer Formel kann mit Hilfe einer Wahrheitstafel in endlichen vielen Schritten überprüft werden:

Die Aussagenlogik ist semantisch entscheidbar!

Dieser abbrechende Algorithmus besitzt jedoch ein exponentielles Verhalten in bezug auf die erforderliche Rechenzeit:

eine n-stellige Formel erfordert 2^n unterschiedliche Belegungen; benötigt jeder Test z.B. eine Rechenzeit von 1 µs, so beträgt die Zeit für die gesamte Auswertung

$$2^{100} \text{ µs} \sim 10^{24} \text{ s.}$$

Hinweis:
Derartige Fragestellungen werden im Rahmen der *Komplexitätstheorie* erörtert (-> das Erfüllbarkeitsproblem der Aussagenlogik ist NP-vollständig).

3.3 Äquivalenzen

Im Mittelpunkt der folgenden Überlegungen stehen allgemeingültige Bijunktionen der Form:

$$B \leftrightarrow C.$$

Def.: *Aussagenlogische Äquivalenz*

Besitzen die Formeln $B(a_1, \ldots, a_n)$ und $C(a_1, \ldots, a_n)$ für jede mögliche Belegung $A(a_1), \ldots, A(a_n)$ den gleichen Wahrheitswert

$$A(B) = A(C),$$

so bezeichnet man die allgemeingültige Bijunktion $B \leftrightarrow C$ als *aussagenlogische Äquivalenz* und schreibt:

$$B \Leftrightarrow C,$$

sprechweise: B ist äquivalent zu C.

Beispiel:

$B(a,b) = \neg a \vee b$, $C(a,b) = a \rightarrow b$

Wahrheitstafel für die Bijunktion $B \leftrightarrow C$:

a	b	¬a	¬a ∨ b	a → b	¬a ∨ b ↔ a → b
w	w	f	w	w	w
w	f	f	f	f	w
f	w	w	w	w	w
f	f	w	w	w	w

Es liegt eine allgemeingültige Bijunktion vor; die Äquivalenz

$$\neg a \vee b \quad \Leftrightarrow \quad a \rightarrow b$$

eröffnet die Möglichkeit, $\neg a \vee b$ in allen Formeln durch $a \rightarrow b$ -wie auch umgekehrt- zu ersetzen.

Exemplarisch kann die Formel

$$a \rightarrow (b \rightarrow c)$$

äquivalent wie folgt umgeformt werden:

$$a \rightarrow (b \rightarrow c) \Leftrightarrow a \rightarrow (\neg b \vee c) \Leftrightarrow \neg a \vee (\neg b \vee c).$$

#

Bemerkung:

Das Zeichen ⇔ für die Äquivalenz gehört nicht zur Syntax der Aussagenlogik; die Zeichenkette

$$\neg a \vee b \quad \Leftrightarrow \quad a \rightarrow b$$

ist keine syntaktisch korrekte Formel.
Das Zeichen ⇔ für die Äquivalenz macht eine Aussage *über* die Formel

$$\neg a \vee b \quad \leftrightarrow \quad a \rightarrow b \,,$$

nämlich die, daß diese Bijunktion allgemeingültig ist.

Sprachen, die Aussagen über andere Sprachen machen, heißen *Metasprachen.*

Das Äquivalenzzeichen ⇔ ist bezüglich der Aussagenlogik ein metasprachliches Symbol.

B $\Leftrightarrow$ C beschreibt eine *Äquivalenzrelation* zwischen den Formeln B und C; für diese Relation gelten in der Meatsprache (das ist hier die Algebra) die folgenden definierenden Eigenschaften:

- *Reflexivität*

$$A \Leftrightarrow A$$

- *Symmetrie*

wenn $B \Leftrightarrow A$, dann $A \Leftrightarrow B$

- *Transitivität*

wenn $A \Leftrightarrow B$ und $B \Leftrightarrow C$, dann $A \Leftrightarrow C$.

Beispiel: *Sprachliche Umsetzung*

$$a \rightarrow b \quad \Leftrightarrow \quad \neg a \vee b$$

betrachte

a: die Ampel zeigt rot,
b: das Auto hält an;

$a \rightarrow b$:

wenn die Ampel rot zeigt, dann hält das Auto an;

$\neg a \vee b$:

die Ampel zeigt nicht rot oder das Auto hält an.

Beispiel:
Äquivalenz zwischen wechselseitiger Subjunktion und Bijunktion:

$$a \leftrightarrow b \Leftrightarrow (a \rightarrow b) \wedge (b \rightarrow a).$$

Wahrheitstafel:

a	b	a ↔ b	a → b	b → a	(a → b) ∧ (b → a)
w	w	w	w	w	w
w	f	f	f	w	f
f	w	f	w	f	f
f	f	w	w	w	w

Beide Seiten der Äquivalenz besitzen den gleichen w-f-Verlauf, wodurch die Äquivalenz bestätigt wird.

Diese Äquivalenz erlaubt es, die Bijunktion durch wechselseitige Subjunktionen darzustellen.

Mit

$$a \rightarrow b \quad \Leftrightarrow \quad \neg a \vee b$$

und

$$(a \rightarrow b) \wedge (b \rightarrow a) \quad \Leftrightarrow \quad (\neg a \vee b) \wedge (\neg b \vee a)$$

folgt:

$$a \leftrightarrow b \quad \Leftrightarrow \quad (\neg a \vee b) \wedge (\neg b \vee a),$$

d.h. die Bijunktion läßt sich äquivalent mit den Junktoren $\neg$, $\wedge$ und $\vee$ darstellen.

#

Übersicht einiger wichtiger Äquivalenzen

Die nachfolgend aufgelisteten Äquivalenzen der Aussagenlogik können bequem anhand der entsprechenden Wahrheitstafeln überprüft werden.

Konjunktions-Äquivalenzen

$a \wedge b \Leftrightarrow b \wedge a$	Kommutativität
$a \wedge (b \wedge c) \Leftrightarrow (a \wedge b) \wedge c$	Assoziativität
$a \wedge a \Leftrightarrow a$	Idempotenz
$a \wedge \top \Leftrightarrow a$	Neutralelement
$a \wedge \neg a \Leftrightarrow \bot$	Gesetz vom Widerspruch
$a \wedge (b \vee c) \Leftrightarrow (a \wedge b) \vee (a \wedge c)$	Distributivität
$a \wedge (a \vee c) \Leftrightarrow a$	Absorptionsgesetz
$\neg(a \wedge b) \Leftrightarrow \neg a \vee \neg b$	de Morgan-Gesetz
$a \wedge b \Leftrightarrow \neg(a \rightarrow \neg b)$	$(\wedge, \rightarrow)$ - Äquivalenz

Disjunktions-Äquivalenzen

$a \vee b \Leftrightarrow b \vee a$	Kommutativität
$a \vee (b \vee c) \Leftrightarrow (a \vee b) \vee c$	Assoziativität
$a \vee a \Leftrightarrow a$	Idempotenz
$a \vee \bot \Leftrightarrow a$	Neutralelement
$a \vee \neg a \Leftrightarrow \top$	Tertium non datur
$a \vee (b \wedge c) \Leftrightarrow (a \vee b) \wedge (a \vee c)$	Distributivität
$a \vee (a \wedge c) \Leftrightarrow a$	Absorptionsgesetz
$\neg(a \vee b) \Leftrightarrow \neg a \wedge \neg b$	de Morgan-Gesetz
$a \vee b \Leftrightarrow \neg a \rightarrow b$	$(\vee, \rightarrow)$ - Äquivalenz

Subjunktions-Äquivalenzen

$a \rightarrow b \Leftrightarrow \neg b \rightarrow \neg a$	Kontrapositionsgesetz
$a \rightarrow (b \wedge c) \Leftrightarrow (a \rightarrow b) \wedge (a \rightarrow c)$	Distributivität
$a \rightarrow (b \vee c) \Leftrightarrow (a \rightarrow b) \vee (a \rightarrow c)$	Distributivität
$(a \wedge b) \rightarrow c \Leftrightarrow (a \rightarrow b) \vee (a \rightarrow c)$	R-Distributivität
$(a \vee b) \rightarrow c \Leftrightarrow (a \rightarrow b) \wedge (a \rightarrow c)$	R-Distributivität
$a \rightarrow (b \rightarrow c) \Leftrightarrow b \rightarrow (a \rightarrow c)$	Vertauschen
$a \rightarrow (b \rightarrow c) \Leftrightarrow (a \rightarrow b) \rightarrow c$	Assoziativität
$a \rightarrow a \Leftrightarrow \top$	Reflexivität
$a \rightarrow b \Leftrightarrow \neg a \vee b$	$(\rightarrow,\vee)$-Umwandlung
$a \rightarrow b \Leftrightarrow \neg(a \wedge \neg b)$	$(\rightarrow,\wedge)$-Umwandlung

Bijunktions-Äquivalenzen

$a \leftrightarrow b \Leftrightarrow b \leftrightarrow a$	Kommutativität
$a \leftrightarrow b \Leftrightarrow \neg b \leftrightarrow \neg a$	Kontrapositionsgesetz
$a \leftrightarrow b \Leftrightarrow (a \rightarrow b) \wedge (b \rightarrow a)$	$(\leftrightarrow,\rightarrow)$-Umwandlung
$a \leftrightarrow b \Leftrightarrow (\neg a \vee b) \wedge (a \vee \neg b)$	$(\leftrightarrow,\wedge)$-Umwandlung
$a \leftrightarrow b \Leftrightarrow (a \wedge b) \vee (\neg a \wedge \neg b)$	$(\leftrightarrow,\vee)$-Umwandlung
$a \leftrightarrow (b \leftrightarrow c) \Leftrightarrow (a \leftrightarrow b) \leftrightarrow c$	Assoziativität
$a \leftrightarrow a \Leftrightarrow \top$	Reflexivität

Äquivalenz der doppelten Negation

$$\neg \neg a \Leftrightarrow a$$

Bemerkungen:
Alle Äquivalenzen sind leicht mittels Wahrheitstafeln nachprüfbar;

Beispiel: *Absorptionsgesetz*

$a \wedge (a \vee c) \Leftrightarrow a$

A(a)	A(c)	A(a ∨ c)	A(a ∧ (a ∨ c))
f	f	f	f
f	w	w	f
w	f	w	w
w	w	w	w

#

Alle Äquivalenzen behalten ihre Gültigkeit, wenn die atomaren Formeln konsistent durch beliebige Formeln ersetzt werden.

Das Dualitätsprinzip der Aussagenlogik liefert einen direkten Übergang von den Konjunktions- zu den Disjunktions-Äquivalenzen - allgemein gilt:
sei $F \Leftrightarrow G$ und F^*, G^* werden aus F, G abgeleitet, indem überall $\wedge$ durch $\vee$ und umgekehrt ersetzt wird, so folgt $F^* \Leftrightarrow G^*$.

3.4 Normalformen

Die Äquivalenzen bilden die Basis für die Vereinfachung von Formeln und sind wesentliches Hilfsmittel für die Konstruktion der Normalformen.

Die Normalformen sind direkt mit der Wahrheitstafel korreliert und zeigen andererseits die Allgemeingültigkeit einer Formel in einer besonders einfachen Art auf.

Beispiel:

Betrachten wir eine Formel wie

$$(a \vee \neg a \vee b \vee \neg b) \wedge (c \vee \neg c) \wedge (a \vee b \vee \neg b \vee c),$$

bei der $\rightarrow$ und $\leftrightarrow$ nicht auftreten, sondern in den Klammern nur Disjunktionen und zwischen den Klammern nur Konjunktionen als Junktoren fungieren, so ergibt sich mit:

$$\mathbf{a} \vee \mathbf{\neg a} \Leftrightarrow \top \quad \text{und} \quad \mathbf{a} \vee \top \Leftrightarrow \top$$

aufgrund der Tatsache, dass in jeder Klammer

mindestens eine atomare Aussage negiert und nicht-negiert auftritt:

$$(\top \vee \top) \wedge (\top) \wedge (a \vee \top \vee c) \Leftrightarrow \top \wedge \top \wedge \top \Leftrightarrow \top;$$

d.h. die obige Formel ist eine dreistellige Tautologie: die Formel ist allgemeingültig!

#

Def.: *Konjunktive Normalform* (KNF)

Ist eine Formel allein aus der Konjunktion von Disjunktionen atomarer Formeln bzw. deren Negationen aufgebaut, dann heißt diese Darstellung *Konjunktive Normalform* (KNF).

Satz:

Jede Formel kann über Äquivalenzumformungen als KNF dargestellt werden.

Beweis:
Die KNF ergibt sich, indem
1. alle vorhandenen Bijunktionen mit der Äquivalenz

$$F \leftrightarrow G \Leftrightarrow (\neg F \vee G) \wedge (F \vee \neg G)$$

eliminiert werden - F, G stehen für beliebige Teilformeln;

2. alle vorhandenen Subjunktionen werden mit der Äquivalenz

$$F \rightarrow G \Leftrightarrow \neg F \vee G$$

umgeformt;

3. Negationen vor Klammerausdrücken werden mit den de Morganschen- Äquivalenzen

$$\neg(F \wedge G) \Leftrightarrow \neg F \vee \neg G \quad , \quad \neg(F \vee G) \Leftrightarrow \neg F \wedge \neg G$$

aufgelöst;
4. Teilformeln wie $F \vee (G \wedge H)$ werden mit dem Distributivgesetz $F \vee (G \wedge H) \Leftrightarrow (F \vee G) \wedge (F \vee H)$ aufgelöst.

Beachte:

Die KNF ist syntaktisch nicht eindeutig festgelegt, denn die Reihenfolge der atomaren Formeln und Disjunktionen sowie mögliche Vereinfachungen werden durch den angegebenen Algorithmus für die Umformung in eine KNF nicht fixiert.

Satz:

Eine Formel ist allgemeingültig, wenn alle Disjunktionen der KNF zumindest eine atomare Formel negiert und zugleich nicht-negiert enthalten.

Beweis:

Nach Voraussetzung enthält jede Disjunktion das verum $\top$; mit $\top \vee F \Leftrightarrow \top$ ergibt sich eine Konjunktion wahrer Aussagen und damit die Allgemeingültigkeit.

Beispiel:
Umformung der Formel

$$(a \wedge \neg(b \vee c)) \rightarrow (b \leftrightarrow c)$$

in eine KNF;

* Umwandlung der Bijunktion

$$(a \wedge \neg(b \vee c)) \rightarrow ((b \vee \neg c) \wedge (\neg b \vee c))$$

* Umwandlung der Subjunktion

$$\neg(a \wedge \neg(b \vee c)) \vee ((b \vee \neg c) \wedge (\neg b \vee c))$$

* Umwandlung der äußeren Negation (de Morgan)

$$\neg a \vee \neg\neg(b \vee c) \vee ((b \vee \neg c) \wedge (\neg b \vee c))$$

$\Leftrightarrow$

$$(\neg a \vee b \vee c) \vee ((b \vee \neg c) \wedge (\neg b \vee c))$$

* Umwandlung mit dem Distributivgesetz

$$(\neg a \vee b \vee c \vee b \vee \neg c) \wedge (\neg a \vee b \vee c \vee \neg b \vee c)$$

Die Formel ist allgemeingültig! #

Zusammenhang der KNF mit der Wahrheitstafel

Betrachten wir die Formel

$$A(a,b,c) = (a \leftrightarrow b) \wedge (a \rightarrow b \vee c)$$

und ihre Umwandlung in eine KNF:

$$A(a,b,c) = (a \vee \neg b) \wedge (\neg a \vee b) \wedge (a \rightarrow b \vee c) ,$$

$$A(a,b,c) = (a \vee \neg b) \wedge (\neg a \vee b) \wedge (\neg a \vee b \vee c) .$$

A ist nicht allgemeingültig; eine Belegung mit

$$A(a) = f , \quad A(b) = A(c) = w$$

liefert z.B.

$$A(A) = f \wedge w \wedge w = f .$$

Die Wahrheitswerte $A(A)$ für die möglichen $2^3 = 8$ Belegungen folgen mit der *Expansion* aller disjunktiven Verknüpfungen derart, dass jede Variable (negiert oder nicht-negiert) in den einzelnen Disjunktionen auftritt.

Die Expansion der KNF basiert auf den folgenden Äquivalenzen:

$$F \Leftrightarrow F \vee \bot, \qquad \bot \Leftrightarrow G \wedge \neg G,$$

$$F \Leftrightarrow F \vee (G \wedge \neg G) \Leftrightarrow (F \vee G) \wedge (F \vee \neg G).$$

Anwendung auf die KNF von A liefert:

$$(a \vee \neg b) = (a \vee \neg b \vee \bot) = (a \vee \neg b \vee (c \wedge \neg c)) =$$

$$= (a \vee \neg b \vee c) \wedge (a \vee \neg b \vee \neg c)$$

und

$$(\neg a \vee b) = (\neg a \vee b \vee c) \wedge (\neg a \vee b \vee \neg c)$$

und insgesamt

$$\begin{aligned} A(a,b,c) = {} & (a \vee \neg b \vee c) \wedge (a \vee \neg b \vee \neg c) \wedge \\ & (\neg a \vee b \vee c) \wedge (\neg a \vee b \vee \neg c) \wedge \\ & (\neg a \vee b \vee c); \end{aligned}$$

aufgrund der Idempotenz ($F \wedge F \Leftrightarrow F$) ergibt sich abschließend die *kanonische* KNF:

$$\begin{aligned} A(a,b,c) = {} & (a \vee \neg b \vee c) \wedge (a \vee \neg b \vee \neg c) \wedge \\ & (\neg a \vee b \vee c) \wedge (\neg a \vee b \vee \neg c). \end{aligned}$$

Die *kanonische* KNF ist nur aus *vollständigen* Disjunktionen aller Variablen (atomaren Formeln) aufgebaut;

vollständige Disjunktionen werden als *Maxterme* bezeichnet und beinhalten jede Variable (negiert oder nicht-negiert) genau einmal.

Die Struktur der kanonischen KNF zeigt direkt, für welche Belegungen die Formel eine *falsche* Aussage darstellt.

Wegen $F \wedge \bot \Leftrightarrow \bot$ ergibt sich $A(A) = f$ für jede Belegung, die einen der Maxterme zu einer falschen Aussage verknüpft.

Für

$$A(a,b,c) = (a \vee \neg b \vee c) \wedge (a \vee \neg b \vee \neg c) \wedge (\neg a \vee b \vee c) \wedge (\neg a \vee b \vee \neg c)$$

bedeutet dies

$$A(A) = f$$

für

$$A(A) = A(f,w,f) = f \wedge w \wedge w \wedge w = f ,$$

$$A(A) = A(f,w,w) = w \wedge f \wedge w \wedge w = f ,$$

$$A(A) = A(w,f,f) = w \wedge w \wedge f \wedge w = f ,$$

$$A(A) = A(w,f,w) = w \wedge w \wedge w \wedge f = f .$$

Die verbleibenden vier Belegungen

$$(w , w , w) , (w , w , f) , (f , f , w) , (f , f , f)$$

ergeben zwangsläufig

$$A(A) = w .$$

Die kanonische KNF erlaubt es, die Wahrheitstafel direkt anzugeben:

a b c	A (a , b , c)	Bezeichnung
w w w	w	*Minterm*
w w f	w	*Minterm*
w f w	f	*Maxterm*
w f f	f	*Maxterm*
f w w	f	*Maxterm*
f w f	f	*Maxterm*
f f w	w	*Minterm*
f f f	w	*Minterm*

Umgekehrt läßt sich anhand der Wahrheitstafel die kanonische KNF leicht aufstellen, indem jeder f-Wert von A mit dem entsprechenden Maxterm der kanonischen KNF berücksichtigt wird.

Analog zur kanonischen KNF erhält man eine *duale* Formulierung in Form der *kanonischen disjunktiven Normalform* (DNF), wobei den w-Werten der Wahrheitstafel *vollständige Disjunktionen* (*Minterme*) zugeordnet und diese disjunktiv miteinander verknüpft werden.

Kanonische DNF

a b c	A (a , b , c)	Bezeichnung
w w w	w	*Minterm*
w w f	w	*Minterm*
w f w	f	*Maxterm*
w f f	f	*Maxterm*
f w w	f	*Maxterm*
f w f	f	*Maxterm*
f f w	w	*Minterm*
f f f	w	*Minterm*

Disjunktive Verknüpfung der Minterme:

$$A(a,b,c) = (a \wedge b \wedge c) \vee (a \wedge b \wedge \neg c) \vee$$

$$(\neg a \wedge \neg b \wedge c) \vee (\neg a \wedge \neg b \wedge \neg c) ;$$

die Formel A ist wahr, wenn einer der Minterme den Wert w annimmt ($w \vee f = w$):

$$A(A) = A(w,w,w) = A(w,w,f)$$

$$= A(f,f,w) = A(f,f,f) = w.$$

3.5 Inferenzen, Schlußregeln

Inferenzen (logische Folgerungen) und bestimmte Schlußregeln sind von grundlegender Bedeutung für die gesamte Logik.

Beispiel: *wenn-dann-Inferenzen*

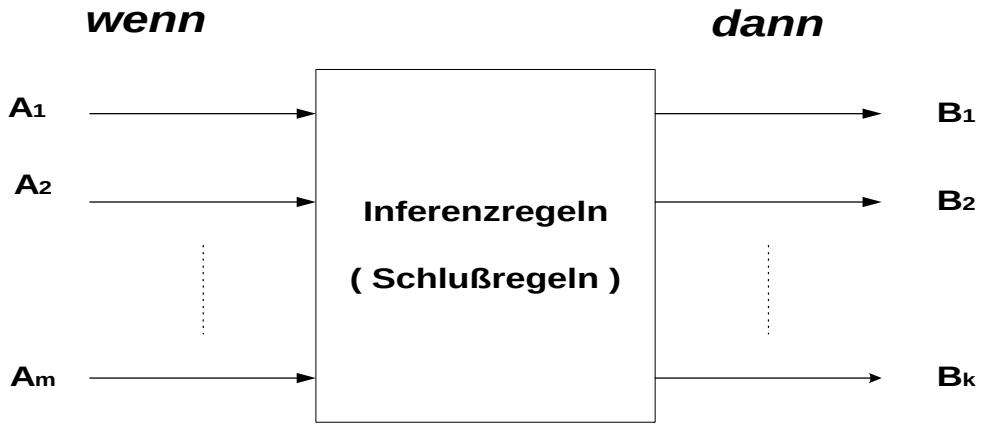
Wenn es regnet, *dann* wird die Straße naß.

Wenn das Wetter schön ist, *dann* gehen wir spazieren.

Wenn eine Zahl nur durch Eins *und* sich selber teilbar ist, *dann* ist diese Zahl eine Primzahl.

#

Allgemeine Struktur des logischen Schlusses:



Def.: *Aussagenlogische Folgerung*

Seien $A_1, \ldots, A_m$ und $B_1, \ldots, B_k$ n-stellige Formeln; dann heißen die $B_1, \ldots, B_k$ aussagenlogische Folgerungen (Konklusionen) aus den Prämissen $A_1, \ldots, A_m$, wenn für jede Belegung $A(x_1), \ldots, A(x_n)$ mit

$$A(A_1) = \ldots = A(A_m) = w$$

auch

$$A(B_1) = \ldots = A(B_k) = w$$

gilt. Abkürzend schreibt man:

$$A_1, \ldots, A_m \qquad B_1, \ldots, B_k$$

oder

$$A_1 \wedge \ldots \wedge A_m \qquad B_j \quad (j = 1, \ldots, k).$$

Beachte:
Wahre Prämissen bedingen wahre Konklusionen!

Seien $A_1, \ldots, A_m$ n-stellige Formeln, deren Konjunktion $A_1 \wedge \ldots \wedge A_m$ konsistent, d.h. erfüllbar ist.

Dann besteht die Menge $\{ B_1, \ldots, B_k \}$ aller Folgerungen aus

der n-stelligen Tautologie,

allen Maxtermen der kanonischen KNF von $A_1 \wedge \ldots \wedge A_m$

und allen Konjunktionen von mindestens zweien dieser Maxterme.

Ist μ die Anzahl der Maxterme, dann gibt es insgesamt

$$k = 2^{\mu}$$

Folgerungen.

Beispiel:

Prämissen: $A_1(a,b) = a \wedge b$, $A_2(a,b) = a \rightarrow b$

Wahrheitstafel:

a	b	A_1	A_2	$A_1 \wedge A_2$	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8
w	w	w	w	w	w	w	w	w	w	w	w	w
w	f	f	f	f	w	w	w	f	f	f	w	f
f	w	f	w	f	w	w	f	w	f	w	f	f
f	f	f	w	f	w	f	w	w	w	f	f	f

$B_1(a,b) = (a \wedge b) \vee (a \wedge \neg b) \vee (\neg a \wedge b) \vee (\neg a \wedge \neg b)$

(kanonische DNF der 2-stelligen Tautologie)

$B_2(a,b) = a \vee b$

$B_3(a,b) = a \vee \neg b$

$B_4(a,b) = \neg a \vee b = a \rightarrow b = A_2(a,b)$

$B_5(a,b) = (\neg a \vee b) \wedge (a \vee \neg b) = a \leftrightarrow b$

$B_6(a,b) = (\neg a \vee b) \wedge (a \vee b) = b$

$B_7(a,b) = (a \vee \neg b) \wedge (a \vee b) = a$

$B_8(a,b) = (a \vee \neg b) \wedge (\neg a \vee b) \wedge (a \vee b) = a \wedge b = A_1(a,b)$

Seien $A_1, \ldots, A_{m,}$ B n-stellige Formeln und B eine aussagenlogische Folgerung aus den $A_1, \ldots, A_m$.
Dann ist die *aussagenlogische Implikation*

$$A_1, \ldots, A_m \qquad B$$

oder

$$A_1 \wedge \ldots \wedge A_m \qquad B$$

äquivalent mit der Aussage

$A_1 \wedge \ldots \wedge A_m \rightarrow$ B ist allgemeingültig.

Als *Schlußregeln* bezeichnet man bestimmte aussagenlogische Schlüsse, die für zwei Prämissen als *Figur* allgemein wie folgt dargestellt werden:

A_1
A_2

B

und für A_1, A_2 B steht,
d.h. $A_1 \wedge A_2 \rightarrow$ B ist allgemeingültig.

Abtrennungsregel

Als Abtrennungsregel (modus ponens) bezeichnet man die Schlußfigur:

a → b	*wenn a, dann b*
a	*nun aber a*
---	---
b	*also b*

.

<u>Beispiel:</u>

Wenn es regnet, dann wird die Straße naß
Es regnet

Die Straße wird naß

#

Widerlegungsregel

Als Widerlegungsregel (modus tollens) bezeichnet man die Schlußfigur:

a → b	*wenn a, dann b*
¬b	*nun aber nicht b*
---	---
¬a	*also nicht a* .

Beispiel:

Wenn es regnet, dann wird die Straße naß
Die Straße ist nicht naß

Es regnet nicht

#

Kettenschluß

Als Kettenschluß (modus barbara) bezeichnet man die Schlußfigur:

a → b	*wenn a, dann b*
b → c	*wenn b, dann c*
---	---
a → c	*also: wenn a, dann c* .

<u>Beispiel:</u>

Wenn es regnet, dann wird die Straße naß
Wenn die Straße naß ist, verlängert sich der Bremsweg

Wenn es regnet, dann verlängert sich der Bremsweg

#

Prädikatenlogik

Obwohl die Aussagenlogik als Grundlage der Booleschen Algebra eine wichtige Rolle einnimmt, ist sie nicht mächtig genug, um viele Eigenschaften von Gegenständen der Wirklichkeit ausreichend darzustellen. Diese Beschränkung läßt sich an folgendem Beispiel veranschaulichen:

Beispiel 1.4 *Querbezüge zwischen Aussagen*
Fifi ist ein Hund; Hunde mögen Knochen, also mag Fifi Knochen. □

Wir wollen nun versuchen, diesen Satz auf der Grundlage der Aussagenlogik zu formalisieren. Dazu wählen wir drei Aussagen, nämlich:

A für die Aussage Fifi ist ein Hund
B für die Aussage Hunde mögen Knochen
C für die Aussage Fifi mag Knochen

Diese Aussagen verknüpfen wir nun mittels Konjunktion und Implikation zur Formel

$$A \wedge B \Rightarrow C$$

Dabei müssen wir jedoch feststellen, daß wir die inhaltlichen Bezüge zwischen den Aussagen des Beispielsatzes, insbesondere den Zusammenhang zwischen dem Begriff „Hund`` und den Eigenschaften von „Fifi``, nicht nachbilden können.

Dies wird erst im Rahmen der **Prädikatenlogik** (engl.: *predicate logic*), die wir in diesem Abschnitt einführen werden, möglich. Die Prädikatenlogik ist eine Erweiterung der Aussagenlogik, die uns zusätzliche Begriffe und Strukturierungsmittel verschafft, um Argumente wie im obigen Beispiel zu formalisieren.

Merkmale der Prädikatenlogik

Bei der Beschreibung der Aussagenlogik sind wir Variablen begegnet, welche stets als Platzhalter für eine der beiden logischen Konstanten `w` oder `t` standen. Ein besonderes Merkmal der Prädikatenlogik ist nun die Verwendung von **Variablen** in einem erweiterten Sinne, nämlich als besonders gekennzeichnete Symbole, die beliebige Objekte eines vorher abgegrenzten Bereichs bezeichnen. Ein weiteres Merkmal ist die Verwendung besonderer Symbole zur sinntragenden Bezeichnung von Objekteigenschaften oder von Beziehungen zwischen mehreren Objekten. Diese Symbole werden **Prädikate** genannt.

Beispiel 1.5 *Prädikat*
Die Eigenschaft, daß ein Tier ein Hund ist, würden wir zum Beispiel durch das Prädikat

$$hund(x)$$

ausdrücken. Dieses Prädikat gilt für alle denkbaren Hunde x, nicht aber für anderen Tiere. □

Einem Prädikat selbst kann kein Wahrheitswert zugewiesen werden. Es drückt nur Eigenschaften oder Beziehungen zwischen Objekten aus. Prädikate können jedoch zu Aussagen werden, wenn man ihre Variablen durch geeignete Individuen aus einem Interessensbereich - hier Namen - ersetzt. Dabei nimmt jedes Vorkommen einer Variable den selben Wert an. Die Variable *x* in unserem vorigen Beispiel stellt demnach einen Platzhalter für viele konkrete Namen dar. Solche Variablen bezeichnen wir als **freie Variable**.

Beispiel 1.6 *Ersetzung freier Variablen*
Die Aussage $hund(Fifi)$ ist zum Beispiel wahr, falls *Fifi* ein Hund ist. Sie ist aber falsch, falls *Fifi* ein Kaninchen oder ein anderes Lebewesen benennt, das man nicht als Hund bezeichnen würde. □

Wir nennen das Prädikat *hund*(*x*) **einstellig**, weil es genau ein Argument annimmt. Häufig wollen wir jedoch auch Beziehungen zwischen Gegenständen oder Personen durch Prädikate charakterisieren. Dies ist möglich durch Verwendung ***n*-stelliger Prädikate** der Form

$$p(x_1, \ldots, x_n).$$

Beispiel 1.7 *Zweistellige Prädikate*
Das zweistellige Prädikat

ehefrau_von(x, y)

soll ausdrücken, daß x die Ehefrau von y ist. Ein Beispiel für eine aus dem Prädikat durch Variablenersetzung abgeleitete wahre Aussage ist

ehefrau_von(Clara Schumann, Robert Schumann),

während die Aussage

ehefrau_von(Cleopatra, Napoléon)

nach unserer historischen Erkenntnis falsch sein muß.

Zu den zweistelligen Prädikaten, die uns seit Schulzeiten geläufig sind, gehören:

Bezeichnung	Schreibweise
numerisch gleich	$a = b$
numerisch kleiner	$a < b$
numerisch größer	$a > b$
numerisch kleiner gleich	$a \leq b$
numerisch größer gleich	$a \geq b$

□

Prädikate benutzen wir in Spezifikationen, um Eigenschaften des entstehenden Programmsystems festzulegen. Prädikate dokumentieren informelle Anforderungen oder durch Beobachtung gewonnene Einsichten in Systemeigenschaften und geben uns damit die Möglichkeit nachzuweisen, daß Eigenschaften von Entwürfen und Implementierungen eine Folge ausdrücklich formulierter Anforderungen sind.

In den Argumenten von Prädikaten wollen wir aber nicht nur Variablen verwenden, sondern auch Ausdrücke, die bestimmte Werte aus einem Interessensbereich darstellen.

Definition 1.2 *Ausdrücke werden ausschließlich gemäß folgender Regeln gebildet:*

1. *Jede Konstante und jede Variable ist ein Ausdruck.*
2. *Wenn $t_1, \ldots, t_m$ Ausdrücke sind und f ein m-stelliges Funktionssymbol darstellt, dann ist auch $f(t_1, \ldots, t_m)$ ein Ausdruck.*
3. *Wenn t_1, t_2 Ausdrücke sind und op einen binären Operator darstellt, dann ist auch $(t_1\ op\ t_2)$ ein Ausdruck.*

Unter dem Begriff **Konstante** verstehen wir Zeichen mit festgelegter Bedeutung wie etwa die Konstante „0`` aus dem Bereich der ganzen Zahlen. Auch **Funktionssymbole** und Operatorzeichen wie etwa „+`` oder „/`` haben eine feste Bedeutung. Sie bezeichnen aber keine konkreten Werte, sondern Operationen auf Objekten eines Interessensbereichs.

Schreibweise. Wir werden im folgenden Variablen in Ausdrücken vorzugsweise durch die Zeichen x, y, z sowie ihre indizierten Versionen und Konstanten durch die Zeichen a, b, c sowie ihre indizierten Versionen darstellen. Als Funktionssymbole wählen wir üblicherweise die Zeichen f, g und h. ∘

Beispiel 1.8 *Ausdrücke aus dem Bereich* $\mathbf{Z}$ *der ganzen Zahlen*

Bezeichnung	Schreibweise
Konstanten	$\ldots, -2, -1, 0, 1, 2, \ldots$
Summe	$t_1 + t_2$
Differenz	$t_1 - t_2$
Produkt	$t_1 * t_2$
Quotient	t_1 / t_2

□

Ein drittes Merkmal der Prädikatenlogik ist die Möglichkeit, Variablen durch die beiden prädikatenlogischen **Quantoren** zu binden. Wir unterscheiden zwischen dem **Allquantor** $\forall$ und dem **Existenzquantor** $\exists$.

Beispiel 1.9 *Quantoren*
Quantoren erlauben es uns, Sätze der Art:

Es gibt eine ganze Zahl x, so daß x durch 5 teilbar ist.
Für alle ganzen Zahlen y gilt: y ist durch 1 teilbar.

wie folgt zu formalisieren:

$$\exists x \bullet teilbar\_durch\ (5, x) \tag{1.2}$$

$$\forall y \bullet teilbar\_durch(1, y) \tag{1.3}$$

□

Die Variablen *x* und *y* in (1.2) beziehungsweise (1.3) heißen **gebundene Variablen**, denn sie sind durch die Quantoren in der durch das Zeichen „ • `` abgetrennten Teilformel gebunden und können darin nicht mehr beliebig ersetzt werden. (1.2) und (1.3) selbst sind auch keine Prädikate sondern Aussagen.

Eine Aussage der Form $\exists x \bullet P(x)$ stellt nämlich sicher, daß es im Interessensbereich, über den die Variable *x* rangiert, wenigstens einen Wert gibt, für den *P* wahr ist. Mit dieser Interpretation können wir informelle Aussagen wie Es gibt wenigstens einen Hund

durch

$$\exists x \bullet hund(x)$$

und die Formulierung Napoléon hat eine Ehefrau

durch

$$\exists x \bullet ehefrau\_von(x, \mathrm{Napoléon})$$

ausdrücken. Die letzte Aussage bedeutet exakt ausgedrückt: „Napoléon hat wenigstens eine Ehefrau``. Um den Fall auszuschließen, daß es mehrere Ersetzungen für *x* geben kann, bei der die resultierende Aussage wahr wird, verwendet man zuweilen den Quantor $\exists!(x)$ oder $\exists_1(x)$ (in der Sprache Z ) mit der Bedeutung: „Es gibt genau ein *x*, so daß ... gilt``.

Eine Schwierigkeit tritt bei der Interpretation von Sätzen dadurch auf, daß nicht alle Ersetzungen von Variablen in Prädikaten sinnvoll sind, wie etwa im Beispiel $ehefrau\_von(Fifi, 3)$. Um diesem Problem zu entgehen, verlangen wir von dem bereits mehrfach verwendeten Begriff des **Interessensbereichs**, daß er genau die Objekte umfaßt, für die ein gegebenes Prädikat einen Sinn ergibt.

Eine Aussage der Form $\forall x \bullet P(x)$ besagt, daß jedes Objekt *x* aus dem Interessensbereich die Eigenschaft *P* besitzt.

Prädikatenlogische Formeln

Zusammenfassend können wir nun definieren, wie die Formeln der Prädikatenlogik gebildet werden.

Definition 1.3 *Jede Formel der Prädikatenlogik ist allein nach folgenden Regeln aufgebaut:*

1. *Die Wahrheitswerte* **w** *und* **f** *sind Formeln.*
2. *Sind* $t_1, \ldots, t_n$ *Ausdrücke und* P *ein* n*-stelliges Prädikat, so ist auch* $P(t_1, \ldots, t_n)$ *eine Formel.*
3. *Falls* P *eine Formel ist, so ist auch* $\neg P$ *eine Formel.*
4. *Falls* P *und* Q *Formeln sind, so sind auch* $P \wedge Q, P \vee Q, P \Rightarrow Q$ *und* $P \Leftrightarrow Q$ *Formeln.*
5. $\forall x \bullet P$ *ist eine Formel, falls* x *eine beliebige Variable ist und* P *einen Formel darstellt; wenn* x *zuvor in* P *frei war, ist es jetzt gebunden.*
6. $\exists x \bullet P$ *ist eine Formel, falls* x *eine beliebige Variable ist und* P *einen Formel darstellt: wenn* x *zuvor in* P *frei war, ist es jetzt gebunden.*

Wir können an dieser Definition erkennen, daß jede aussagenlogische Formel auch eine Formel der Prädikatenlogik ist.

Das Wesen freier und gebundener Variablen wird bei der Untersuchung der Bedeutung einer quantifizierten Formel der Form $\forall x \bullet P$ deutlich: Ist *x* die einzige Variable, die in *P* vorkommt, so ist $\forall x \bullet P$ genau dann eine wahre Aussage (im Sinne der Aussagenlogik), wenn *P* für alle Ersetzungen von *x* den Wert wahr annimmt. Enthält *P* noch weitere (also freie) Variable, so repräsentiert $\forall x \bullet P$ ein Prädikat über eben diese freien Variablen, nicht jedoch über *x*, da *x* kein Platzhalter sondern eine gebundene Variable ist und somit nicht ersetzt werden kann. Dieses Prädikat ist für bestimmte konstante Ersetzungen der freien Variablen in *P* genau dann wahr, wenn *P* für alle möglichen Ersetzungen von *x* und unter den für die freien Variablen gewählten Ersetzungen wahr ist. Sonst hat $\forall x \bullet P$ der Wert f. Für Formeln der Form $\exists x \bullet P$ gelten analoge Überlegungen.

Beispiel 1.10 *Freie und gebundene Variablen*
Der Ausdruck:

$$\exists y : \mathbb{Z} \bullet x = y^2$$

ist keine Aussage, weil von der Wahl der Ersetzung für die freie Variable x abhängt, ob der Formel wahr oder falsch ist. Das Symbol $\mathbb{Z}$ bezeichnet hier die Menge der ganzen Zahlen.

In dem Formel

$$\exists x : \mathbb{Z} \bullet x = x^2$$

kommt dagegen keine freie Variable vor. Der Formel repräsentiert einfach eine wahre Aussage. □

Konstruktion prädikatenlogischer Formeln

Abschließend wollen wir auf unser Anfangsbeispiel zurückkommen. Mit den in diesem Abschnitt eingeführten Konzepten läßt sich die Aussage aus Beispiel 1.4 im Abschnitt Prädikatenlogik in folgender prädikatenlogischen Formel fassen:

$$\forall x \bullet (hund(x) \Rightarrow mag(x, Knochen)).$$

Die Formel *Es gibt Hunde, die Knochen mögen*.

können wir in folgende Aussage umsetzen:

$$\exists x \bullet (hund(x) \wedge mag(x, Knochen))$$

Selbsttestaufgabe 1.6 *Prädikatenlogische Formeln*
Setzen Sie folgende Formulierungen in prädikatenlogische Formeln um:

1. Falls x ein Hund ist und y ein eßbarer Gegenstand ist, dann wird y von x gemocht.
2. Es gibt zumindest einen Hund x, der jeden eßbaren Gegenstand y mag.
3. Für jeden eßbaren Gegenstand y gibt es einen Hund x, der y mag.

Merke:

Die in der Umgangssprache eher übliche Formulierung:
„Es gibt einen Hund, der jeden eßbaren Gegenstand mag"
ist mehrdeutig, weil er sowohl wie Formel 2 als auch wie Formel 3 oben verstanden werden kann.

◇

Gesetze der Prädikatenlogik

Die aus der Aussagenlogik bekannte Begriffe wie semantische Äquivalenz, Tautologie, Widerspruch, Erfüllbarkeit und der Schlußbegriff lassen sich direkt auf die Prädikatenlogik übertragen. Der einzige Unterschied besteht dabei in einer Verallgemeinerung des Begriffes „Belegung``: Variablen werden nun nicht mehr durch boolesche Werte, sondern durch Elemente ihres jeweiligen Interessensbereichs ersetzt. Die entsprechenden Definitionen können ansonsten wortwörtlich übernommen werden, sodaß wir hier auf eine Wiederholung verzichten.

Ebenso sind auch auch alle Theoreme der Aussagenlogik Theoreme der Prädikatenlogik. Inbesondere sind also auch die in Tabelle 1.2 aufgeführten Äquivalenzen in der Prädikatenlogik gültig. In Tabelle 1.3 sind einige wichtige weitere Äquivalenzen der Prädikatenlogik aufgelistet.

1.	$\forall x \bullet P(x)$	$\equiv$	$\neg\ \exists x \bullet \neg\ P(x)$
2.	$\forall x \bullet \neg\ P(x)$	$\equiv$	$\neg\ \exists x \bullet P(x)$
3.	$\neg\ \forall x \bullet P(x)$	$\equiv$	$\exists x \bullet \neg\ P(x)$
4.	$\neg\ \forall x \bullet \neg\ P(x)$	$\equiv$	$\exists x \bullet P(x)$

Tabelle 1.3: Äquivalenzen der Prädikatenlogik

Die Äquivalenzen drücken aus, daß e Aussage, daß ein Prädikat *P* für alle Objekte eines Interessensbereichs gilt, gleichbedeutend ist mit der Aussage, daß es kein Objekt im Interessensbereich gibt, für das *P* nicht gilt,

1. die Aussage, daß ein Prädikat *P* für alle Objekte eines Interessensbereichs nicht gilt, gleichbedeutend ist mit der Aussage, daß es kein Objekt im Interessensbereich gibt, für das *P* gilt,

2. die Aussage, daß ein Prädikat *P* nicht für alle Objekte eines Interessensbereichs gilt, gleichbedeutend ist mit der Aussage, daß es wenigstens ein Objekt im Interessensbereich gibt, für das *P* nicht gilt, und daß

3. die Aussage, daß ein Prädikat *P* nicht für alle Objekte eines Interessensbereichs nicht gilt, gleichbedeutend ist mit der Aussage, daß es wenigstens ein Objekt im Interessensbereich gibt, für das *P* gilt.

Diese vier Regeln sind Verallgemeinerungen der de Morganschen Gesetze für beliebe Objekte aus möglicherweise unendlichen Interessensbereichen. Weitere wichtige Regeln der Prädikatenlogik sind in Tabelle 1.4 angegeben. Sie gelten unter der Annahme, daß die Variablen *x* und *y* aus demselben Interessensbereich stammen.

$$\forall x \bullet P(x) \Rightarrow P(y)$$
$$P(x) \Rightarrow \exists y \bullet P(y)$$
$$\forall x \bullet P(x) \Rightarrow \exists x \bullet P(x) \text{ falls Interessensbereich} \neq \emptyset$$
$$\exists x \bullet (\exists y \bullet Q(x,y)) \equiv \exists y \bullet (\exists x \bullet Q(x,y))$$
$$\forall x \bullet (\forall y \bullet Q(x,y)) \equiv \forall y \bullet (\forall x \bullet Q(x,y))$$
$$(\exists x \bullet (\forall y \bullet Q(x,y))) \Rightarrow (\forall y \bullet (\exists x \bullet Q(x,y)))$$
$$\forall x \bullet (P(x) \Rightarrow R(x)) \Rightarrow (\forall x \bullet P(x) \Rightarrow \forall x \bullet R(x))$$
$$\forall x \bullet (P(x) \Rightarrow R(x)) \Rightarrow (\exists x \bullet P(x) \Rightarrow \exists x \bullet R(x))$$

Tabelle 1.4: Gesetze der Prädikatenlogik

Auch die Schlußregeln der Aussagenlogik und die Theoreme 1.1 und 1.2 übertragen sich auf die Prädikatenlogik. Als Beispiel für eine neue Schlußregel der Prädikatenlogik betrachten wir die $^{\forall}$-Entfernung . Sie ist genau die Schlußregel, welche der ersten Implikation aus Tabelle 1.4 entspricht:

$$\frac{\forall x \bullet P(x)}{P(y)}$$

Diese Regel erlaubt es uns also, von der Gültigkeit einer allquantifizierten Aussage auf die Gültigkeit der Aussage im Spezialfall zu schließen. Weitere Schlußregeln der Prädikatenlogik finden sich im Anhang.

Anhang

Das folgende System enthält Schlußregeln, mit denen wir die Operatoren der Aussagenlogik in Formeln einführen und wieder entfernen können. Für jeden Operator gibt es also jeweils zwei Arten von Regeln:

- Einführungsregeln, auch Introduktionsregeln genannt, sowie
- Enfernungsregeln, auch Eliminationsregeln genannt.

Erstere werden mit dem Anhängsel - **I** bezeichnet, letztere mit dem Anhängsel - **E**.

Es handelt sich bei dem System um einen aus Darstellungsgründen leicht abgeänderten Auszug aus dem Systems des natürlichen Schließens von Gentzen, siehe z.B. auch Literaturangabe [Bib92]. Für Schlußregeln verwenden wir dabei die folgende Form:

$$\frac{Prämisse_1,\ Prämisse_2,\ \ldots,\ Prämisse_n}{Schlußfolgerung}$$

Die oberhalb der waagerechten Linie angegebenen Prämissen bestimmen die Voraussetzungen zur Anwendung einer Regel. Die Schlußfolgerung unterhalb dieser Linie beschreibt das Ergebnis der Regelanwendung.

Ex falso quodlibet	$\frac{\mathbf{f}}{D}$		
$\wedge$ **-I**	$\frac{A,\ B}{A \wedge B}$	$\wedge$ **-E**	$\frac{A \wedge B}{A}$ $\quad$ $\frac{A \wedge B}{B}$
$\vee$ **-I**	$\frac{A}{A \vee B}$ $\quad$ $\frac{B}{A \vee B}$	$\vee$ **-E**	$\frac{A \vee B,\ A \Rightarrow C,\ B \Rightarrow C}{C}$
$\neg$ **-I**	$\frac{A \Rightarrow \mathbf{f}}{\neg A}$	$\neg$ **-E**	$\frac{A,\ \neg A}{\mathbf{f}}$
$\Rightarrow$ **-I**	$\frac{\begin{array}{c}[A]\\ \vdots \\ B\end{array}}{A \Rightarrow B}$	$\Rightarrow$ **-E**	$\frac{A,\ A \Rightarrow B}{B}$
$\Leftrightarrow$ **-I**	$\frac{A \Rightarrow B,\ B \Rightarrow A}{A \Leftrightarrow B}$	$\Leftrightarrow$ **-E**	$\frac{A \Leftrightarrow B}{A \Rightarrow B}$ $\quad$ $\frac{A \Leftrightarrow B}{B \Rightarrow A}$

Tabelle A.1: Aussagenlogisches Schlußsystem

Bis auf die $\Rightarrow$-Einführung sollten diese Regeln selbsterklärlich sein. Letztere ist wie folgt zu lesen:

Wenn B sich unter der hypothetischen Annahme von A schlußfolgern läßt, dann gilt $A \Rightarrow B$

Die Rechtfertigung für diese Regel bietet gerade das Deduktionstheorem 1.2.

Diese Regeln können wir nun verwenden, um aus Formeln neue Formeln abzuleiten, die uns zusätzliche Einsichten in die Eigenschaften des durch die Formeln beschriebenen Modells oder Sachverhalts liefern.

Beispiel A.1 *Ableitung*
Um zu zeigen, daß sich aus der Gültigkeit von A, B und $(A \wedge B) \Rightarrow C$ die Gültigkeit von C ableiten läßt, kurz

$$A,\ B,\ (A \wedge B \Rightarrow C) \vdash C,$$

führen wir die folgende Ableitung durch:

1.	A	Voraussetzung
2.	B	Voraussetzung
3.	$A \wedge B \Rightarrow C$	Voraussetzung
4.	$A \wedge B$	$\wedge$-Einführung und 1., 2.
5.	C	$\Rightarrow$-Entfernung und 3., 4.

□

Die Ableitung in Beispiel A.1 kann man auch durch einen **Ableitungsbaum**, dessen Wurzel aus der Konklusion und dessen Blätter aus den Prämissen bestehen, darstellen:

$$\dfrac{\dfrac{A,\ B}{A \wedge B,} \qquad A \wedge B \Rightarrow C}{C}$$

Selbsttestaufgabe A.1 *Ableitung*
Zeigen Sie

$$A \wedge B \vdash A \vee B$$

durch Anwendung der oben eingeführten Schlußregeln. ◇

5 Schaltalgebra

5.1 Verknüpfungszeichen der Schaltalgebra

5.2 Logische Funktionen

5.3 Schaltsymbole der Grundverknüpfungen

5.4 Regeln der Schaltalgebra

5.5 Logikstufen

5.6 Darstellung der Elementarverknüpfungen in NAND- und NOR-Technik

5.7 Normalformen, Minterme und Maxterme

5.8 Minimierung

5.1 Verknüpfungszeichen der Schaltalgebra

Die Grundlage der digitalen Schaltungstechnik bildet eine zwei-elementige **Boolesche Algebra** auf der Menge B := { 0 , 1 } mit den Verknüpfungen:

* *Konjunktion*

$$B \times B \to B \quad , \quad (x,y) \mapsto x \wedge y$$

Verknüpfungstabelle:

x	y	x **v** y
0	0	0
0	1	0
1	0	0
1	1	1

* *Disjunktion*

$$B \times B \to B \quad , \quad (x,y) \mapsto x \vee y$$

Verknüpfungstabelle:

x	y	x **w** y
0	0	0
0	1	1
1	0	1
1	1	1

* Negation

$$B \to B \quad , \quad x \mapsto \overline{x}$$

Verknüpfungstabelle:

x	$\overline{x}$
0	1
1	0

Die algebraische Struktur (B; v, w, G) wird als Schaltalgebra bezeichnet - C. Shannon (1938).

Eine Schaltfunktion (logische Funktion) ist eine eindeutige Zuordnungsvorschrift, die jeder der 2^n Wertekombinationen der Schaltvariablen (binäre oder logische Variablen) $x_1, x_2, ..., x_n, x_i \ 0 \ \{0,1\}$ (i=1(1)n) eindeutig einen Funktionswert

$$y = f(x_1, x_2, ..., x_n) \ 0 \ \{0,1\}$$

zuweist.

Logische Verknüpfungszeichen (DIN 66000):

Verknüpfung	Symbol	Beispiel	
Negation	& , ¬	A , ¬A	nicht A
Konjunktion, UND-Verknüpf.	v	A v B	A und B
Disjunktion, ODER-Verknüpf.	w	A w B	A oder B
NAND-Verknüpf.	v	A v B A v B	A nand B, nicht(A und B)
NOR-Verknüpf.	w	A w B A w B	A nor B, nicht(A oder B)
Implikation	6	A 6 B	A Pfeil B
Äquivalenz	76	A 76 B	A Doppelpfeil B
Antivalenz, Exklusiv-ODER, XOR-Verknüpf.	76	A 76 B	A xor B

Nicht genormte Verknüpfungszeichen:

" C " für v (Und),

" + " für w (Oder),

" / " für : (Äquivalenz),

" / , r " für : (XOR).

Vorrangregeln:

wachsende
Priorität

8
|
| Negation (stärkste Bindung)
|
| UND, ODER, NAND, NOR
|
| XOR, Implikation, Äquivalenz

Beachte:
Da beispielsweise die Verknüpfungen UND und ODER die gleiche Priorität besitzen, müssen innerhalb einer Gleichung Klammern gesetzt werden!

Als abkürzende Schreibweise bezeichnet man das Weglassen des Verknüpfungszeichens bei der direkten UND-Verknüpfung logischer Variablen, z.B.:

$$(x_1 \cdot x_2 \cdot \overline{x}_3) \circ (\overline{x}_2 \cdot x_3) \;'\; x_1 x_2 \overline{x}_3 \circ \overline{x}_2 x_3$$

DIN 66000 *abkürzende Schreibweise*

Verknüpfungstabellen

$$y = f(x_1, x_2)$$

Arbeitstabelle:

x_1	x_2	y
0 V	0 V	0 V
0 V	4 V	4 V
4 V	0 V	4 V
4 V	4 V	4 V

Pegeltabelle:

x_1	x_2	y
L	L	L
L	H	H
H	L	H
H	H	H

Wahrheitstabellen:

Positive Logik

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

ODER - Verknüpfung

Negative Logik

x_1	x_2	y
1	1	1
1	0	0
0	1	0
0	0	0

UND - Verknüpfung

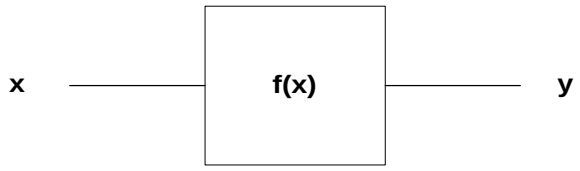
5.2 Logische Funktionen

Für n binäre Eingangsvariablen gibt es 2^n verschiedene Kombinationsmöglichkeiten, so daß für eine binäre Ausgangsgröße die Anzahl der möglichen Funktionen gleich

$$2^{2^n}$$

ist.

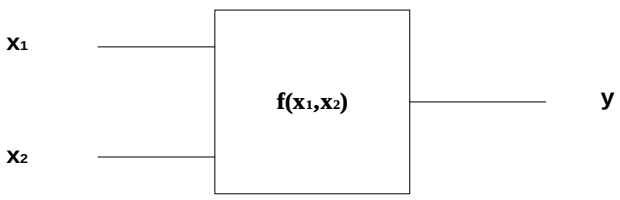
*** Funktionen für eine Eingangsvariable:**



$$y_i = f_i(x) \quad , \quad i = 0,1,2,3$$

x	0	1	***Logische Funktion***	***Bezeichnung***
y_0	0	0	$y_0 = 0$	Konstante 0
y_1	0	1	$y_1 = x$	Abbild, Treiber
y_2	1	0	$y_2 = \neg x$	Negation
y_3	1	1	$y_3 = 1$	Konstante 1

Logische Funktionen für zwei Eingangsvariablen:



$$y_i \text{ ' } f_i(x_1,x_2) \ , \quad i \text{ ' } 0,1,...,15$$

X1 X2	1 0 1 0 1 1 0 0	***Logische Funktion***	***Bezeichnung***
Y0	0 0 0 0	Y0 = 0	Konstante 0
Y1	0 0 0 1	Y1 = X1 w X2 = X1 w X2	NOR
Y2	0 0 1 0	Y2 = X1X2 = X1 v X2	Inhibition
Y3	0 0 1 1	Y3 = X2	Negation X2
Y4	0 1 0 0	Y4 = X1X2	Inhibition
Y5	0 1 0 1	Y5 = X1	Negation X1
Y6	0 1 1 0	Y6 = X1 <-> X2 = X1X2 w X1X2	XOR
Y7	0 1 1 1	Y7 = X1 v X2 = X1 v X2	NAND
Y8	1 0 0 0	Y8 = X1 v X2 = X1X2	UND
Y9	1 0 0 1	Y9 = X1 <-> X2 = X1X2 w X1X2	Äquivalenz
Y10	1 0 1 0	Y10 = X1	Identität X1
Y11	1 0 1 1	Y11 = X2 -> X1 = X1 w X2	Implikation
Y12	1 1 0 0	Y12 = X2	Identität X2
Y13	1 1 0 1	Y13 = X1 -> X2 = X1 w X2	Implikation
Y14	1 1 1 0	Y14 = X1 w X2	ODER
Y15	1 1 1 1	Y15 = 1	Konstante 1

Besondere praktische Relevanz besitzen die Funktionen UND, ODER, NEGATION, NAND, NOR und XOR!

Die Darstellung einzelner logischer Funktionen über die Grundverknüpfungen NEGATION, UND und ODER läßt sich anhand einer Wertetabelle überprüfen.

Beispiel:

$$y6 \text{ ' } x1 \text{ 7|6 } x2 \text{ ' } x1\overline{x2} \text{ º } \overline{x1}x2$$

$x1$	$x2$	$x1\overline{x2}$	$\overline{x1}x2$	$x1\overline{x2}$ º $\overline{x1}x2$	$y6$
1	1	0	0	0	0
0	1	0	1	1	1
1	0	1	0	1	1
0	0	0	0	0	0

#

Alle logischen Verknüpfungen für zwei Schaltvariablen lassen sich durch Konjunktion, Disjunktion und Negation darstellen!!

Def.:

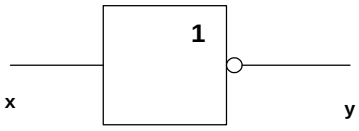
Ein System F von Verknüpfungen heißt *Verknüpfungsbasis* für eine Funktionsmenge F, wenn sich jede Funktion f 0 F mit diesen Verknüpfungen allein darstellen läßt.

F = { & , v , w } bildet eine Verknüpfungsbasis für { y0, y1,...,y15 }.

5.3 Schaltsymbole der Grundverknüpfungen

*** Negation (NICHT - Verknüpfung)**

Schaltsymbol - DIN 40900:



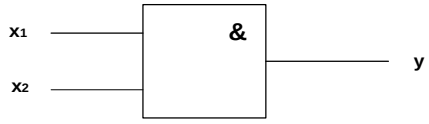
Funktion:

$$y = \overline{x} = \neg x$$

Wahrheitstabelle:

x	y
0	1
1	0

*** Konjunktion (UND - Verknüpfung)**

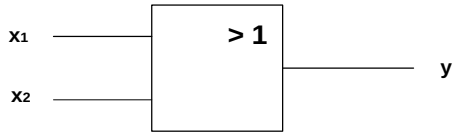


$$y = x_1 x_2 = x_1 \wedge x_2$$

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

„ *n Eingangsvariablen*: $y = x_1 x_2 \ldots x_n$ “

* Disjunktion (ODER - Verknüpfung)

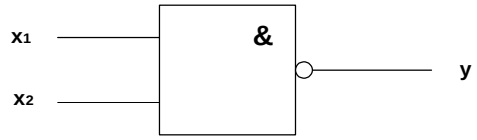


$y = x_1 \vee x_2$

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

n Eingangsvariablen: $y = x_1 \vee x_2 \vee \ldots \vee x_n$

* NAND - Verknüpfung (NOT AND - / NICHT UND - Verknüpfung)

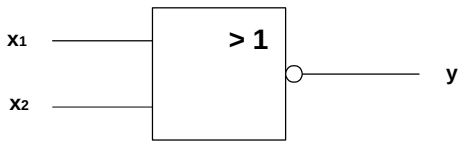


$y = \overline{x_1 x_2} = x_1 \bar{\vee} x_2$

x_1	x_2	y
0	0	1
0	1	1
1	0	1
1	1	0

f n Eingangsvariablen: $y = \overline{x_1 x_2 \ldots x_n}$

* NOR - Verknüpfung (NOT OR - / NICHT ODER - Verknüpfung)

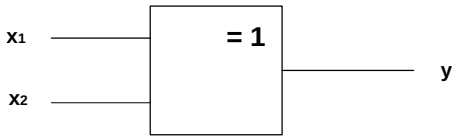


$$y = \overline{x_1 \vee x_2} = x_1 \overline{\vee} x_2$$

x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	0

n Eingangsvariablen: $y = \overline{x_1 \vee x_2 \vee \ldots \vee x_n}$

* XOR - Verknüpfung (EXKLUSIV ODER - / ANTIVALENZ - Verknüpfung)



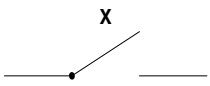
$$y = x_1 \veebar x_2 = \overline{x_1}\, x_2 \vee x_1 \overline{x_2}$$

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

5.4 Regeln der Schaltalgebra

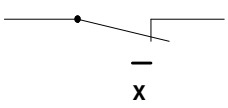
Elementarverknüpfungen		
NICHT	**UND**	**ODER**
$\overline{1}$ ' 0	0 v 0 ' 0	0 w 0 ' 0
$\overline{0}$ ' 1	0 v 1 ' 0	0 w 1 ' 1
	1 v 0 ' 0	1 w 0 ' 1
	1 v 1 ' 1	1 w 1 ' 1

Zur Erläuterung der Regeln der Algebra lassen sich einfache Serien- und Parallelschaltungen verwenden; hierbei bildet



einen **Schliesser** (Arbeitskontakt), wobei
x = 0 zu *Schalter geöffnet* und
x = 1 zu *Schalter geschlossen* korrespondiert,

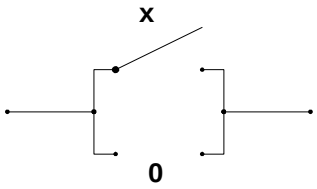
und



einen **Öffner** (Ruhekontakt), für
x = 0 ist der *Schalter geschlossen* und
x = 1 bedeutet *Schalter geöffnet*.

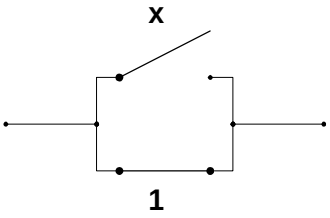
1.

$x \vee 0 = x$



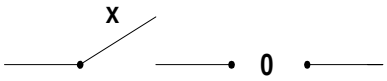
2.

$x \vee 1 = 1$



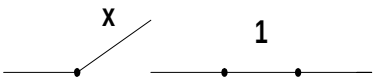
3.

$x \wedge 0 = 0$



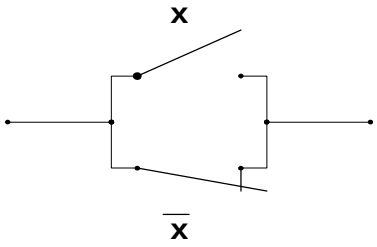
4.

$x \wedge 1 = x$



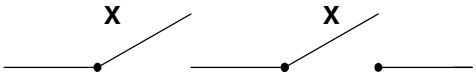
5.

$x \vee \overline{x} = 1$



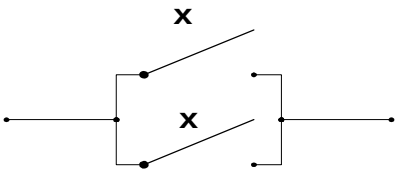
6.

$x \wedge x = x$



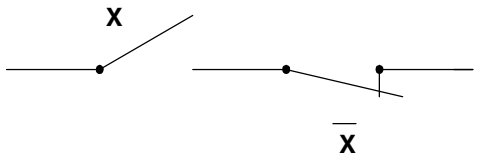
7.

$x \vee x = x$



8.

$x \vee \overline{x} \text{ ' } 0$



9.

$\overline{x} \text{ ' } x$

10. Kommutative Gesetze

a)

$x1 \vee x2 \vee x3 \text{ '} x2 \vee x1 \vee x3 \text{ '} x3 \vee x2 \vee x1 \text{ ' } . \; . \; .$

b)

$x1 \text{ w } x2 \text{ w } x3 \text{ '} x2 \text{ w } x1 \text{ w } x3 \text{ '} x3 \text{ w } x2 \text{ w } x1 \text{ ' } . \; . \; .$

11. Assoziative Gesetze

a)

$x1 \vee x2 \vee x3 \text{ '} x1 \vee (x2 \vee x3) \text{ '} (x1 \vee x2) \vee x3$

b)

$x1 \text{ w } x2 \text{ w } x3 \text{ '} x1 \text{ w } (x2 \text{ w } x3) \text{ '} (x1 \text{ w } x2) \text{ w } x3$

12. Distributive Gesetze

a)

$x1 \text{ w } (x2 \vee x3) \text{ '} (x1 \text{ w } x2) \vee (x1 \text{ w } x3)$

b)

$x1 \vee (x2 \text{ w } x3) \text{ '} (x1 \vee x2) \text{ w } (x1 \vee x3)$

13. De Morgansche Gesetze

a)

$$\overline{x1 \vee x2 \vee \ldots \vee xn} \;' \; \bar{x1} \wedge \bar{x2} \wedge \ldots \wedge \bar{xn}$$

b)

$$\overline{x1 \wedge x2 \wedge \ldots \wedge xn} \;' \; \bar{x1} \vee \bar{x2} \vee \ldots \vee \bar{xn}$$

14. Shannonsches Gesetz

$$\overline{f(x1, x2, \ldots xn \,;\, \vee, \wedge)} \;' \; f(\bar{x1}, \bar{x2}, \ldots \bar{xn} \,;\, \wedge \vee)$$

15. Kürzungsregeln

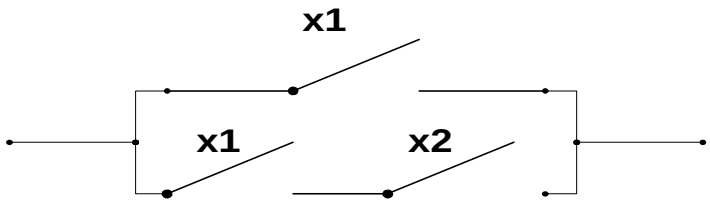
a)

$$x1 \wedge (x1 \vee x2) \;' \; x1$$

denn

$$x1 \wedge (x1 \vee x2) \;'^{4.}\; (x1 \vee 1) \wedge (x1 \vee x2) \;'^{12.}$$
$$'^{12.}\; x1 \vee (1 \wedge x2) \;'^{2.}\; x1 \vee 1 \;'^{4.}\; x1$$

bzw.



b)

$$x1 \vee (x1 \wedge x2) \;' \; x1$$

$$[\; x1 \vee (x1 \wedge x2) \;'^{1.}\; (x1 \wedge 0) \vee (x1 \wedge x2) \;'^{12.}$$
$$'^{12.}\; x1 \wedge (0 \vee x2) \;'^{3.}\; x1 \wedge 0 \;'^{1.}\; x1 \;]$$

c)

$$x1 \text{ w } (\bar{x1} \text{ v } x2) \text{ ' } x1 \text{ w } x2$$

$$x1 \text{ w } (\bar{x1} \text{ v } x2) \text{ '}^{12.} (x1 \text{ w } \bar{x1}) \text{ v } (x1 \text{ w } x2) \text{ '}^{5.} 1 \text{ v } (x1 \text{ w } x2) \text{ '}^{4.} x1 \text{ w } x2$$

d)

$$x1 \text{ v } (\bar{x1} \text{ w } x2) \text{ ' } x1 \text{ v } x2$$

$$x1 \text{ v } (\bar{x1} \text{ w } x2) \text{ '}^{12.} (x1 \text{ v } \bar{x1}) \text{ w } (x1 \text{ v } x2) \text{ '}^{8.} 0 \text{ w } (x1 \text{ v } x2) \text{ '}^{1.} x1 \text{ v } x2$$

e)

$$(x1 \text{ v } x2) \text{ w } (x1 \text{ v } \bar{x2}) \text{ ' } x1$$

$$[\ (x1 \text{ v } x2) \text{ w } (x1 \text{ v } \bar{x2}) \text{ '}^{12.} x1 \text{ v } (x2 \text{ w } \bar{x2}) \text{ '}^{5.} x1 \text{ v } 1 \text{ '}^{4.} x1\ ]$$

f)

$$(x1 \text{ w } x2) \text{ v } (x1 \text{ w } \bar{x2}) \text{ ' } x1$$

$$[\ (x1 \text{ w } x2) \text{ v } (x1 \text{ w } \bar{x2}) \text{ '}^{12.} x1 \text{ w } (x2 \text{ v } \bar{x2}) \text{ '}^{8.} x1 \text{ w } 0 \text{ '}^{1.} x1\ ]$$

16. Dualitätsprinzip

Mit jeder aus den Regeln der Schaltalgebra ableitbaren Aussage gilt auch die entsprechende Aussage, die sich bei Vertauschung von v mit w und 0 mit 1 ergibt - zwei derartige Aussagen heißen dual.

Der Grund hierfür ist die Struktur der Grundregeln:

$$a \text{ w } b \text{ ' } b \text{ w } a \quad , \quad a \text{ v } b \text{ ' } b \text{ v } a$$

$$a \text{ w } (b \text{ v } c) \text{ ' } (a \text{ w } b) \text{ v } (a \text{ w } c) \quad , \quad a \text{ v } (b \text{ w } c) \text{ ' } (a \text{ v } b) \text{ w } (a \text{ v } c)$$

$$a \text{ w } \bar{a} \text{ ' } 1 \quad , \quad a \text{ v } \bar{a} \text{ ' } 0$$

$$a \text{ w } 0 \text{ ' } a \quad , \quad a \text{ v } 1 \text{ ' } a$$

17. Negation von Schaltfunktionen

Die Negation einer Schaltfunktion ergibt sich über das Shannonsche Gesetz, indem man zur dualen Funktion übergeht und jede Variable einzeln negiert:

$$\overline{f(x1 \text{ , } x2 \text{ , } \ldots xn \text{ ; v , w})} \text{ ' } f(\bar{x1} \text{ , } \bar{x2} \text{ , } \ldots \bar{xn} \text{ ; w v}) \text{ .}$$

Beispiel:

$$y \text{ ' } (x1 \text{ w } x2 \text{ w } x3) \text{ v } (\bar{x1} \text{ w } x2 \text{ w } x3)$$

Duale Funktion:

$$y \text{ ' } (x1 \text{ v } x2 \text{ v } x3) \text{ w } (\bar{x1} \text{ v } x2 \text{ v } x3)$$

oder

$$y \text{ ' } x1\ x2\ x3 \text{ w } \bar{x1}\ x2\ x3$$

Negierte Funktion:

$$y \text{ ' } \bar{x1}\ \bar{x2}\ \bar{x3} \text{ w } x1\ \bar{x2}\ \bar{x3}$$

#

5.5 Logikstufen

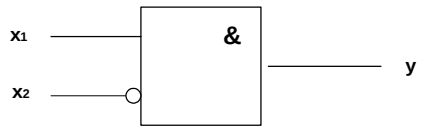
Allgemein bezeichnet man als Stufigkeit oder Verknüpfungstiefe einer digitalen Schaltung die Anzahl der Gatterstufen, die hintereinander geschaltet sind.

(i) Einstufige Logik

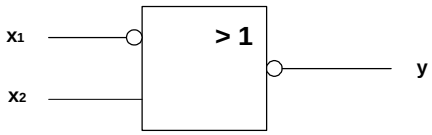
Eine digitale Schaltung ist einstufig, wenn zwischen Eingang und Ausgang nur eine Gatterstufe liegt; eine Negation am Ein- oder Ausgang wird allgemein nicht als separate Stufe gezählt!

Beispiele:

$$y = x1\,\bar{x2}$$



$$y = \overline{x1 \vee x2}$$



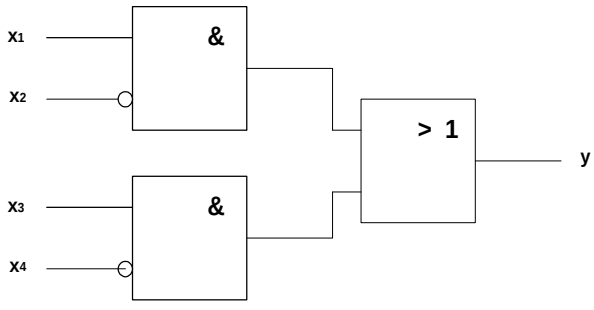
#

(ii) Zweistufige Logik

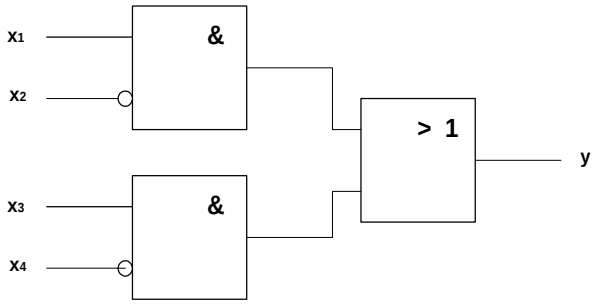
Eine digitale Schaltung ist zweistufig, wenn zwischen Ein- und Ausgang zwei Gatterstufen liegen.

Beispiel:

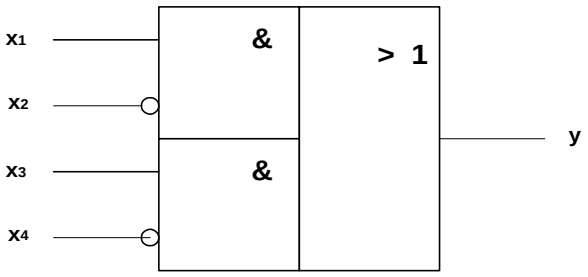
$$y = x1\bar{x2} \vee x3\bar{x4}$$



Vereinfachend stellt man die Schaltung



durch direktes Aneinanderfügen der Gatterstufen folgendermaßen dar:



(iii) n-stufige Logik

Eine digitale Schaltung ist n-stufig, wenn zwischen Eingang und Ausgang n Gatterstufen hintereinander geschaltet sind.

Beachte:

Die Verzögerungszeiten sämtlicher Stufen addieren sich!
Zeitkritische Schaltungen sollten auf einer zweistufigen Logik basieren.

5.6 Darstellung der Elementarverknüpfungen in NAND- und NOR-Technik

Die Elementarverknüpfungen NEGATION, UND sowie ODER lassen sich technisch durch die alleinige Verwendung von NAND-Gattern bzw. NOR-Gattern realisieren; d.h. { G , v } oder { G , w } stellen für sich allein Verknüpfungsbasen dar.

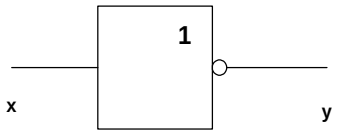
Die De Morganschen Gesetze

$$\overline{x \vee y} = \bar{x} \wedge \bar{y} \quad , \quad \overline{x \wedge y} = \bar{x} \vee \bar{y}$$

ermöglichen jeweils eine der beiden Verknüpfungen v bzw. w aus der Basis { G , v , w } zu entfernen.

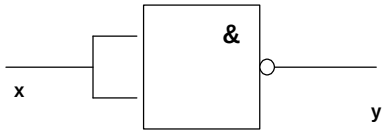
*** NEGATION**

$$y = \bar{x}$$



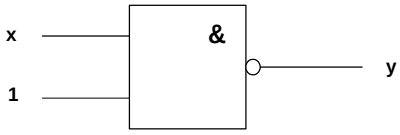
NAND-Technik:

$y = \overline{x \vee x}$ (*Regel* 6)

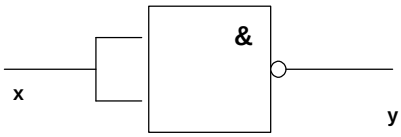


oder

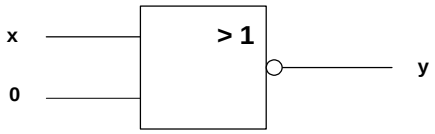
$y = \overline{x \vee 1}$ (*Regel* 4)



NOR-Technik:



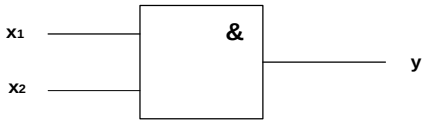
$$y \;'\; \overline{x \;w\; x} \quad (\textit{Regel}\ 7)$$



oder

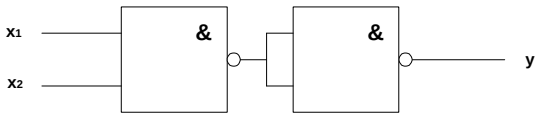
$$y \;'\; \overline{x \;w\; 0} \quad (\textit{Regel}\ 1)$$

*** UND-Verknüpfung (Konjunktion)**



$$y \;'\; x1 \;v\; x2$$

NAND-Technik:



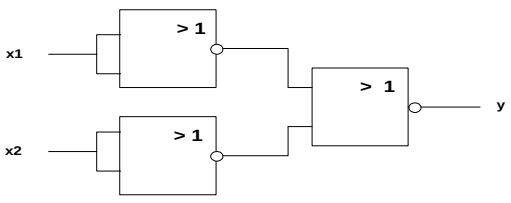
$$y \;'\; \overline{\overline{x1 \;v\; x2}} \quad (\textit{Regel}\ 9)$$

$$y \;'\; \overline{\overline{x1 \;v\; x2} \;v\; \overline{x1 \;v\; x2}} \quad (\textit{Regel}\ 6)$$

NOR-Technik:

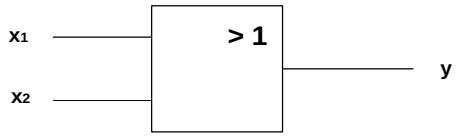
$$y \;'\; \overline{\overline{(x1 \;w\; x1)} \;v\; \overline{(x2 \;w\; x2)}} \quad (\textit{Regel}\ 7\ ,\ 9)$$

$$y \;'\; \overline{\overline{(x1 \;w\; x1)} \;w\; \overline{(x2 \;w\; x2)}} \quad (\textit{Regel}\ 13)$$



* ODER - Verknüpfung (Disjunktion)

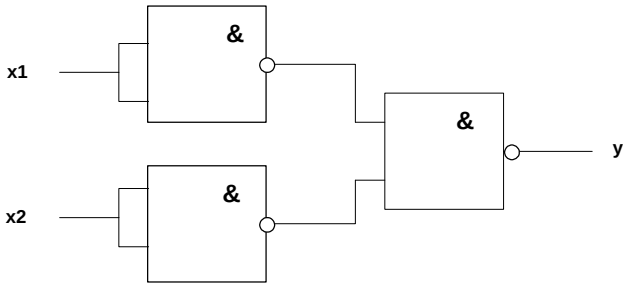
$$y \;'\; x1 \;\mathrm{w}\; x2$$



NAND - Technik:

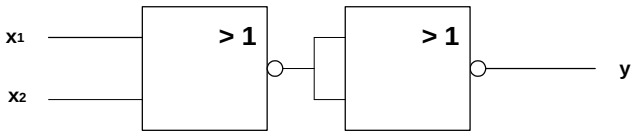
$$y \;'\; \overline{\overline{(x1 \;\mathrm{v}\; x1) \;\mathrm{w}\; (x2 \;\mathrm{v}\; x2)}} \quad (\; Regel\; 6\;,\; 9\;)$$

$$y \;'\; \overline{\overline{x1} \;\mathrm{v}\; \overline{x1} \;\mathrm{v}\; \overline{x2} \;\mathrm{v}\; \overline{x2}} \quad (\; Regel\; 13\;)$$



NOR - Technik:

$$y \;'\; \overline{\overline{x1 \;\mathrm{w}\; x2}} \;'\; \overline{\overline{x1 \;\mathrm{w}\; x2} \;\mathrm{w}\; \overline{x1 \;\mathrm{w}\; x2}} \quad (\; Regel\; 9\;,\; 7\;)$$



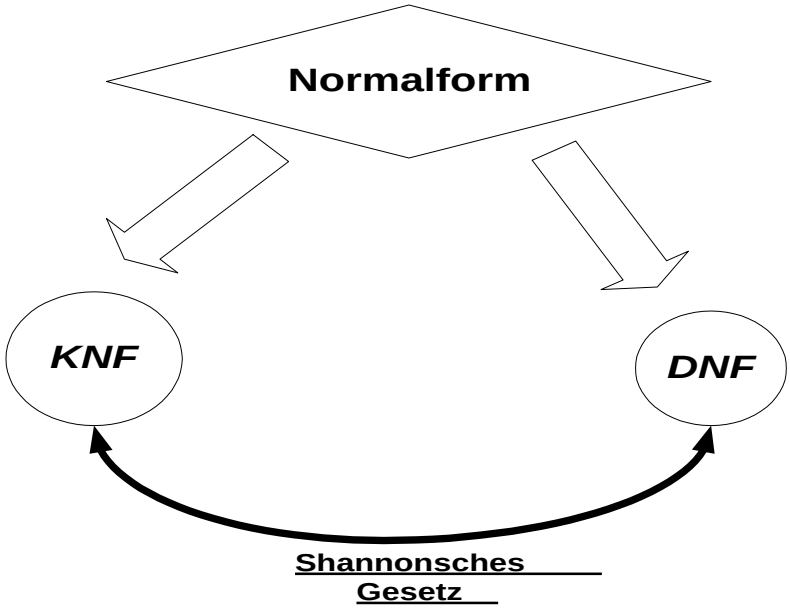
5.7 Normalformen, Minterme und Maxterme

Jede Schaltfunktion kann durch zwei gleichwertige Normalformen:

* KNF (konjunktive Normalform)

* DNF (disjunktive Normalform)

dargestellt werden - hierbei treten nur die Elementarverknüpfungen NEGATION, KONJUNKTION und DISJUNKTION auf.



Die Basiselemente der beiden Normalformen sind die Minterme für die DNF und die Maxterme für die KNF:

- Minterme (Vollkonjunktionen)

Minterme sind konjunktive Verknüpfungen aller Eingangsvariablen, bei denen jede Variable - negiert oder nichtnegiert - genau einmal auftritt.
Beispiel:

3 Variablen A , B , C;

$$A \vee \overline{B} \vee C \quad , \quad A \vee B \vee \overline{C} \quad , \quad A \vee B \vee C \quad , \ . \ . \ .$$

- Maxterme (Volldisjunktionen)

Maxterme sind disjunktive Verknüpfungen aller Eingangsvariablen, bei denen jede Variable - negiert oder nichtnegiert - genau einmal auftritt.

Beispiel:

3 Variablen A , B , C;

$$A \vee \overline{B} \vee C \quad , \quad A \vee B \vee \overline{C} \quad , \quad \overline{A} \vee \overline{B} \vee C \quad , \ . \ . \ .$$

#

Im Falle von n Schaltvariablen lassen sich insgesamt 2^n Min- und Maxterme bilden!

Minterme für zwei Eingangsvariablen:

	x1 *x2*	*x1x2*	*x1x2*	*x1x2*	*x1x2*
0	0 0	1	0	0	0
1	0 1	0	1	0	0
2	1 0	0	0	1	0
3	1 1	0	0	0	1

Jeder Minterm wird nur für eine Kombination der Eingangsvariablen “1", für alle übrigen ist er “0".

Der Minterm mit dem Wert “1" entsteht durch konjunktive Verknüpfung aller Schaltvariablen, wobei die Eingangsvariablen mit dem Wert “0" negiert werden.

Maxterme für zwei Eingangsvariablen:

	x1 *x2*	*x1* w *x2*	*x1* w *x2*	*x1* w *x2*	*x1* w *x2*
0	0 0	0	1	1	1
1	0 1	1	0	1	1
2	1 0	1	1	0	1
3	1 1	1	1	1	0

Der Maxterm mit dem Wert “0” entsteht durch disjunktive Verknüpfung aller Schaltvariablen, wobei die Eingangsvariablen mit dem Wert “1"
negiert werden.

Bemerkung: Abkürzende Notation für Minterme

Jeder Minterm läßt sich eindeutig über die Bitkombination der Eingangsvariablen bzw. deren Dezimalwert identifizieren, z.B.:

(0) ' $\bar{x1}\bar{x2}$, (1) ' $\bar{x1}x2$, (2) ' $x1\bar{x2}$, (3) ' $x1x2$.

Min- und Maxterme für drei Eingangsvariablen:

	x1	x2	x3	Minterm	Maxterm
0	0	0	0	$\bar{x1}\bar{x2}\bar{x3}$	$x1 \vee x2 \vee x3$
1	0	0	1	$\bar{x1}\bar{x2}x3$	$x1 \vee x2 \vee \bar{x3}$
2	0	1	0	$\bar{x1}x2\bar{x3}$	$x1 \vee \bar{x2} \vee x3$
3	0	1	1	$\bar{x1}x2x3$	$x1 \vee \bar{x2} \vee \bar{x3}$
4	1	0	0	$x1\bar{x2}\bar{x3}$	$\bar{x1} \vee x2 \vee x3$
5	1	0	1	$x1\bar{x2}x3$	$\bar{x1} \vee x2 \vee \bar{x3}$
6	1	1	0	$x1x2\bar{x3}$	$\bar{x1} \vee \bar{x2} \vee x3$
7	1	1	1	$x1x2x3$	$\bar{x1} \vee \bar{x2} \vee x3$

Bemerkung:
Der Min-/Maxterm korrespondiert zur kleinsten/größten unterscheidbaren Fläche im KV-Diagramm.

- Normalformen

Ziel: Aufstellen der Schaltfunktion für die Ausgangsvariable y anhand der Wahrheitstabelle

Die logische Funktion für y folgt indem entweder

(i) alle Minterme mit y = 1 disjunktiv miteinander verknüpft werden; dies liefert die DNF (Disjunktive Normalform)

oder

(ii) alle Maxterme mit y = 0 konjunktiv miteinander verknüpft werden; dies liefert die KNF (Konjunktive Normalform).

Beachte:
Disjunktive und konjunktive Normalform sind äquivalent (Dualitätsprinzip)!

In der Praxis wird zumeist der Schaltungsentwurf mit der DNF durchgeführt.

Beispiel 1:

$$y \text{ ' } f(x1 , x2 , x3) \text{ ' } (x1 \text{ 6 } x2) \text{ v } (\bar{x1} : x3)$$

Wahrheitstabelle:

	x1	*x2*	*x3*	*x1* 6 *x2*	$\bar{x1}$: *x3*	*y*
0	0	0	0	1	0	0
1	0	0	1	1	1	1
2	0	1	0	1	0	0
3	0	1	1	1	1	1
4	1	0	0	0	1	0
5	1	0	1	0	0	0
6	1	1	0	1	1	1
7	1	1	1	1	0	0

Bilde für alle Zeilen mit y = 1 die Minterme:

	Minterm
1	$x\bar{1}x\bar{2}x3$
3	$x\bar{1}x2x3$
6	$x1x2x\bar{3}$

Disjunktive Verknüpfung der Minterme liefert die DNF für y:

$$y \text{ ' } x\bar{1}x\bar{2}x3 \text{ w } x\bar{1}x2x3 \text{ w } x1x2x\bar{3} \quad .$$

#

Beispiel 2:

	x1	*x2*	*x3*	*y*	
0	0	0	0	0	
1	0	0	1	1	$x\bar{1}x\bar{2}x3$
2	0	1	0	1	$x\bar{1}x2x\bar{3}$
3	0	1	1	0	
4	1	0	0	1	$x1x\bar{2}x\bar{3}$
5	1	0	1	0	
6	1	1	0	0	
7	1	1	1	0	

/ DNF:

$$y \text{ ' } x\bar{1}x\bar{2}x3 \text{ w } x\bar{1}x2x\bar{3} \text{ w } x1x\bar{2}x\bar{3}$$

Alternative Schreibweise:

Das Ergebnis für die DNF läßt sich aufgrund der Gewichtung der Eingangsvariablen auch wie folgt notieren:

$$y' \text{ (1) w (2) w (4)} \quad , \quad y' \text{ w (1 , 2 , 4)} \quad .$$

Beispiel 3:

	$x1$	$x2$	$x3$	y	$\overline{y}$
0	0	0	0	0	1
1	0	0	1	1	0
2	0	1	0	1	0
3	0	1	1	1	0
4	1	0	0	0	1
5	1	0	1	1	0
6	1	1	0	1	0
7	1	1	1	0	1

Maxterme

DNF: (disjunktive Verknüpfung der Minterme)

$$y' \; \overline{x1}\,\overline{x2}x3 \text{ w } \overline{x1}x2\overline{x3} \text{ w } \overline{x1}x2x3 \text{ w } x1\overline{x2}x3 \text{ w } x1x2\overline{x3}$$

KNF: (konjunktive Verknüpfung der Maxterme)

$$y' \text{ (}x1 \text{ w } x2 \text{ w } x3\text{) v (}\overline{x1} \text{ w } x2 \text{ w } x3\text{) v (}\overline{x1} \text{ w } \overline{x2} \text{ w } \overline{x3}\text{)}$$

Beachte:
Die KNF liefert Vorteile, wenn die y-Spalte weniger 0-Werte als 1-Werte enthält!

Die KNF besitzt 3 Maxterme entsprechend einer Realisierung mit
3 ODER- und 1 UND-Gatter;
die DNF umfaßt 5 Minterme und führt auf 5 UND- und 1 ODER-Gatter.

Bemerkung:

Die KNF folgt aus der DNF für die negierte Schaltfunktion

$$\bar{y} \text{ ' } \bar{x1}\bar{x2}\bar{x3} \text{ w } x1\bar{x2}\bar{x3} \text{ w } x1x2x3$$

mit Hilfe des Shannonschen Gesetzes

$$\overline{f(x1 , x2 , x3 ; w , v)} \text{ ' } f(\bar{x1} , \bar{x2} , \bar{x3} ; v , w)$$

zu:

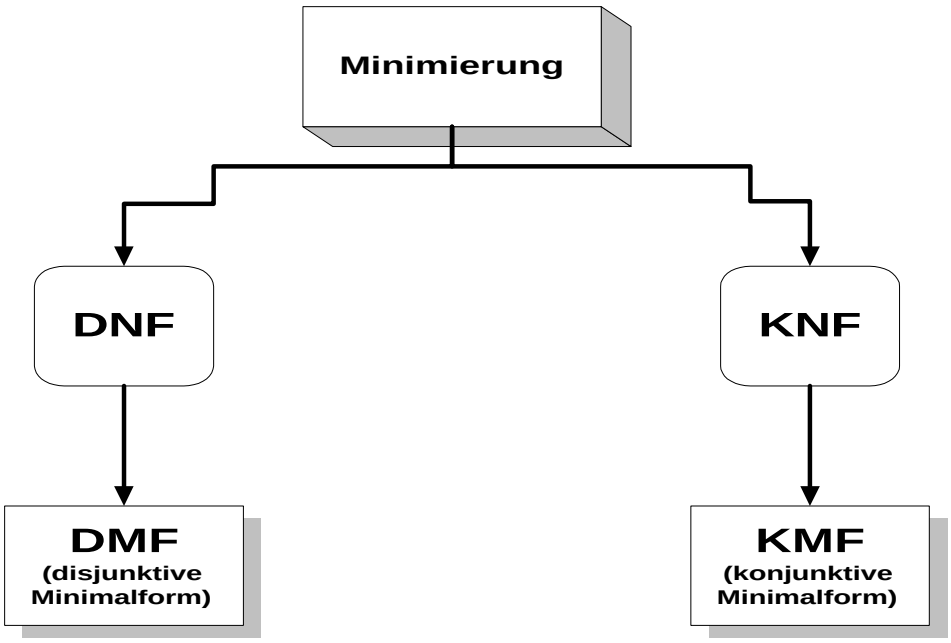
$$y \text{ ' } (x1 \text{ w } x2 \text{ w } x3) \text{ v } (\bar{x1} \text{ w } x2 \text{ w } x3) \text{ v } (\bar{x1} \text{ w } \bar{x2} \text{ w } \bar{x3}) \quad .$$

#

5.8 Minimierung von Schaltfunktionen

In der Praxis ist eine Schaltungsrealisierung anhand der Normalformen oftmals unvorteilhaft, so daß eine Minimierung der logischen Funktion - Elimination der unnötigen Variablen - erhebliche Vereinfachungen liefert.

Die Minimierung der DNF führt auf die disjunktive, minimale Gleichung oder disjunktive Minimalform (DMF); die Minimierung der KNF liefert die konjunktive, minimale Gleichung bzw. die konjunktive Minimalform (KMF).



Wesentliche Aspekte der Minimierung sind:

* es existieren zumeist mehrere gleichwertige Lösungen,
* ein Vergleich der DMF mit der KMF liefert die minimale logische Gleichung,
* die KMF folgt aus der negierten DMF,
* die DMF folgt aus der negierten KMF,
* der Schaltungsaufwand für die KMF und die negierte DMF ist gleich groß, gleiches gilt für die DMF und die negierte KMF.

Beispiel:

Die Minimierung einer Schaltfunktion lieferte
für die **DMF**:

$$' \; (x1 \vee \bar{x4}) \wedge (x1 \vee \bar{x2}) \wedge (\bar{x2} \vee \bar{x3}) \wedge (\bar{x3} \vee \bar{x4}) \quad (1$$

und für die **KMF**:

$$y \; ' \; (x1 \wedge \bar{x3}) \vee (\bar{x2} \wedge \bar{x4}) \quad (2) \quad .$$

Herleitung der negierten KMF aus der DMF:

Negation der Gleichung (1) bedeutet

$$\bar{y} \; ' \; \overline{(x1 \vee x4) \wedge (x1 \vee x2) \wedge (x2 \vee x3) \wedge (x3 \vee x4)}$$

und das Shannonsche Gesetz liefert mit

$$\bar{y} \; ' \; (\bar{x1} \wedge x4) \vee (\bar{x1} \wedge x2) \vee (x2 \wedge x3) \vee (x3 \wedge x4)$$

die **negierte KMF**:

$$' \; \overline{(x1 \wedge x4) \vee (x1 \wedge x2) \vee (x2 \wedge x3) \vee (x3 \wedge x4)} \quad (3$$

Herleitung der negierten DMF aus der KMF:

$$\bar{y} \; ' \; \overline{(x1 \wedge x3) \vee (x2 \wedge x4)} \; ,$$

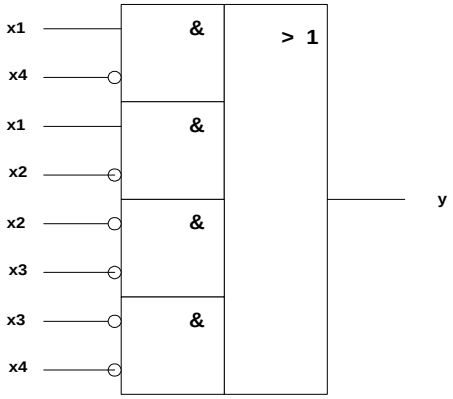
$$\bar{y} \; ' \; (\bar{x1} \vee x3) \wedge (x2 \vee x4)$$

Negation von (2) und Shannonsches Gesetz liefern
und somit die **negierte DMF**:

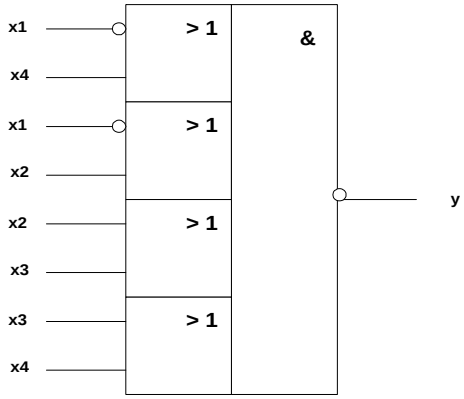
$$y \; ' \; \overline{(x1 \vee x3) \wedge (x2 \vee x4)} \quad (4) \quad .$$

Schaltungstechnische Realisierung:

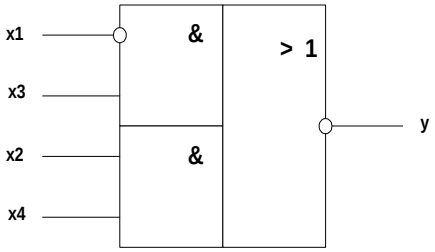
DMF



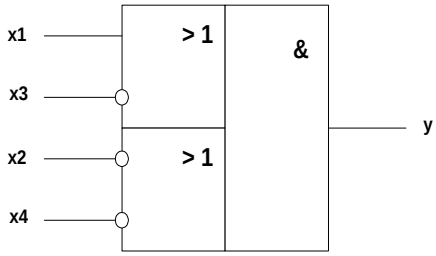
negierte KMF



negierte DMF



KMF



Die disjunktive (konjunktive) Schaltung folgt aus der konjunktiven (disjunktiven) durch Vertauschen der UND- und ODER-Gatter und Negation aller Eingänge sowie des Ausgangs!

#

Minimierungsverfahren

Die wichtigsten Verfahren zur Vereinfachung von Schaltfunktionen sind:

- Algebraische Umformungen

  Mit Hilfe der Regeln 1. - 15. der Booleschen Algebra wird die Gleichung sukzessive vereinfacht.

  Beispiel:

$$y = (\bar{x1} \vee \bar{x2} \vee \bar{x3}) \wedge (\bar{x1} \vee x2 \vee \bar{x3}) \wedge (x1 \vee \bar{x2} \vee x3) \wedge (x1 \vee x2 \vee x3)$$

$$= (\bar{x1} \vee \bar{x3})(\bar{x2} \wedge x2) \wedge (x1 \vee x3)(\bar{x2} \wedge x2)$$

$$= (\bar{x1} \vee \bar{x3}) \wedge (x1 \vee x3) \qquad [\ddot{A}quivalenz]$$

#

- Algorithmische Verfahren

  Programme zur Schaltungsminimierung basieren auf algorithmischen Methoden, die nahezu eine beliebig große Anzahl von Variablen verarbeiten können. Das bekannteste Verfahren ist der Quine-McCluskey-Algorithmus.

- Graphische Verfahren

  Karnaugh-Veitch-Diagramme eigen sich für die Minimierung logischer Funktionen bis zu fünf Eingangsvariablen.

Karnaugh-Veitch-Diagramme (KV-Diagramme)

Hiermit können sowohl die DMF als auch die KMF aufgestellt werden; i.f. wird die DMF diskutiert - die KMF folgt mittels Negation und dem Shannonschen Gesetz aus der DMF.

Das KV-Diagramm ist eine alternative schachbrettartige Darstellung der Wahrheitstabelle mit 2^n Feldern für n Eingangsvariablen.

Jedes Feld des KV-Diagramms korrespondiert zu einem Minterm und benachbarte Felder unterscheiden sich nur in einer Eingangsvariablen.

In die einzelnen Felder werden die Werte 0 , 1 oder * der Ausgangsvariablen eingetragen. “*” bedeutet unbestimmter Funktionswert
(redundanter Term, don’t care term) und kann beliebig durch “1" oder “0" ersetzt werden.

Grundprinzip:

Benachbarte 1-Felder lassen sich mit dem Distributiven Gesetz zusammenfassen:

$$(x1 \vee x2) \wedge (x1 \vee \overline{x2}) = x1 \vee (x2 \wedge \overline{x2}) = x1 \quad .$$

Ziel:

Zusammenfassen möglichst vieler benachbarter 1-Felder zu einem Block, wobei die Gesamtzahl der Blöcke im KV-Diagramm zu minimieren ist.

Grundlegende Regeln:

1. Die Werte der Ausgangsvariablen 0, 1 oder * werden für alle 2^n Kombinationen der Eingangsvariablen (2^n Minterme) in die entsprechenden 2^n Felder des KV-Diagramms eingetragen.

2. Benachbarte 1-Felder werden zu einem Block zusammengefaßt, wobei redundante Felder (Lückenfüller) ebenso berücksichtigt werden.

3. Ein Block umfaßt 2^i Felder mit i=0, 1, ..., n.

4. ZweiBlöcke, die sich nur in einer Variablen unterscheiden, sind benachbart und lassen sich zu einem größeren Block zusammenfassen.

5. Ein 1-Feld oder redundantes Feld kann in verschiedenen Blöcken enthalten sein.

6. Jeder Block wird durch die konjunktive Verknüpfung der zugehörigen Eingangsvariablen beschrieben.

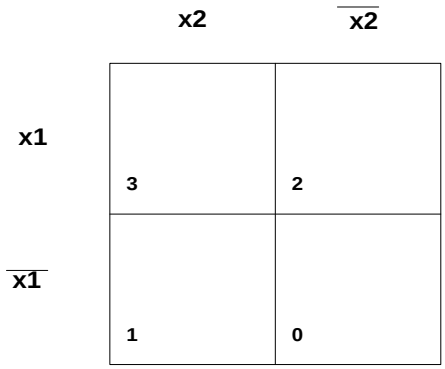
7. Die logische Gleichung folgt mit der disjunktiven Verknüpfung der einzelnen Blöcke.

8. Die resultierende logische Gleichung ist genau dann minimal, wenn die Blöcke maximale Größe besitzen und die Anzahl der Blöcke minimal ist.

Bemerkung:
Das Zusammenfassen der 0-Felder zu Blöcken liefert die negierte DMF und mit dem Shannonschen Gesetz folgt hieraus die KMF.

KV-Diagramm für zwei Eingangsvariablen

	x1	*x2*	*Minterm*
0	0	0	¬x1 ¬x2
1	0	1	¬x1 x2
2	1	0	x1 ¬x2
3	1	1	x1 x2



Beachte:

x2 (x1) hat die Wertigkeit 2^0 (2^1); die einzelnen Variablen werden außen am KV-Diagramm in fortlaufender Reihenfolge aufgeführt, wobei man mit der Eingangsvariablen, die in der Wahrheitstabelle rechts steht (Wertigkeit 2^0), beginnt und diese am oberen Rand plaziert und anschließend entgegen dem Uhrzeigersinn mit der Bezeichnung fortschreitet.

Wesentlich für den Aufbau der KV-Diagramme ist die Einschrittigkeit benachbarter Felder!

x1 x2	x1 ¬x2
¬x1 x2	¬x1 ¬x2

Vereinfachungsblöcke:

Block	Zahl der Eingangsvariablen
1 Feld (2^0)	2
2 Felder (2^1)	1
4 Felder (2^2)	0

Beispiel 1:

	x1	x2	y
0	0	0	1
1	0	1	0
2	1	0	1
3	1	1	0

DNF:

$$y \text{ ' } \bar{x1}\bar{x2} \text{ w } x1\bar{x2}$$

KV-Diagramm:

	X2	¬X2
X1	**0** 3	**1** 2
¬X1	**0** 1	**1** 0

DMF:

Die Felder 0 und 2 bilden einen Vereinfachungsblock, der nur von ¬x2 abhängt.

	X2	¬X2
X1	**0** 3	**1** 2
¬X1	**0** 1	**1** 0

ž $y \text{ ' } \bar{x2}$

Beachte:
Das Zusammenfassen der 0-Felder 1 und 3 liefert die negierte DMF:

$$\bar{y} \text{ ' } x2 \quad .$$

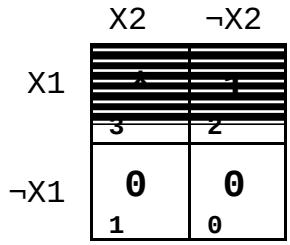
Beispiel 2:

	x1	x2	y
0	0	0	0
1	0	1	0
2	1	0	1
3	1	1	*

DNF:

$$y \text{ ' } x1\bar{x2}$$

KV-Diagramm:



DMF:

Das Feld 2 und das redundante Feld 3 bilden einen Vereinfachungsblock:

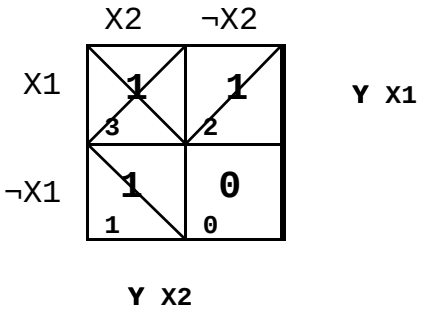
$$y \text{ ' } x1 \quad .$$

#

Beispiel 3:

DNF:

$$y \leftrightarrow x1x2 \vee \bar{x1}x2 \vee x1\bar{x2}$$



DMF:

$$y \leftrightarrow x1 \vee x2$$

Alternativ lässt sich das Ergebnis natürlich auch anhand algebraischer Umformungen herleiten:

$$y \leftrightarrow x1x2 \vee \bar{x1}x2 \vee x1\bar{x2} \leftrightarrow x2\,(x1 \vee \bar{x1}) \vee x1\bar{x2} \leftrightarrow$$
$$x2 \vee x1\bar{x2} \leftrightarrow (x2 \vee \bar{x2}) \wedge (x2 \vee x1) \leftrightarrow x2 \vee x1.$$

#

Beispiel 4:

Sind die 1-Felder diagonal angeordnet, so ist keine Blockbildung möglich; in der logischen Funktion tritt die Antivalenz- bzw. Äquivalenzfunktion auf!

	X2	¬X2
X1	**0** 3	**1** 2
¬X1	**1** 1	**0** 0

y ' *x1* 7|6 *x2* ' $\bar{x1}x2$ w $x1\bar{x2}$

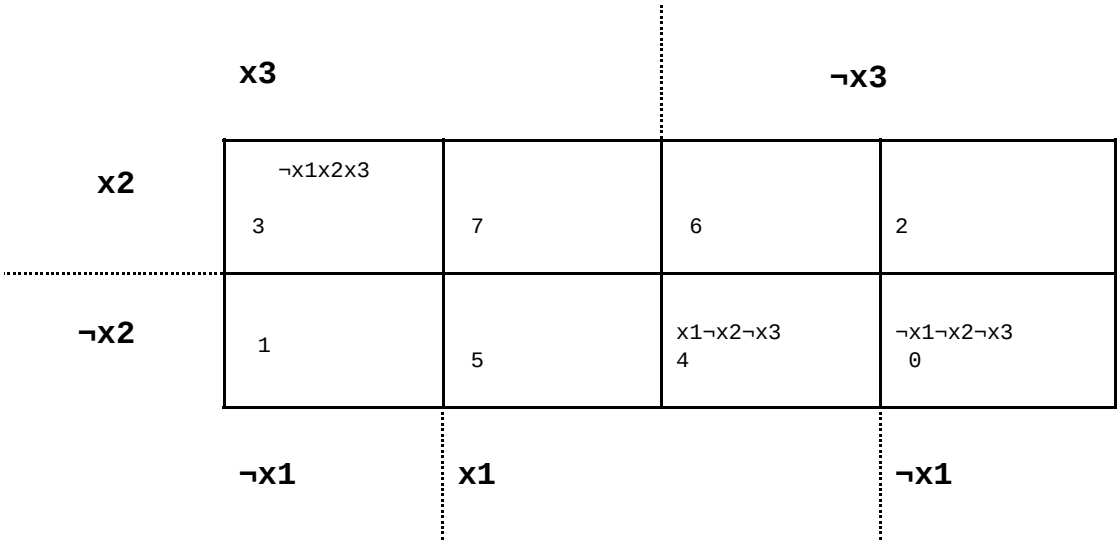
	X2	¬X2
X1	**1** 3	**0** 2
¬X1	**0** 1	**1** 0

y ' *x1* : *x2* ' *x1x2* w $\bar{x1}\bar{x2}$.

#

KV-Diagramm für drei Eingangsvariablen

	x1	*x2*	*x3*	*Minterm*
0	0	0	0	¬x1 ¬x2 ¬x3
1	0	0	1	¬x1 ¬x2 x3
2	0	1	0	¬x1 x2 ¬x3
3	0	1	1	¬x1 x2 x3
4	1	0	0	x1 ¬x2 ¬x3
5	1	0	1	x1 ¬x2 x3
6	1	1	0	x1 x2 ¬x3
7	1	1	1	x1 x2 x3



Vereinfachungsblöcke:

Block	Zahl der Eingangsvariablen
1 Feld (2^0)	3
2 Felder (2^1)	2
4 Felder (2^2)	1
8 Felder (2^3)	0

Beispiel 5:

	x1	x2	x3	y1	y2
0	0	0	0	1	1
1	0	0	1	1	1
2	0	1	0	1	1
3	0	1	1	0	0
4	1	0	0	0	*
5	1	0	1	0	*
6	1	1	0	0	0
7	1	1	1	0	0

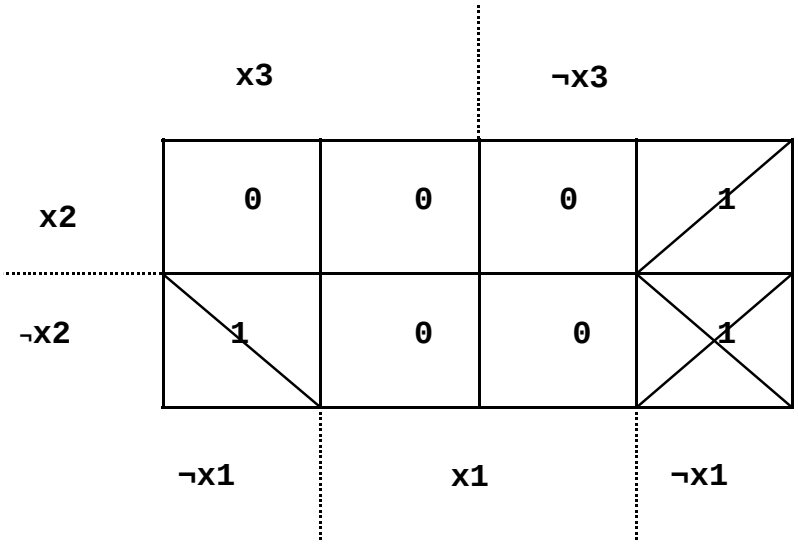
DNF:

$$y1 \text{ ' } \bar{x1}x2\bar{x3} \text{ w } \bar{x1}x2x3 \text{ w } x1\bar{x2}\bar{x3}$$

$$y2 \text{ ' } \bar{x1}\bar{x2}\bar{x3} \text{ w } \bar{x1}\bar{x2}x3 \text{ w } \bar{x1}x2\bar{x3} \text{ w } x1\bar{x2}\bar{x3} \text{ w } x1\bar{x2}x3 ,$$

$$y2 \text{ ' } y1 \text{ plus redundante Terme: } x1\bar{x2}\bar{x3} , \; x1\bar{x2}x3$$

(i) Minimierung für y1:



Zusammenfassen der Felder 2 und 0 (/): $x1x3$

Zusammenfassen der Felder 1 und 0 (\ ): $x1x2$

Beachte:

Am Rand liegende Felder können benachbart sein und zusammengefasst werden!

DMF:

$$y1 \text{ ' } \bar{x1}\bar{x2} \text{ w } \bar{x1}\bar{x3} \qquad (1)$$

negierte DMF:

Zusammenfassen der 0-Felder
(Felder 4, 5, 6, 7 µ x1, Felder 3, 7 µ x2x3)

Y

$$\bar{y1} \text{ ' } x1 \text{ w } x2x3$$

bzw.

$$y1 \text{ ' } \overline{x1 \text{ w } x2x3} \qquad (2)$$

Schaltungstechnische Realisierung:

(1) erfordert 2 UND- und 1 ODER-Verknüpfung;
(2) erfordert 1 UND- und 1 NOR-Verknüpfung.

Die konjunktiven, minimalen Gleichungen ergeben sich direkt aus den Gleichungen (1) und (2) mit Hilfe des Shannonschen Gesetzes:

negierte KMF:

(1), Regel 14 Y

$$y1 \text{ ' } \overline{(x1 \text{ w } x2) \text{ v } (x1 \text{ w } x3)} \qquad (1')$$

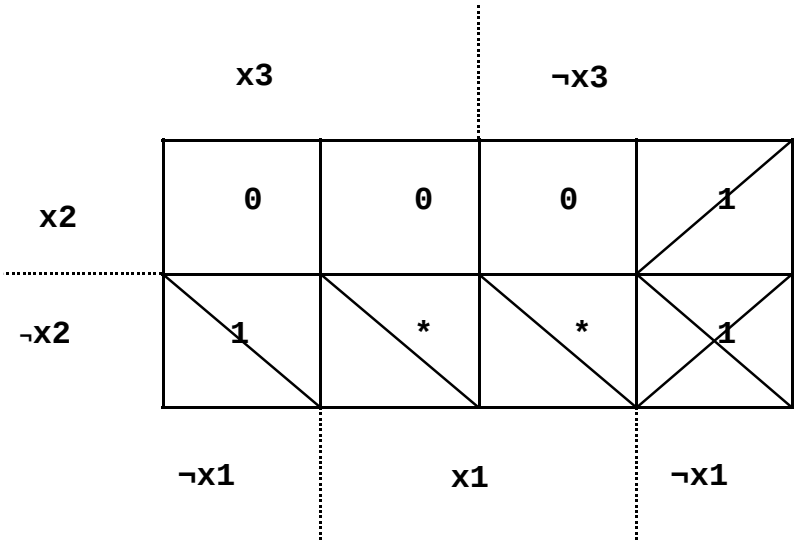
KMF:

(2), Regel 14 Y

$$y1 \text{ ' } \bar{x1} \text{ v } (\bar{x2} \text{ w } \bar{x3}) \qquad (2')$$

(ii) Minimierung für y2:

“ * “ – “ 1 “



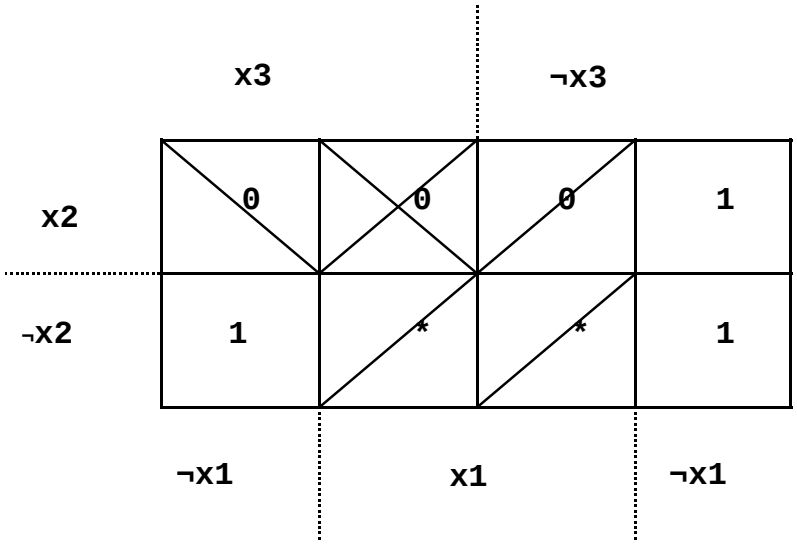
Zusammenfassen der Felder 2 und 0 (/): *x1x3*

Zusammenfassen der Felder 1, 5, 4 und 0 (\ ): *x2*

DMF:

$$y2 = \bar{x2} \vee \bar{x1}\bar{x3} \qquad (3)$$

Zusammenfassen der 0-Felder:
“ * “ – “ 0 “



Der Zweierblock (3,7) und der Viererblock (4,5,6,7) liefern:

$$y\bar{2} \text{ ' } x1 \text{ w } x2x3$$

bzw. folgt für die negierte DMF:

$$y2 \text{ ' } \overline{x1 \text{ w } x2x3} \qquad (4) \text{ .}$$

Beachte:
Bezüglich einer technischen Realisierung sind (3) und (4) gleichwertig.

Für die negierte KMF ergibt sich hier:

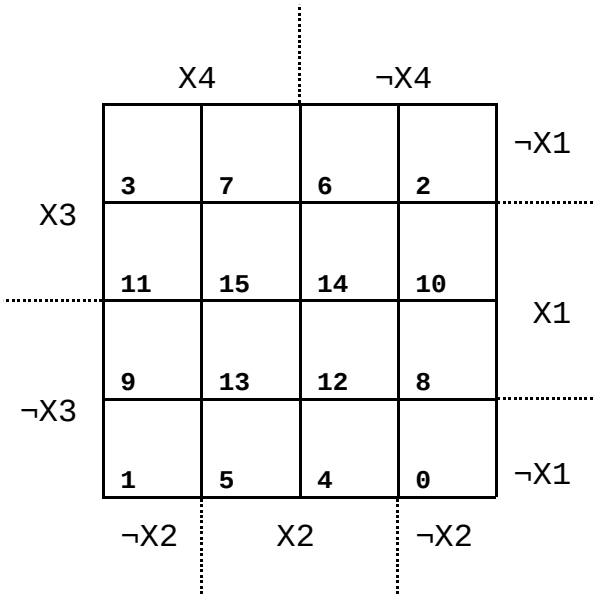
$$y2 \text{ ' } \overline{x2 \text{ v } (x1 \text{ w } x3)} \qquad (3^{)})$$

und für die KMF:

$$y2 \text{ ' } \bar{x1} \text{ v } (\bar{x2} \text{ w } \bar{x3}) \qquad (4^{)}) \text{ .}$$

KV-Diagramm für vier Eingangsvariablen

	x1	*x2*	*x3*	*x4*	*Minterm*
0	0	0	0	0	¬x1 ¬x2 ¬x3 ¬x4
1	0	0	0	1	¬x1 ¬x2 ¬x3 x4
2	0	0	1	0	¬x1 ¬x2 x3 ¬x4
.		.			.
.		.			.
.		.			.
14	1	1	1	0	x1 x2 x3 ¬x4
15	1	1	1	1	x1 x2 x3 x4



Vereinfachungsblöcke:

Block	Zahl der Eingangsvariablen
1 Feld (2^0)	4
2 Felder (2^1)	3
4 Felder (2^2)	2
8 Felder (2^3)	1
16 Felder (2^4)	0

Beispiel 6:

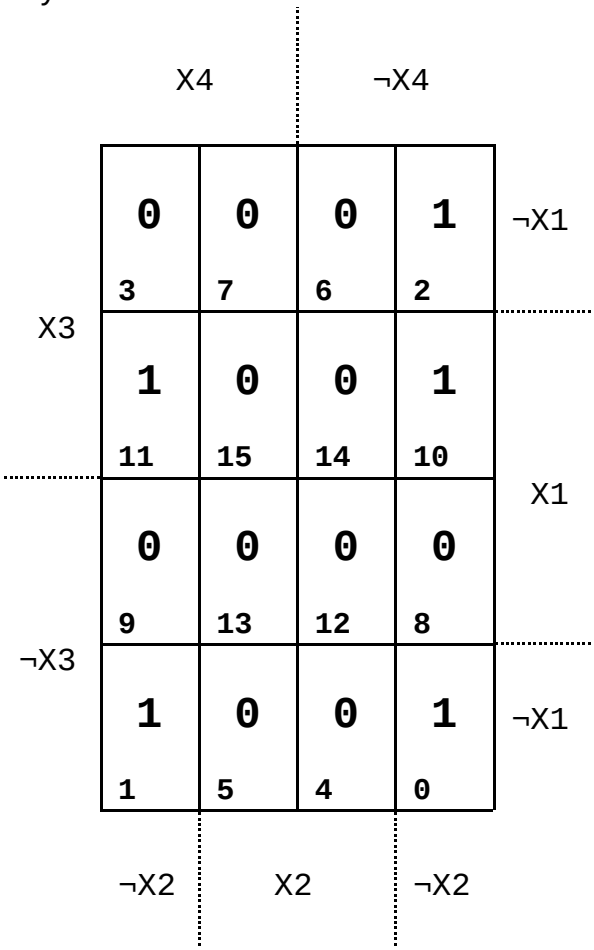
	x1	x2	x3	x4	y1	y2
0	0	0	0	0	1	1
1	0	0	0	1	1	1
2					1	1
3					0	*
4					0	0
5					0	0
6					0	0
7					0	0
8					0	*
9					0	*
10					1	1
11					1	1
12					0	*
13					0	0
14					0	0
15	1	1	1	1	0	*

DNF:

$y1 \text{ ' } \bar{x1}x2\bar{x3}\bar{x4} \text{ w } \bar{x1}\bar{x2}\bar{x3}x4 \text{ w } \bar{x1}\bar{x2}x3\bar{x4} \text{ w } x1\bar{x2}x3\bar{x4} \text{ w } x1\bar{x2}x3x4$

y2 ' y1 plus redundante Terme: (3) , (8) , (9) , (12) , (15)

(i) Minimierung für y1:



Zusammenfassen der Felder 2 und 10 : *x2x3x4*

Zusammenfassen der Felder 11 und 10 : *x1x2x3*

Zusammenfassen der Felder 1 und 0 : *x1x2x3*

Beachte:
Anstelle der Zusammenfassung der Felder 2 und 10 können alternativ auch die Felder 2 und 0 einen Block bilden!

DMF:

$$y1 = \bar{x2}x3\bar{x4} \vee x1\bar{x2}x3 \vee \bar{x1}\bar{x2}\bar{x3} \qquad (1)$$

negierte DMF:

Zusammenfassen der 0-Felder

$$\bigvee$$

$$y1 = \overline{x2 \vee x1x3 \vee x1x3x4} \qquad (2)$$

((2) ist schaltungstechnisch günstiger!)

Die konjunktiven, minimalen Gleichungen ergeben sich direkt aus den Gleichungen (1) und (2) mit Hilfe des Shannonschen Gesetzes:

negierte KMF:

(1), Regel 14 $\bigwedge$

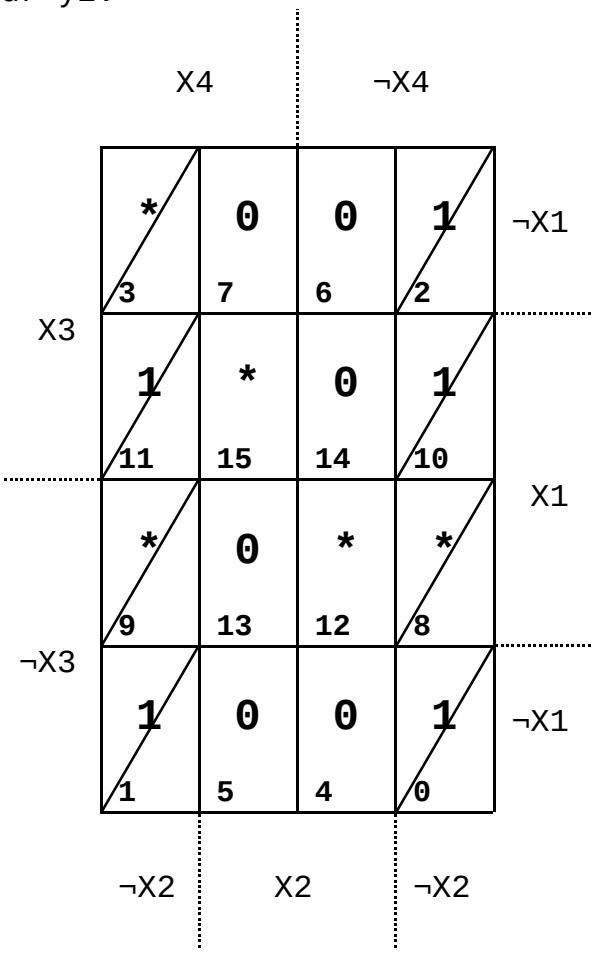
$$y1 = \overline{(x2 \vee x3 \vee x4) \wedge (x1 \vee x2 \vee x3) \wedge (x1 \vee x2 \vee x3)} \qquad (1')$$

KMF:

(2), Regel 14 $\bigwedge$

$$y1 = \bar{x2} \wedge (\bar{x1} \vee x3) \wedge (x1 \vee \bar{x3} \vee \bar{x4}) \qquad (2')$$

(ii) Minimierung für y2:



Die Berücksichtigung der redundanten Felder liefert für die minimale logische Funktion:

$$y2 \text{ ' } x2 \quad .$$

KV-Diagramme für fünf Eingangsvariablen:

Die Eingangsvariable X5 ist in der folgenden Wahrheitstabelle das LSB mit einer Wertigkeit 2^0 und X1 korrespondiert zum MSB mit der Wertigkeit 2^4.

	X1	X2	X3	X4	X5	Minterm
0	0	0	0	0	0	¬X1 ¬X2 ¬X3 ¬X4 ¬X5
1	0	0	0	0	1	¬X1 ¬X2 ¬X3 ¬X4 X5
2	0	0	0	1	0	¬X1 ¬X2 ¬X3 X4 ¬X5
3	0	0	0	1	1	¬X1 ¬X2 ¬X3 X4 X5
.			.			.
.			.			.
.			.			.
31	1	1	1	1	1	X1 X2 X3 X4 X5

Aufbau des KV-Diagramms mit Minterm-Nummerierung:

	¬X1		X1		¬X1			
	X5				¬X5			
X4	3	7	23	19	18	22	6	2
	11	15	31	27	26	30	14	10
¬X4	9	13	29	25	24	28	12	8
	1	5	21	17	16	20	4	0
	¬X3	X3		¬X3		X3		¬X3

Allgemein lassen sich hier folgende Vereinfachungsblöcke bilden:

Block	Zahl der Eingangsvariablen
1 Feld	5 Eingangsvariablen
2 Felder	4 "
4 "	3 "
8 "	2 "
16 "	1 "
32 "	0 "

.

KV-Diagramme für sechs Variablen (64 Minterme) können als ebene oder räumliche Darstellungen gezeichnet werden; ähnlich wie bei fünf logischen Variablen stellt sich die Auswertung als relativ beschwerlich dar - insbesondere können benachbarte Randfelder nur schwer identifiziert werden. In der Praxis verwendet man aus diesem Grunde algorithmische Verfahren zur Schaltungsminimierung wie z.B. das Verfahren von Quine-McCluskey.

Beispiel:

Wahrheitstabelle:

	X1	X2	X3	X4	X5	Y
4	0	0	1	0	0	1
5	0	0	1	0	1	1
6	0	0	1	1	0	1
7	0	0	1	1	1	1
16	1	0	0	0	0	1
24	1	1	0	0	0	1
			Rest			0

KV-Diagramm:

	¬X1		X1			¬X1		
	X5				¬X5			
X4	**0** 3	**1** 7	**0** 23	**0** 19	**0** 18	**0** 22	**1** 6	**0** 2	¬X2
	0 11	**0** 15	**0** 31	**0** 27	**0** 26	**0** 30	**0** 14	**0** 10	X2
¬X4	**0** 9	**0** 13	**0** 29	**0** 25	**1** 24	**0** 28	**0** 12	**0** 8	
	0 1	**1** 5	**0** 21	**0** 17	**1** 16	**0** 20	**1** 4	**0** 0	¬X2
	¬X3	X3		¬X3		X3		¬X3	

Zusammenfassen (Blocken) der 1-Felder liefert die DMF für Y:

$$Y = X1\ X2\ X3 \vee X1\ X3\ X4\ X5 \quad .$$

Entsprechend liefert die Blockbildung der 0-Felder die negierte DMF:

$$\overline{Y} = X1\ X3 \vee X1\ X2 \vee X1\ X3 \vee X1\ X5 \vee X1\ X4 \quad .$$

Hier bietet sich natürlich eine schaltungstechnische Realisierung der DMF an.

Mit Hilfe des Shannonschen Gesetzes ergeben sich die minimalen konjunktiven Gleichungen:

- negierte KMF

$$\overline{Y} = (X1 \vee X2 \vee X3) \wedge (X1 \vee X3 \vee X4 \vee X5) \quad ,$$

- KMF

$$Y = (X1 \vee X3) \wedge (X1 \vee \overline{X2}) \wedge (\overline{X1} \vee \overline{X3}) \wedge (\overline{X1} \vee \overline{X5}) \wedge (\overline{X1} \vee \overline{X4}) \quad .$$

Schaltungsminimierung a'la Quine-McCluskey

Vorgehensweise:

1. Stellen Sie die Funktion in disjunktiver Normalform dar.

2. Fassen Sie Terme der Form

$$(A \vee \neg x) \text{ und } (A \vee x)$$

in der DNF zusammen zu A.

Reduzieren Sie durch wiederholte Anwendung dieses Verfahrens sukzessive die Anzahl der Variablen in den einzelnen Teilausdrücken.

3. Streichen Sie unnötige reduzierte Terme.

Beispiel:

Betrachte die als DNF vorgelegte logische Funktion:

$$\begin{aligned} f(a,b,c,d) = {} & (a \vee b \vee c \vee \neg d) \wedge \\ & (a \vee b \vee \neg c \vee d) \wedge \\ & (a \vee b \vee \neg c \vee \neg d) \wedge \\ & (a \vee \neg b \vee c \vee \neg d) \wedge \\ & (a \vee \neg b \vee \neg c \vee \neg d) \wedge \\ & (\neg a \vee \neg b \vee c \vee d) \wedge \\ & (\neg a \vee \neg b \vee c \vee \neg d) \; . \end{aligned}$$

Tabellarische Auflistung der Vollkonjunktionen:

Zeile	Gruppe	Vollkonjunktion
1	1	$K_1 := a \vee b \vee c \vee \neg d$
2		$K_2 := a \vee b \vee \neg c \vee d$
3	2	$K_3 := a \vee b \vee \neg c \vee \neg d$
4		$K_4 := a \vee \neg b \vee c \vee \neg d$
5		$K_5 := \neg a \vee \neg b \vee c \vee d$
6	3	$K_6 := a \vee \neg b \vee \neg c \vee \neg d$
7		$K_7 := \neg a \vee \neg b \vee c \vee \neg d$

Die Basis für die obige Gruppeneinteilung bildet jeweils die Anzahl der negierten Variablen in den einzelnen Mintermen.

Betrachtet man nun benachbarte Gruppen, so unterscheiden sich die Vollkonjunktionen dadurch, daß eine Variable negiert und nichtnegiert auftritt, d.h. Zusammenfassungen in der Form

$$(A \vee \neg x) \wedge (A \vee x) = A \vee (\neg x \wedge x) = A \vee 1 = A$$

lassen sich mit Hilfe des Distributivgesetzes der Booleschen Algebra durchführen.

Beispielsweise liefert ein Zusammenfassen der Zeilen 1 und 3 der Tabelle:

$K_1 \wedge K_3 = (a \vee b \vee c \vee \neg d) \wedge (a \vee b \vee \neg c \vee \neg d) = a \vee b \vee \neg d$.

Sukzessives Zusammenfassen führt dann auf das Zwischenergebnis:

Zeile				
1 (1,3)	a	b		¬d
2 (4,6)	a	¬b		¬d
3 (1,4)	a		c	¬d
4 (3,6)	a		¬c	¬d
5 (2,3)	a	b	¬c	
6 (5,7)	¬a	¬b	c	
7 (4,7)		¬b	c	¬d

In dieser Tabelle geben die Zahlen in Klammern die Zeilennummern der Zeilen aus der ersten Tabelle an, die zusammengefaßt wurden. Dicke Linien trennen Ausdrücke mit unterschiedlichen Variablen; dünne Linien korrespondieren wiederum zu einer Gruppeneinteilung bezüglich der auftretenden negierten Variablen.

Weitere Zusammenfassungen lassen sich dann nur für Ausdrücke mit gleichem Variableninhalt wie folgt durchführen:

Zeile				
1 (1,2)	a			¬d
2 (3,4)	a			¬d
3 (5)	a	b	¬c	
4 (6)	¬a	¬b	c	
5 (7)		¬b	c	¬d

An dieser Stelle können keine weiteren Zusammenfassungen vorgenommen werden - die Zeilen 3 und 4 in der letzten Tabelle können nicht zusammengefaßt werden, da sie sich in mehreren Variablen bezüglich der Negierung unterscheiden!

Bei doppelt auftretenden Zeilen kann eine gestrichen werden (x w x = x).

Die derart erhaltenen Ausdrücke bezeichnet man als *reduzierte Terme* bzw. treffender als *Primimplikanten*; für das Beispiel ergibt sich:

$R_1 = a \vee \neg d$, $R_2 = a \vee b \vee \neg c$ $R_3 = \neg a \vee \neg b \vee c$, $R_4 = \neg b \vee c \vee \neg d$,

und es gilt:

$$f(a,b,c,d) = K_1 \text{ w } K_2 \text{ w } K_3 \text{ w } K_4 \text{ w } K_5 \text{ w } K_6 \text{ w } K_7$$

$$= R_1 \text{ w } R_2 \text{ w } R_3 \text{ w } R_4 \quad .$$

Die minimierte Form, die DMF der Funktion f ergibt sich nun über die Bestimmung einer geeigneten Teilmenge der Primimplikanten R_i (i=1,...,4), die zusammen alle Minterme K_i (i=1,...,9) überdecken. Das Streichen unnötiger reduzierter Terme läßt sich anhand der folgenden Tabelle bequem durchführen.

	K_1	K_2	K_3	K_4	K_5	K_6	K_7
R_1	*		*	*		*	
R_2		*	*				
R_3					*		*
R_4				*			*

Die matrixförmige Darstellung (2. Quinesche Tabelle) zeigt - angedeutet durch * -, welche Minterme durch die einzelnen Primimplikanten beschrieben werden. Hierzu schaut man sich den Aufbau des Primimplikanten an und alle Minterme mit gleicher Teilstruktur werden durch diesen ersetzt.

Beispielsweise ersetzt

$$R_4 = \neg b \vee c \vee \neg d$$

die Vollkonjunktionen

$$K_4 := a \vee \mathbf{\neg b \vee c \vee \neg d} \quad \text{und} \quad K_7 := \neg a \vee \mathbf{\neg b \vee c \vee \neg d} \ .$$

Insgesamt werden alle Paare (R_i,K_j) mit i=1,...,4 und j=1,...,7 darauf untersucht, ob der Primimplikant R_i in der Vollkonjunktion K_j enthalten ist, d.h. ob R_i durch Streichen von Variablen in K_j erhalten werden kann. Ist dies gegeben, so wird die zugehörige Kreuzungsstelle in der Tabelle mit einem * makiert.

Betrachtet man beispielsweise den reduzierten Term

$$R_1 = a \vee \neg d \ ,$$

so überdeckt R_1 die Vollkonjunktionen K_1, K_3, K_4, K_6, wobei die Variablen b und c eliminiert werden; die folgende Tabelle zeigt dies explizit:

R_1	a			¬d
K_1	**a**	b	c	**¬d**
K_2	**a**	b	¬c	d
K_3	**a**	b	¬c	**¬d**
K_4	**a**	¬b	c	**¬d**
K_5	¬a	¬b	c	d
K_6	**a**	¬b	¬c	**¬d**
K_7	¬a	¬b	c	**¬d**

und in der 2. Quineschen Tabelle ist in den entsprechenden Spalten der Zeile R_1 jeweils ein * einzutragen.

Eine Interpretation der 2. Quineschen Tabelle stellt sich nun wie folgt dar:

Befindet sich in der i-ten Spalte an den Schnittpunkten mit den Zeilen $j_1,...,j_p$ (p < 4) jeweils ein * , so impliziert dies, daß wenn der Wert von $K_i = 1$ ist, ebenfalls die Werte der reduzierten Terme $R_{j1},...,R_{jp}$ gleich 1 sind, d.h. einer dieser reduzierten Terme genügt, um die Vollkonjunktion K_i zu ersetzen - aufgrund der fehlenden Variablen nimmt dieser Primimplikant öfter den Wert 1 an als K_i.

Befindet sich umgekehrt in der i-ten Zeile an den Schnittpunkten mit den Spalten $j_1,...,j_p$ (p < 7) jeweils ein * , so impliziert dies

$$R_i \leq K_{j_1} \circ \ldots \circ K_{j_p}$$

,d.h. ist $R_i = 1$, so besitzt mindestens ein K_{jl} (l 0 {1,...,p}) den Wert 1.

Insgesamt bedeutet dies, daß eine Teilmenge der reduzierten Terme derart ausgewählt werden muß, daß in jeder Spalte immer wenigstens ein * berücksichtigt wird!

Befindet sich in einer Spalte nur ein * , so ist der korrespondierende reduzierte Term generell in die Lösung aufzunehmen. Dies bedeutet dann gleichfalls, daß auch alle Spalten, die in der zu diesem reduzierten Term gehörenden Zeile ein * aufweisen ebenfalls berücksichtigt sind und gestrichen werden können.

Allgemein gibt es für die Auswahl, der in die Lösung aufzunehmenden Primimplikaten, verschiedene Möglichkeiten!

Die Menge der Primimplikanten ist eindeutig bestimmt, die DMF kann dagegen äquivalent durch unterschiedliche Lösungen dargestellt werden.

Beim obigen Beispiel mit

	K_1	K_2	K_3	K_4	K_5	K_6	K_7
R_1	*		*	*		*	
R_2		*	*				
R_3					*		*
R_4				*			*

muß R_1 in die Lösung aufgenommen werden, da nur hiermit die Vollkonjunktionen K_1 und K_6 ersetzt werden können; R_2 überdeckt allein K_2 und R_3 ersetzt K_5 und K_7. R_4 wird für die Lösung nicht benötigt, da K_4 und K_7 bereits abgedeckt sind.

Als DMF folgt somit:

$$f(a,b,c,d) = R_1 \text{ w } R_2 \text{ w } R_3 \quad .$$

Die Vorteile des Minimierungsverfahrens von Quine-McCluskey sind:

(i) es handelt sich um ein algorithmisches Verfahren, das als Programm auf einem Rechner ausgeführt werden kann,

(ii) die Anzahl der logischen Variablen ist - im Gegensatz zum KV-Verfahren - beliebig.

6 Schaltnetze und Schaltwerke

6.1 Schaltnetze

6.2 Schaltwerke

6.1 Schaltnetze

Schaltnetz

(kombinatorische Logik, kombinatorische Schaltung)

* Kombination von Verknüpfungsgliedern ohne Speicherglieder

* die Werte der Ausgangsvariablen werden eindeutig durch die Eingangsvariablen festgelegt

* Beispiele

  - Code-Wandler,
  - Multiplexer, Demultiplexer,
  - Addierer,
  - . . . . . .

Code - Wandler

Aufbau eines Code-Wandlers, der den 8-4-2-1-Code in den Gray-Code überführt;

Zusatzbedingung:
Fehlermeldung bei Auftreten einer Pseudotetrade im 8-4-2-1-Code, der Wert der Ausgangsvariablen ist in diesem Fall beliebig.

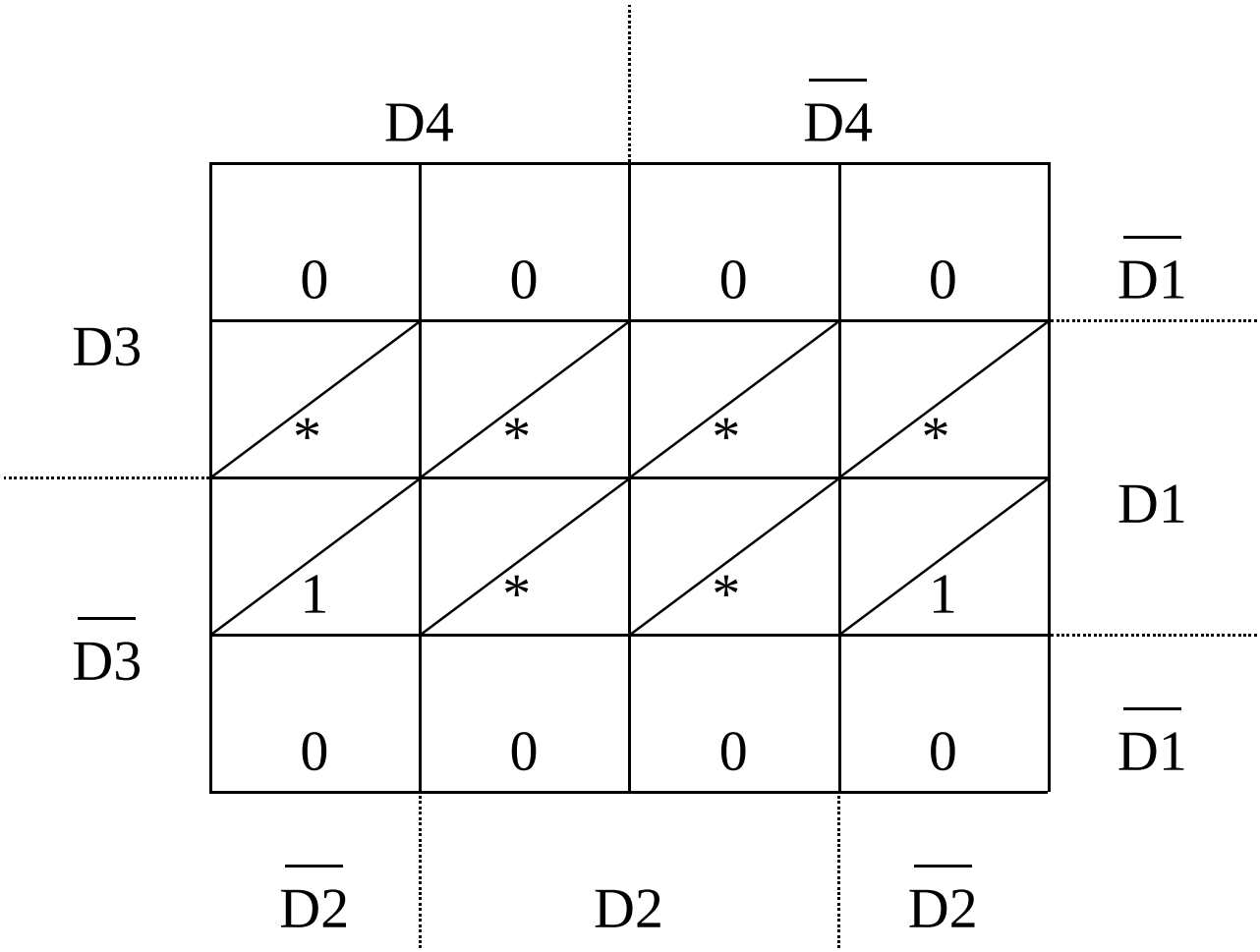
Der Code-Wandler stellt ein *Multi-Output-Schaltnetz* dar - die Minimierung wird für jede Ausgangsvariable separat durchgeführt.

	8-4-2-1-Code				Gray-Code				Feh-ler
dez.	D1	D2	D3	D4	G1	G2	G3	G4	F
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	0
2	0	0	1	0	0	0	1	1	0
3	0	0	1	1	0	0	1	0	0
4	0	1	0	0	0	1	1	0	0
5	0	1	0	1	0	1	1	1	0
6	0	1	1	0	0	1	0	1	0
7	0	1	1	1	0	1	0	0	0
8	1	0	0	0	1	1	0	0	0
9	1	0	0	1	1	1	0	1	0
Ps	1	0	1	0	*	*	*	*	1
eu	1	0	1	1	*	*	*	*	1
do	1	1	0	0	*	*	*	*	1
te	1	1	0	1	*	*	*	*	1
tra	1	1	1	0	*	*	*	*	1
den	1	1	1	1	*	*	*	*	1

È **È** **È**

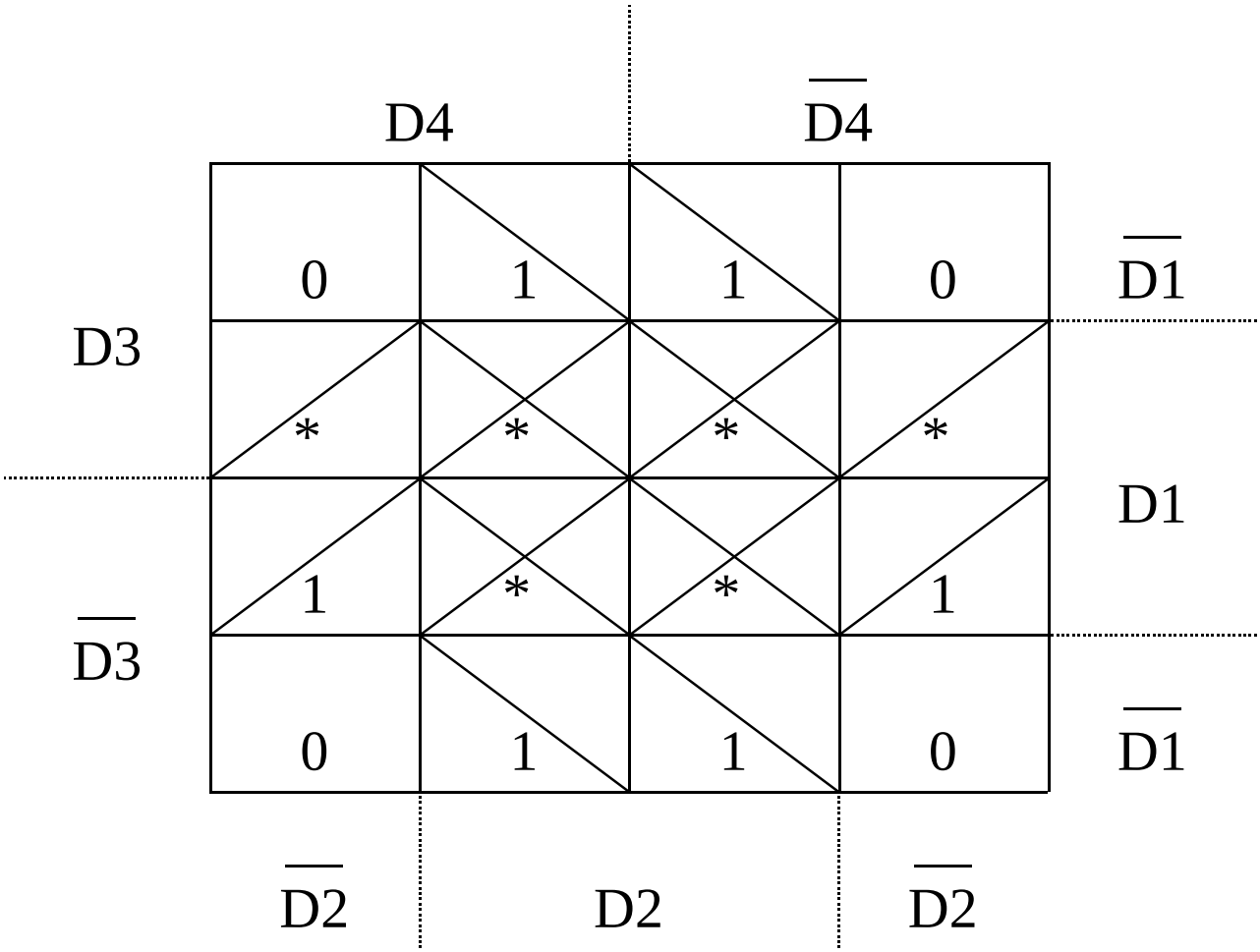
Eingangsvariablen Ausgangsvariablen Fehler-ausgang

G1:


D4
$\overline{D4}$
0 0 0 0
$\overline{D1}$
D3
* * * *
D1
1 * * 1
$\overline{D3}$
0 0 0 0
$\overline{D1}$
$\overline{D2}$
D2
$\overline{D2}$

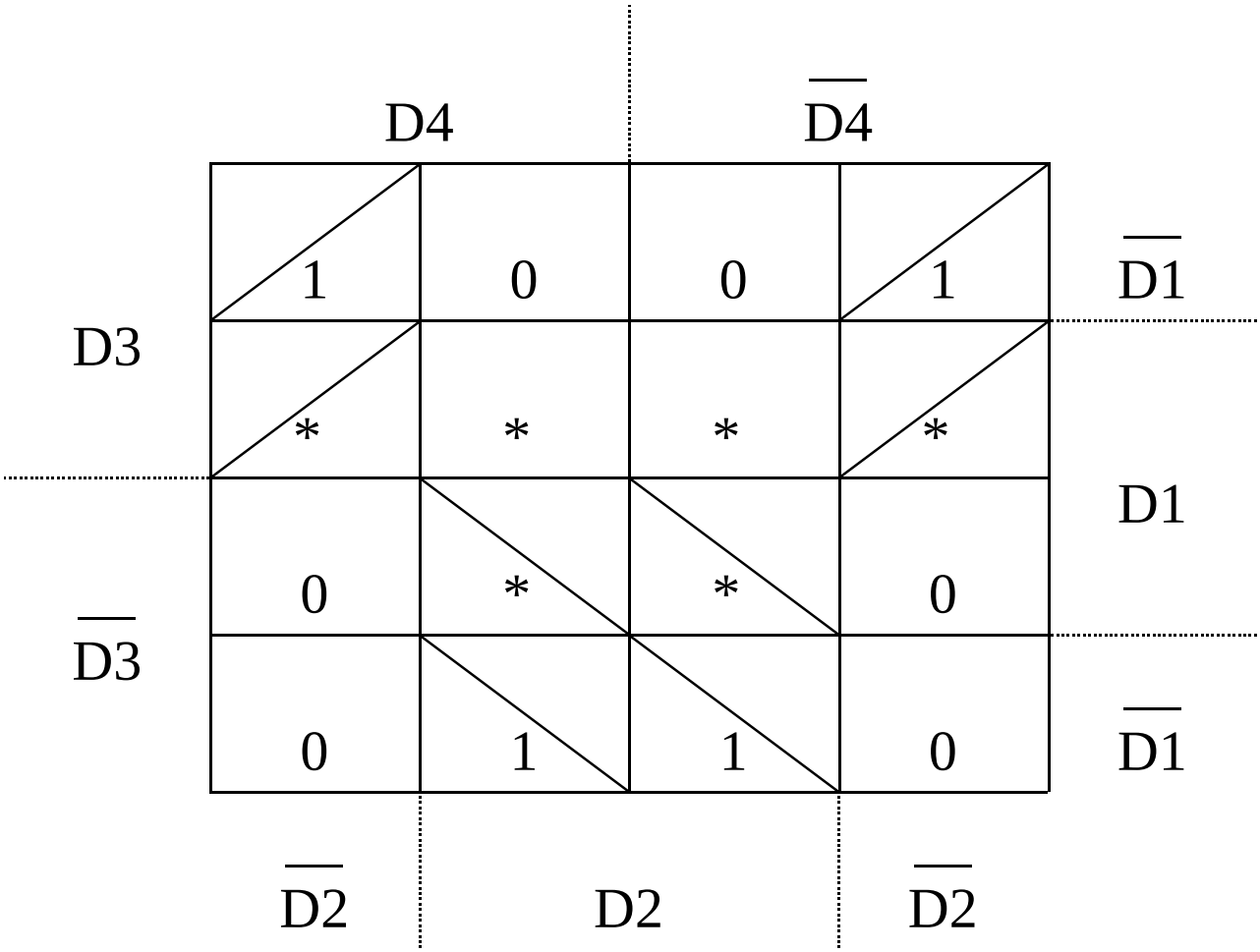
<u>G1 = D1</u>

G2:


D4
$\overline{D4}$
0
1
1
0
$\overline{D1}$
D3
*
*
*
*
D1
1
*
*
1
$\overline{D3}$
0
1
1
0
$\overline{D1}$
$\overline{D2}$
D2
$\overline{D2}$

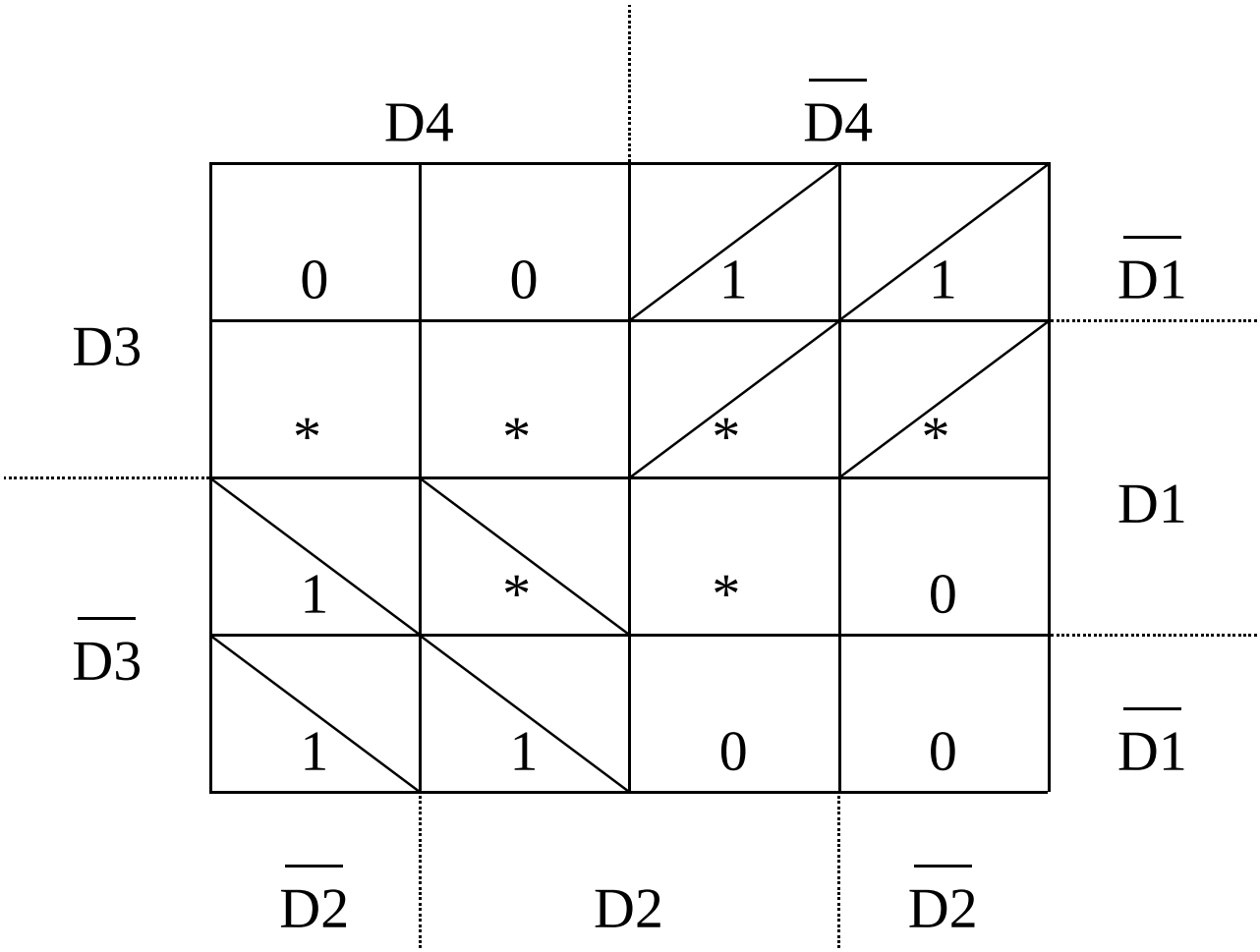
G2 = D1 w D2

G3:



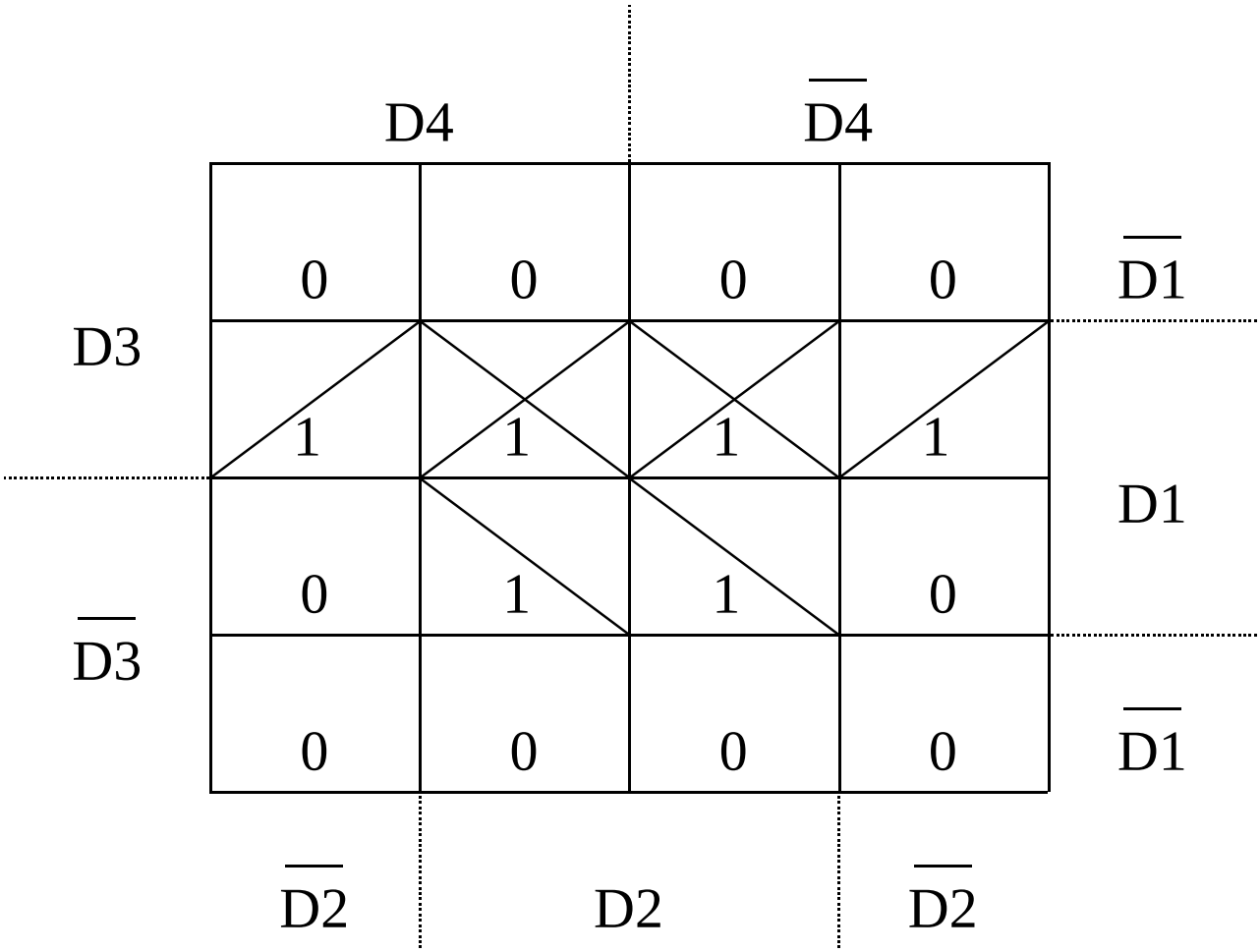
$G3 = D2\,\overline{D3}\ \vee\ \overline{D2}\,D3 = D2\ 7|6\ D3$

G4:



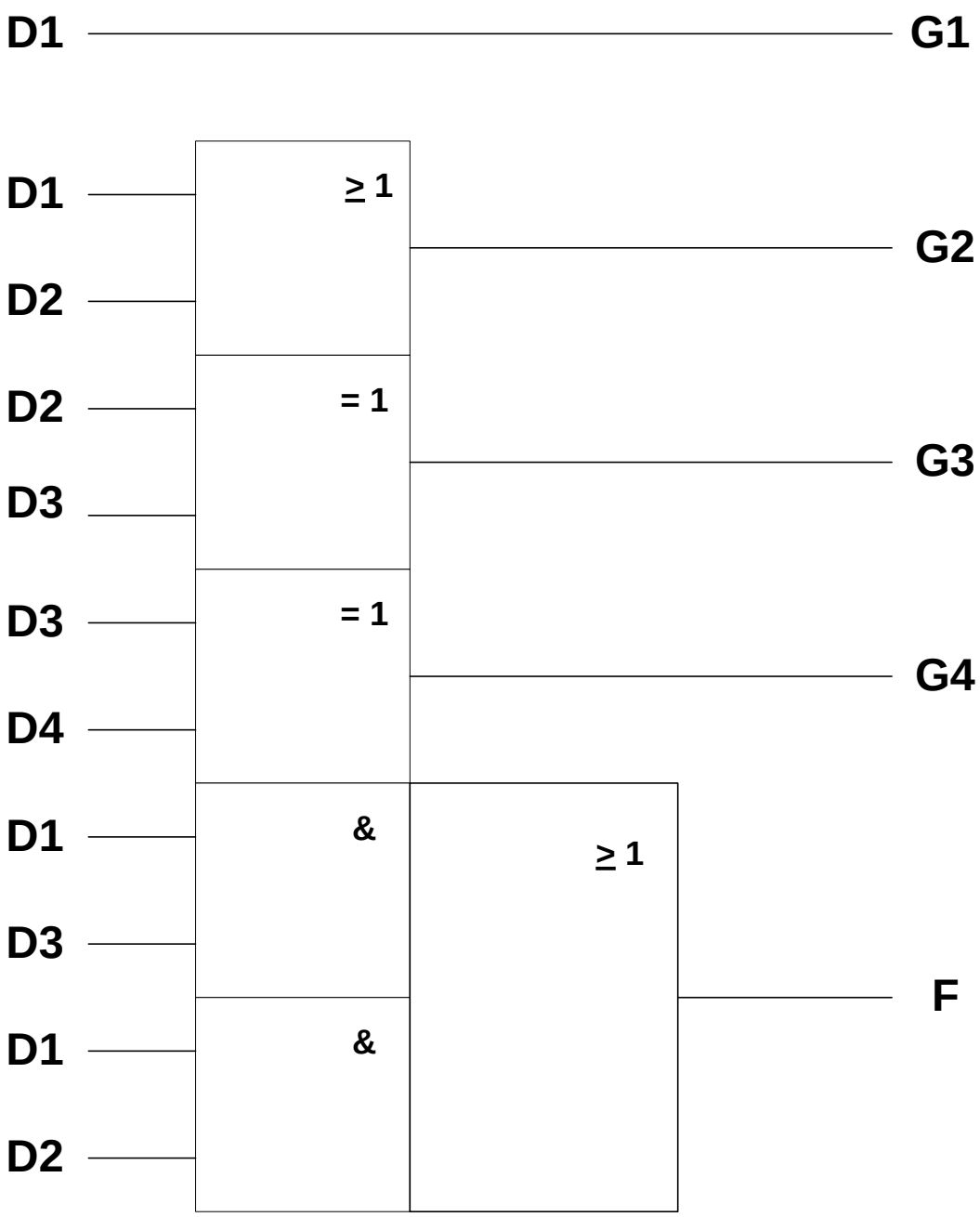
$G4 = D3\,\overline{D4}$ w $\overline{D3}\,D4 = D3$ 7|6 $D4$

F:

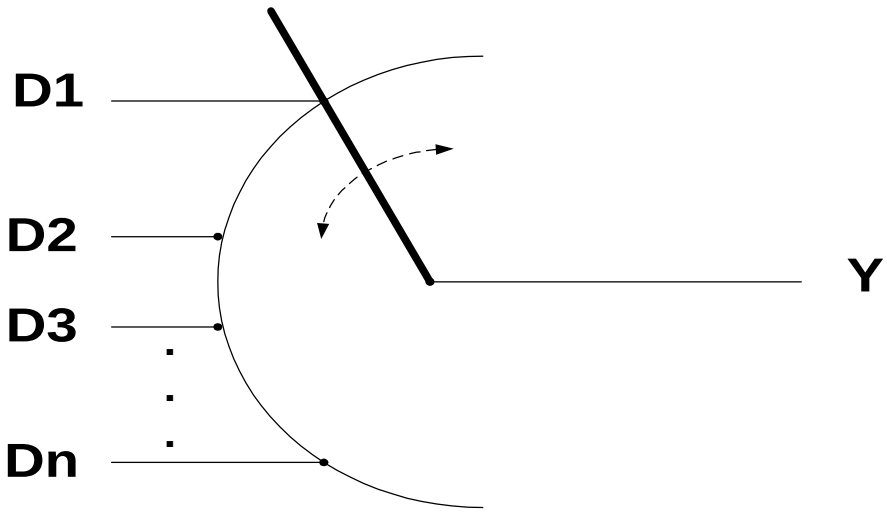


F = D1 D3 w D1 D2

Schaltung des Code-Wandlers:



Multiplexer



Selektion eines Datenkanals; die n Datenkanäle werden über ld n Steuervariablen codiert.

Entwurf eines 4-zu-1-Multiplexers:

~ Anzahl der Steuervariablen: ld 4 = 2

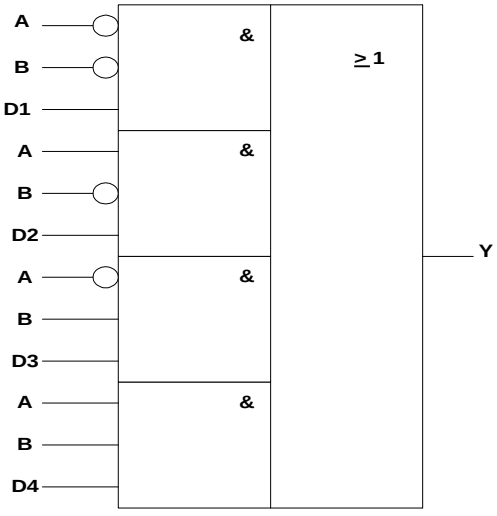
~ Wahrheitstafel

	Steuervariablen		**selektierter Datenkanal**
Adresse	B	A	Y
0	0	0	D1
1	0	1	D2
2	1	0	D3
3	1	1	D4

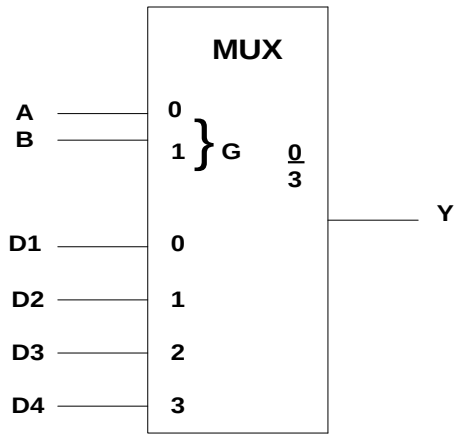
Logische Funktion:

$$Y = \overline{A}\,\overline{B}D1 \vee A\overline{B}D2 \vee \overline{A}BD3 \vee ABD4$$

Schaltung

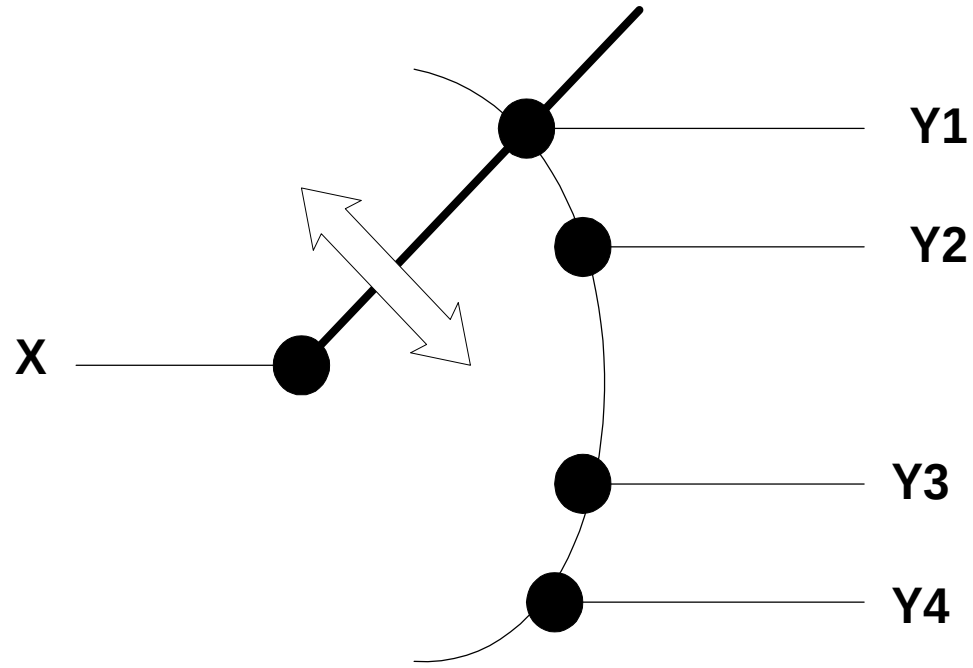


Schaltsymbol



Demultiplexer

Selektion eines Ausgangskanals



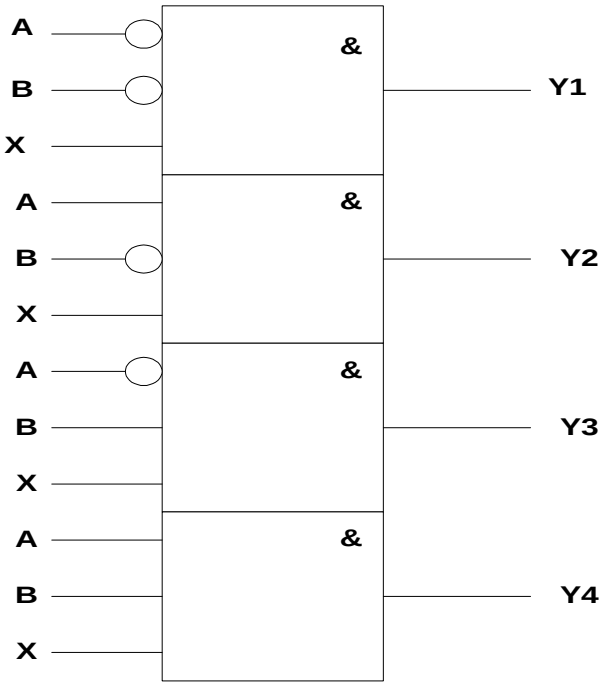
Entwurf eines 1-zu-4-Demultiplexers:
4 Ausgänge erfordern 2 Steuervariablen

Adresse	B	A	Y1	Y2	Y3	Y4
0	0	0	X	0	0	0
1	0	1	0	X	0	0
2	1	0	0	0	X	0
3	1	1	0	0	0	X

X bezeichnet den Dateneingang

$$Y1 \text{ ' } \overline{A}\overline{B}X \quad , \quad Y2 \text{ ' } A\overline{B}X \, ,$$

$$Y3 \text{ ' } \overline{A}BX \quad , \quad Y4 \text{ ' } ABX$$



Addierer

Basis: 1-Bit-Halbaddierer

Der 1-Bit-Halbaddierer liefert die Summe und den Übertrag (Carry) für die einstellige Addition zweier Dualzahlen.

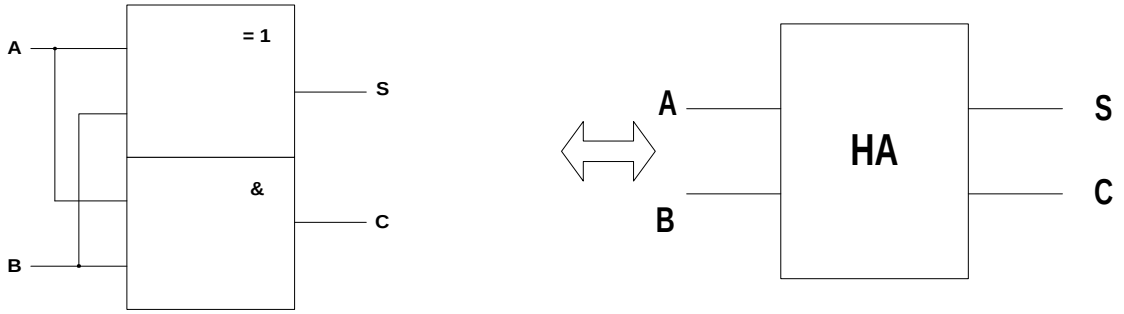
Wahrheitstafel:

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

KNF:

$$S \;'\; \bar{A}B \;\; \mathtt{w} \;\; A\bar{B} \;'\; A \;\; 7|6 \;\; B \;,$$

$$C \;'\; AB$$



Ein Volladdierer ist eine Schaltung, die den bei der Addition höherwertiger Stellen auftretenden Übertrag der vorherigen Stelle erfasst.

1-Bit-Volladdierer:

A_i	B_i	C_{i-1}	S_i	C_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

i = 1 , 2 , 3 , . . . indiziert die zu addierende Dualstelle;

i = 1 korrespondiert zum LSB.

Logische Funktionen für i = 1:

$$S_1 = C_0 \bar{A}_1 \bar{B}_1 \vee \bar{C}_0 \bar{A}_1 B_1 \vee C_0 A_1 \bar{B}_1 \vee C_0 A_1 B_1$$

$$= C_0 \vee (\bar{A}_1 \bar{B}_1 \vee A_1 B_1) \vee \bar{C}_0 \vee (\bar{A}_1 \bar{B}_1 \vee A_1 B_1)$$

$$= C_0 \vee (A_1 \equiv B_1) \vee \bar{C}_0 \vee (A_1 \not\equiv B_1)$$

$$= C_0 \vee \overline{(A_1 \not\equiv B_1)} \vee \bar{C}_0 \vee (A_1 \not\equiv B_1)$$

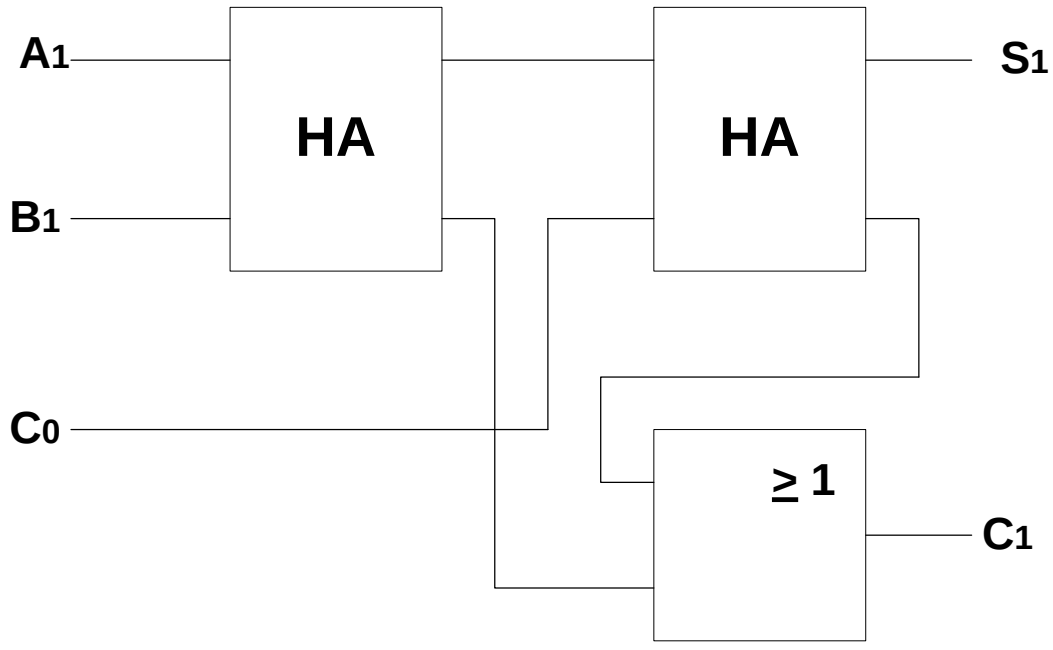
$$= \underline{\underline{C_0 \not\equiv (A_1 \not\equiv B_1)}}$$

$$C_1 \Leftrightarrow C_0 \bar{A}_1 B_1 \vee C_0 A_1 \bar{B}_1 \vee \bar{C}_0 A_1 B_1 \vee C_0 A_1 B_1$$

$$\Leftrightarrow (C_0 \vee (\bar{A}_1 B_1 \vee A_1 \bar{B}_1)) \vee (A_1 B_1 \vee (\bar{C}_0 \vee C_0))$$

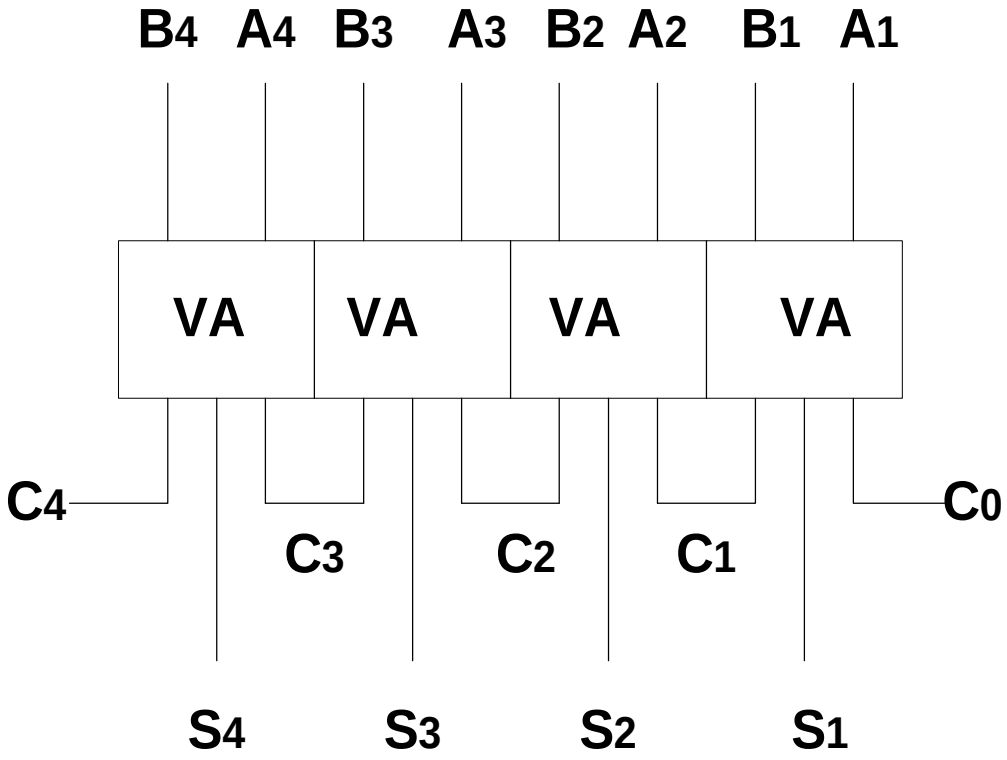
$$\Leftrightarrow (C_0 \vee (A_1 \veebar B_1)) \vee (A_1 B_1)$$

Schaltungstechnisch wird der 1-Bit-Volladdierer mit zwei Halbaddierern und einem ODER-Gatter realisiert:



Die Addition mehrstelliger Dualzahlen ergibt sich über eine serielle Verknüpfung mehrerer Volladdierer.

4-Bit-Volladdierer:



Es treten erhebliche Verzögerungszeiten auf, da der Übertrag der vorherigen Stelle berechnet werden muß.

Addierer, die den Übertrag im voraus ermitteln und damit eine wesentlich geringere Verzögerungszeit liefern, heißen *Carry Look Ahead - Addierer*.

6.2 Schaltwerke

Schaltwerk

(sequentielle Logik, sequentielle Schaltung)

* Funktionseinheit deren Ausgangsvariablen von den Eingangsvariablen und vom inneren Zustand der Einheit abhängt

* der Ausgangswert zu einem bestimmten Zeitpunkt wird durch die Eingangswerte zu diesem und endlich vielen vorangegangenen Zeitpunkten (Takten) festgelegt

* Beispiele

  ˜ Monostabile Kippstufen (Monoflops),

  ˜ Bistabile Kippstufen (Flipflops),

  ˜ Zähler

  ˜ . . . . . .

Bistabile Kippstufen (Flipflops)

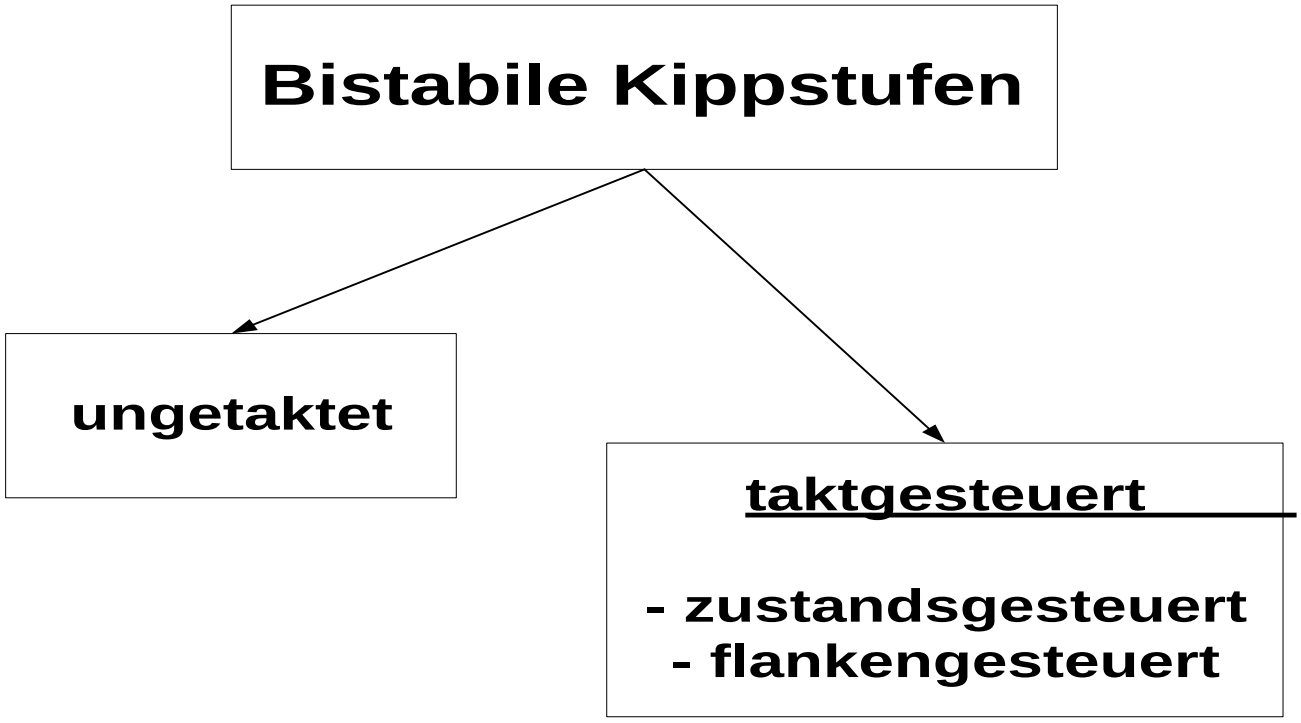
Charakteristisch sind zwei stabile Ausgangszustände;

ein FF (Flipflop) speichert die Informationseinheit 1 Bit:

FF gesetzt: ‘1' wird gespeichert,

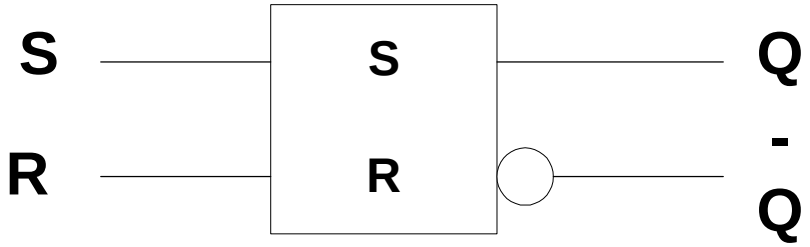
FF rückgesetzt: ‘0' wird gespeichert.

Einteilung der Flipflops



Ungetaktetes RS - FF

(RS - Basis -FF, RS - Latch)



S: Setzeingang (Set), S = 1 ~> setzen,

R: Rücksetzeingang (Reset), S = 0 ~> löschen,

Q: Ausgang,

$\overline{Q}$: komplementärer Ausgang.

Für S = 0 und R = 0 speichert das FF den Logik-Zustand:

RS - FF gesetzt: $Q = 1$, $\overline{Q} = 0$

RS - FF rückgesetzt: $Q = 0$, $\overline{Q} = 1$.

Entwurf des RS - FF

Voraussetzung (Annahme):

Gleichzeitiges *Setzen* (S = 1) und
Rücksetzen (R = 1) tritt nicht auf,

S = 1 und R = 1 ist eine “verbotene” Kombination der Eingangsvariablen!

S = 1 und R = 1 korrespondiert zu einem redundanten Term, der bei der Minimierung verwendet wird:

Wahrheitstafel

S	R	Q^m	Q^{m+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	*
1	1	1	*

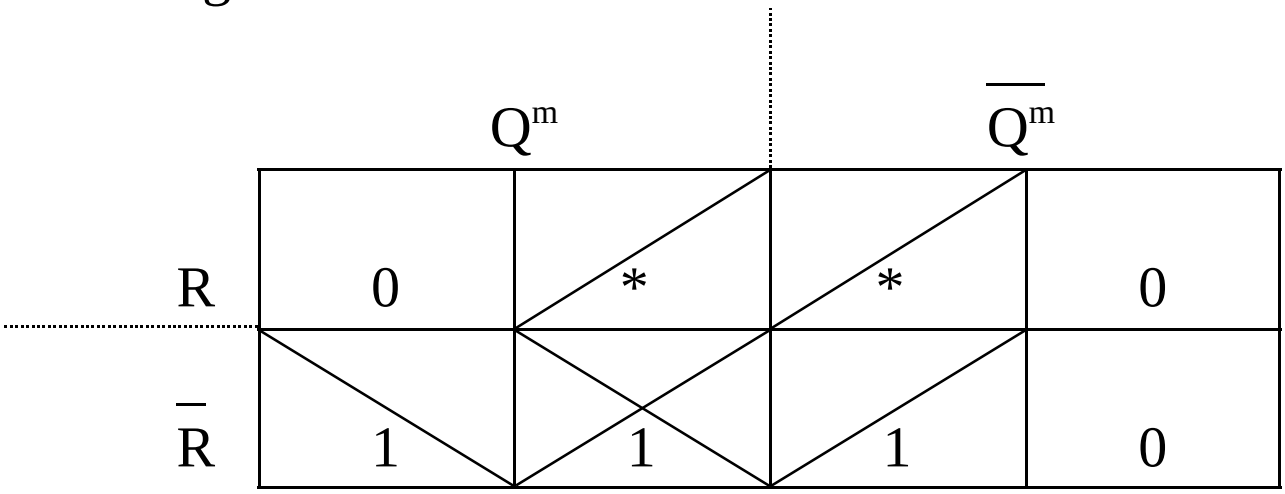
Beachte

Unterscheidung zwischen Aus- und Eingangsgrößen:

Zeitpunkt t^m : Q^m (am Eingang),

Zeitpunkt t^{m+1} : Q^{m+1} (am Ausgang).

KV -Diagramm:



Übergangsbedingung:

$$Q^{m+1} = S \vee \overline{R}\, Q^m$$

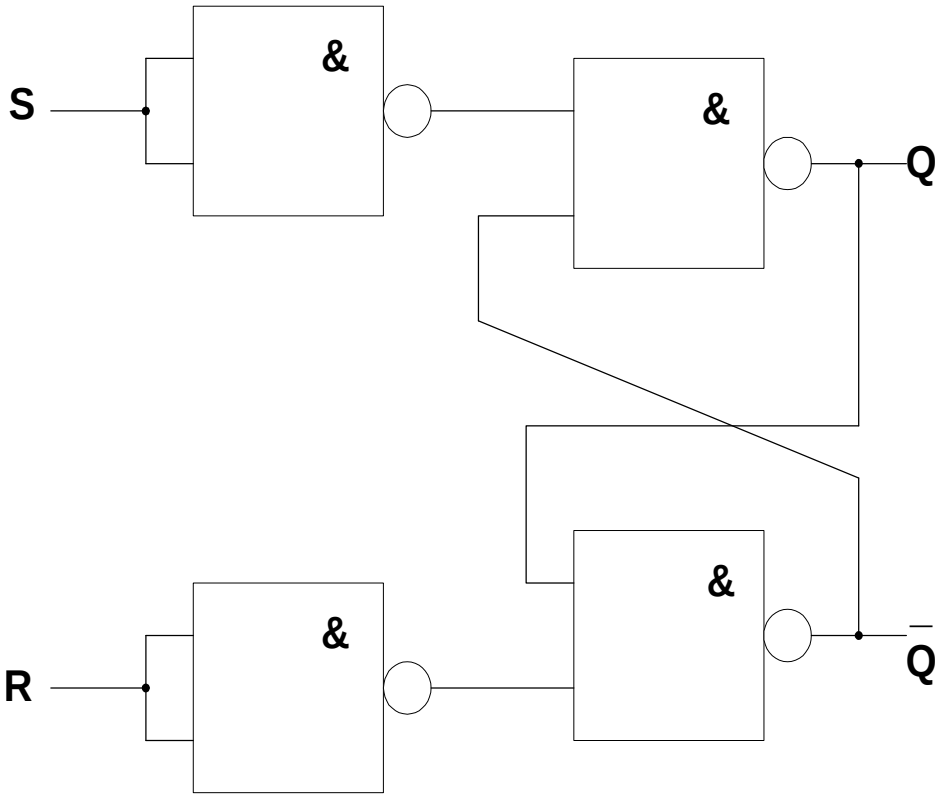
<u>Bemerkung:</u>
Wesentlich sind die Rückkopplungen der Ausgänge auf die Eingänge: die auf den Eingang rückgekoppelten Ausgangszustände werden mit den Eingangsvariablen verknüpft und bestimmen die neuen Ausgangszustände, die sich wiederum auf den Eingang auswirken!

Schaltungstechnische Realisierung:

Shannonsches Gesetz »

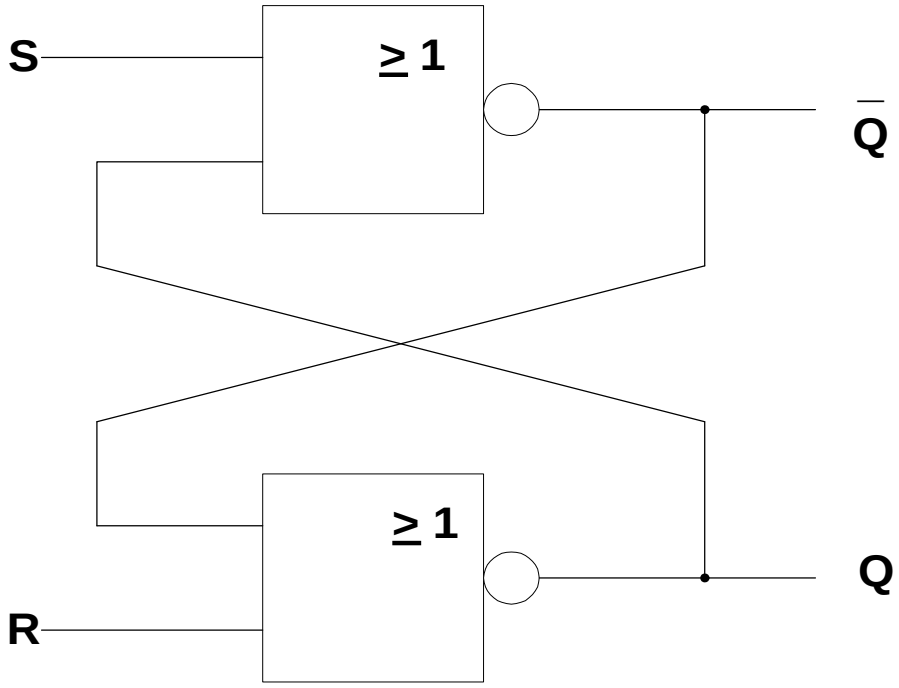
* NAND - Technik

$$Q^{m+1} = \overline{\overline{S} \vee \overline{R}\, Q^{m}}$$



* NOR - Technik

$$\overline{Q^{m\%1}} \quad ' \quad \overline{S \quad \mathbf{w} \quad \overline{R \ \mathbf{w} \ \overline{Q}^{m}}}$$



Beachte:

Für die “verbotene” Eingangskombination S = 1 und R = 1 ist der resultierende Ausgangszustand von der verwendeten Schaltungstechnik abhängig:

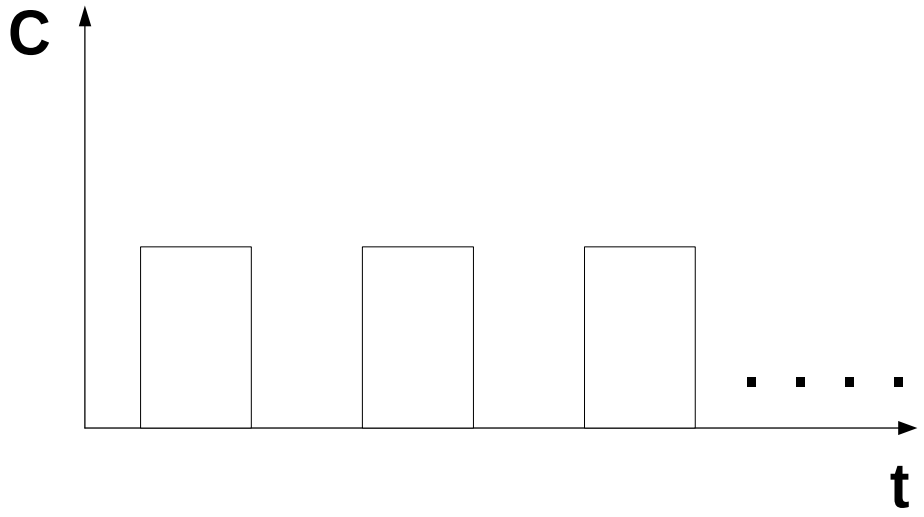
NAND-Technik (S = R = 1) Y Q = $\overline{Q}$ = 1,

NOR-Technik (S = R = 1) Y Q = $\overline{Q}$ = 0.

Zustandgesteuertes RS - Flipflop
(einzustandgesteuertes Flipflop)

Erweiterung des ungetakteten RS - FF um einen Takteingang C

Taktsignal (Clock):



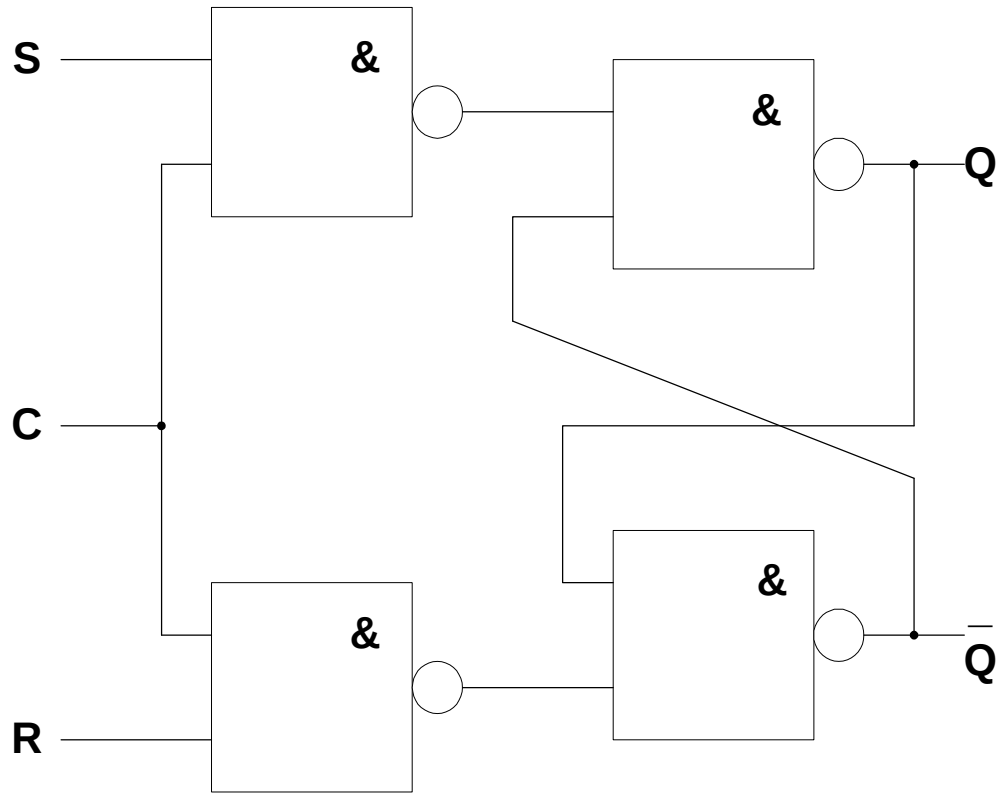
Die Eingangssignale wirken sich nicht direkt auf den Zustand des FF aus, sondern nur in Verbindung mit dem Takt C !

Beide Eingangssignale werden konjunktiv mit dem Taktsignal verknüpft,
Übergangsbedingung:

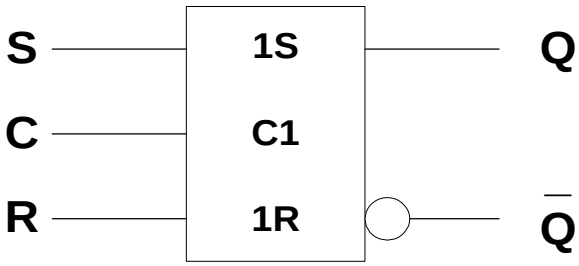
$$Q^{m+1} = (S \vee C) \wedge (\overline{R \vee C} \vee Q^{m})$$

Der Ausgangszustand kann sich nur für C = 1 ändern; geht das Taktsignal in den Logik-Zustand 0 über, speichert das Flipflop den gegebenen Ausgangswert.

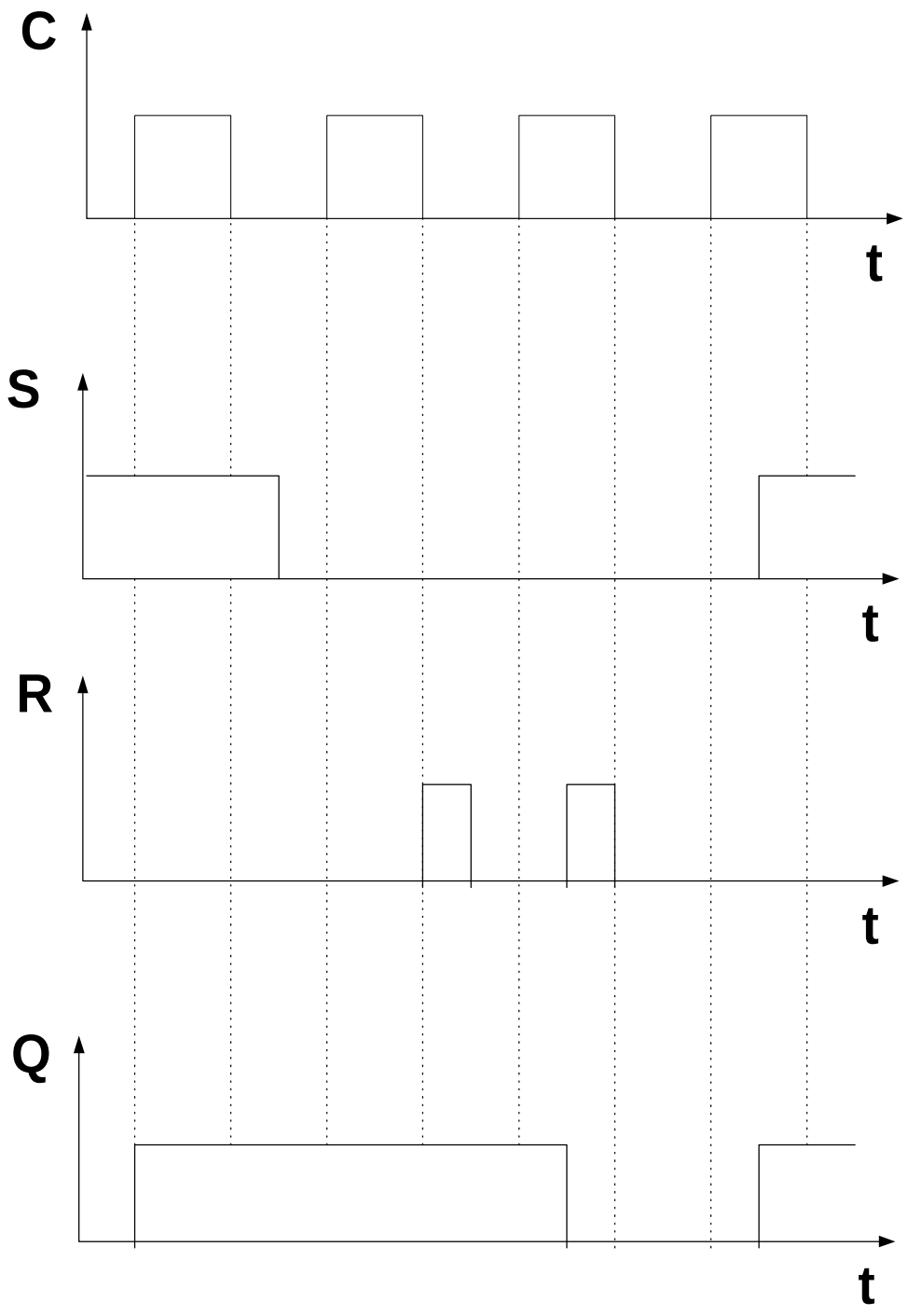
Zustandsgesteuertes RS - FF in NAND - Technik:



Schaltsymbol:

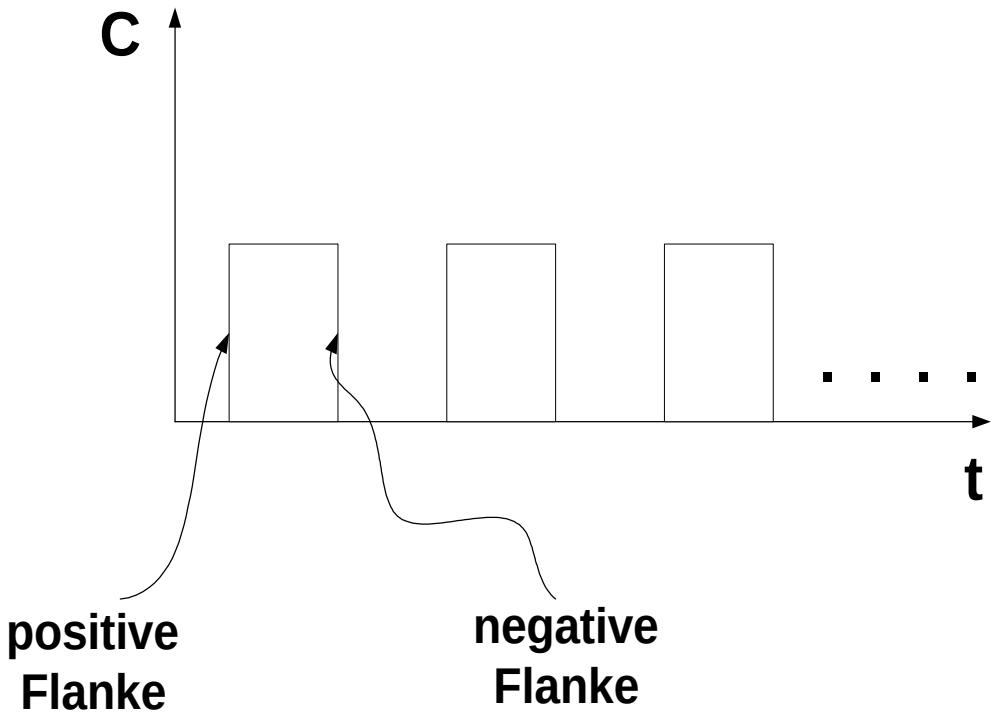


Signalzeitplan des zustandsgesteuerten RS - FF:



Flankengesteuertes RS - Flipflop

Einflankengesteuerte FF sind getaktete FF, die mit der positiven oder negativen Taktflanke gesetzt bzw. rückgesetzt werden; der Ausgangszustand kann sich nur mit der schaltenden Flanke ändern.



Es liegt die Übergangsbedingung des ungetakteten RS -FF vor, jedoch ist es hier die schaltende Flanke des Taktes C, die das FF setzt bzw. zurücksetzt.

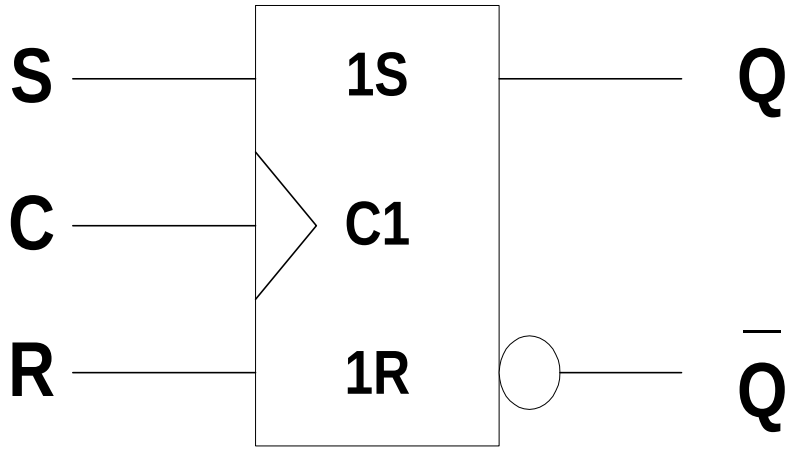
Übergangsbedingung:

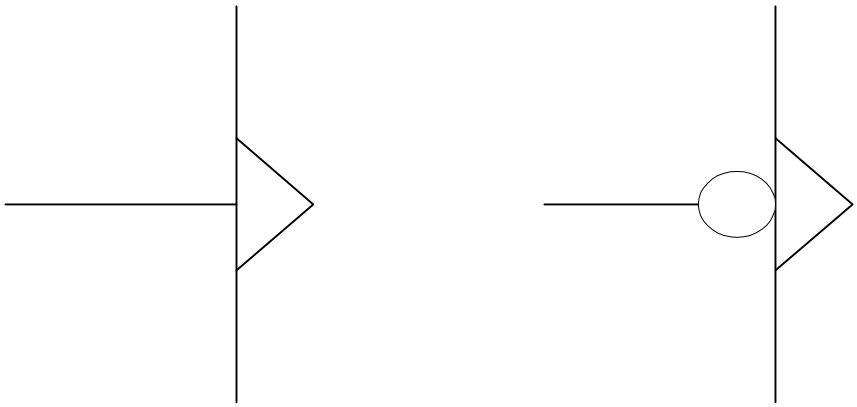
$$Q^{m+1} = S \vee \bar{R}\, Q^{m}$$

Pegeltabelle (positiv flankengesteuertes RS - FF):

S	R	C	Q^{m+1}
*	*	L	Q^m
*	*	H	Q^m
L	L	↓	Q^m
H	L	↓	H
L	H	↓	L
H	H	*	*

Schaltsymbol:





L -> H **H -> L**

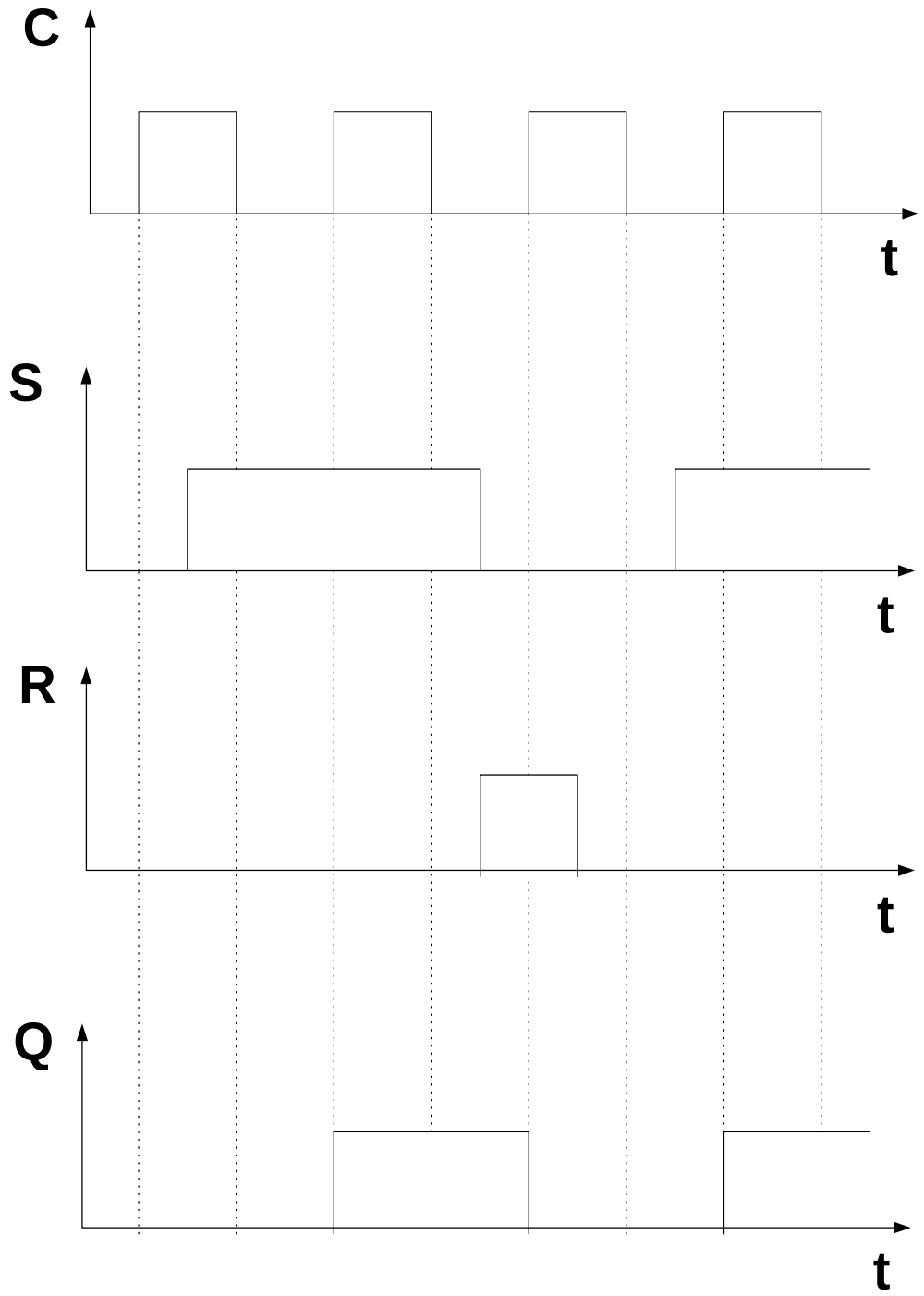
positiv **negativ**

flankengesteuerte Takteingänge

Der Takt benötigt eine endliche Zeit um von L auf H oder umgekehrt zu wechseln.

Technisch erfolgt eine genauere Steuerung, wenn die Signale nur während der positiven oder negativen Flanke abgefragt werden.

Signalzeitplan:



Flankengesteuertes JK - Flipflop

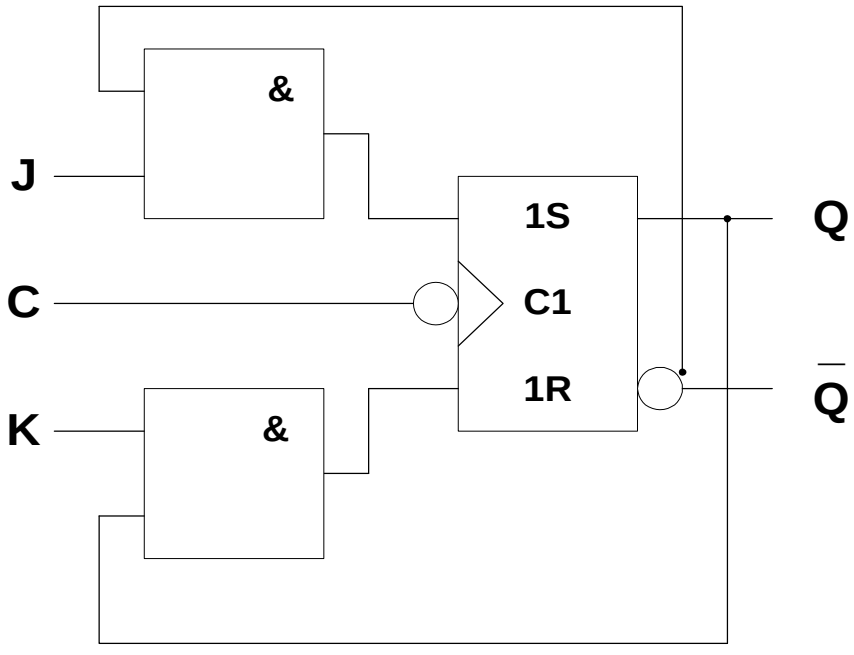
Das Setzen eines Speicherelements, wenn es bereits im Zustand 1 ist, bzw. das Löschen des FF, wenn es zurückgesetzt ist, ist überflüssig!

Forderung:

Setzen] $\overline{Q} = 1$,

Löschen] $Q = 1$.

Das flankengesteuerte JK - FF folgt mit zusätzlichen Rückkopplungen und zwei Eingangs-UND-Gatter aus dem flankengesteuerten RS - FF.



Pegeltabelle				Wahrheitstabelle				
J	K	C	Q^{m+1}		J	K	Q^m	Q^{m+1}
*	*	L	Q^m	0	0	0	0	0
*	*	H	Q^m	1	0	0	1	1
L	L	9	Q^m	2	0	1	0	0
L	H	9	L	3	0	1	1	0
H	L	9	H	4	1	0	0	1
H	H	9	$\overline{Q^m}$	5	1	0	1	1
				6	1	1	0	1
				7	1	1	1	0

Übergangsbedingung:

DNF:

$$Q^{m+1} = \bar{J}\,\bar{K}\,Q^m \vee J\,\bar{K}\,\overline{Q^m}$$

$$\vee\ J\,\bar{K}\,Q^m \vee J\,K\,\overline{Q^m}$$

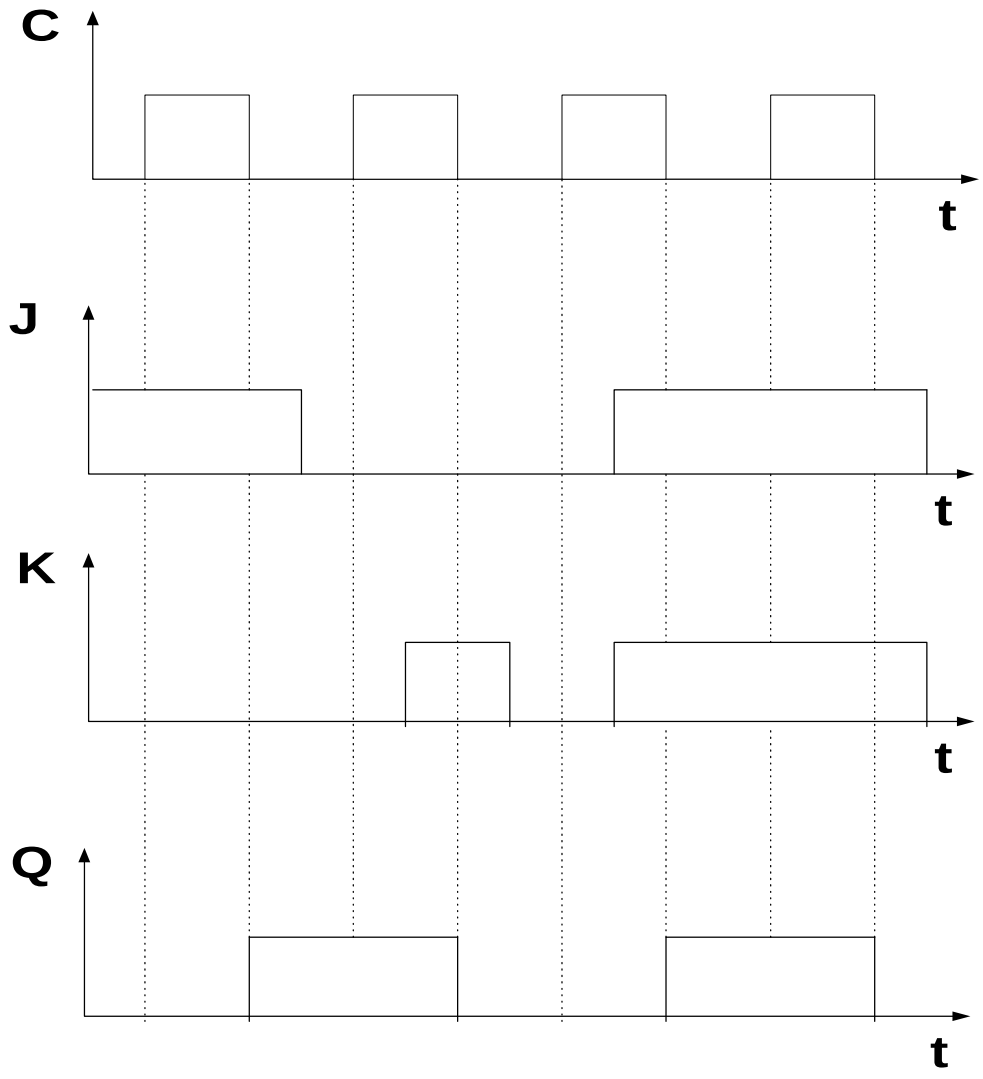
DMF:

$$Q^{m+1} = J\,\overline{Q^m} \vee \bar{K}\,Q^m$$

Beachte:

Der beim RS - FF auftretende Fall S = R = 1 wird hier umgangen; aufgrund der gekreuzten Rückkopplung wird für J = K = 1 mit jeder negativen Taktflanke der Wert der Ausgangsvariablen invertiert.

Signalzeitplan (Anfangszustand Q = 0):



Master - Slave - Prinzip

In der Praxis beim Aufbau von Zählern und Schieberegistern sind transparente FF ungeeignet;

benötigt werden FF, die den Eingangszustand *zwischenspeichern* und diesen erst an den Ausgang übertragen, wenn die Eingänge bereits wieder *verriegelt* sind.

Lösung: **Kombination zweier Flipflops**

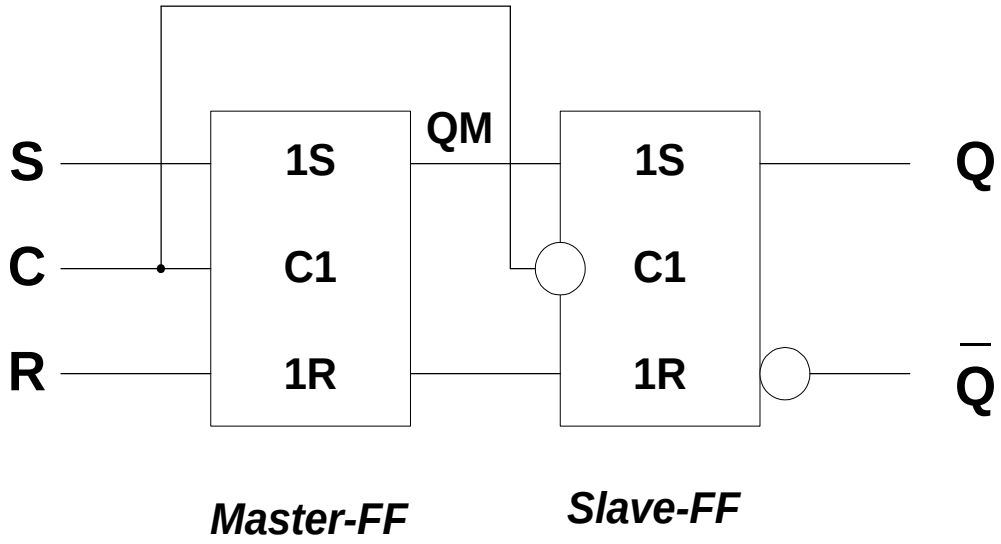
» *Master - FF* am Eingang

» *Slave - FF* am Ausgang

Master - Slave - Prinzip:

Zu keinem Zeitpunkt ist der Eingang direkt mit dem Ausgang verbunden!

Zweizustandsgesteuertes RS - Flipflop



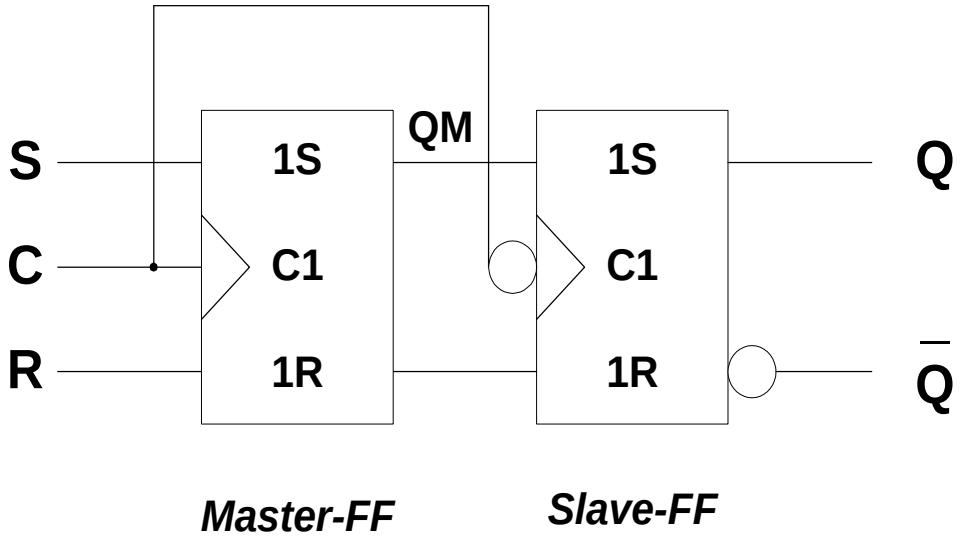
Das Master - FF ist für C = 1 transparent - die Übernahme in das Eingangs - FF erfolgt wie beim zustandsgesteuerten RS - FF.

Beim Übergang des Taktes in den Logik-Zustand 0 wird das Slave - FF gesetzt bzw. rückgesetzt.

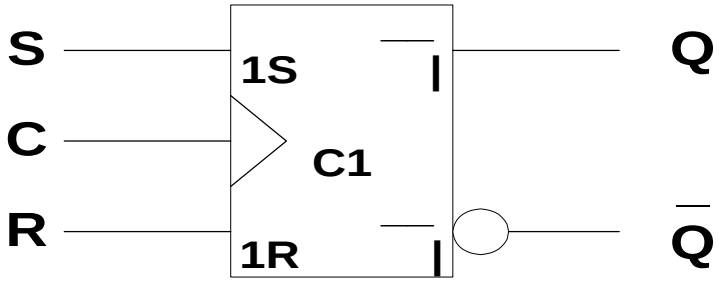
Zweiflankengesteuerte Flipflops

Das Eingangs - FF (Master - FF) übernimmt die Information mit der positiven Taktflanke und gibt diese mit der negativen an das Ausgangs - FF (Slave - FF) weiter.

Zweiflankengesteuertes RS - FF:



Schaltsymbol:



¬ kennzeichnet einen retardierten Ausgang.

7 Endliche Automaten

7.1 Deterministische endliche Automaten

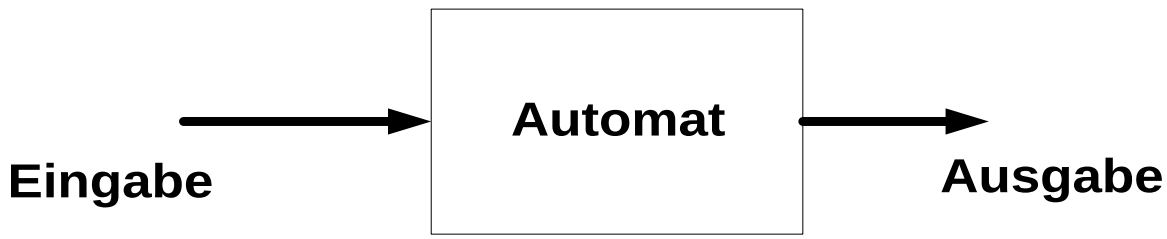
7.2 Nichtdeterministische endliche Automaten

7.3 Endliche Automaten mit g-Übergängen

7.1 Deterministische endliche Automaten

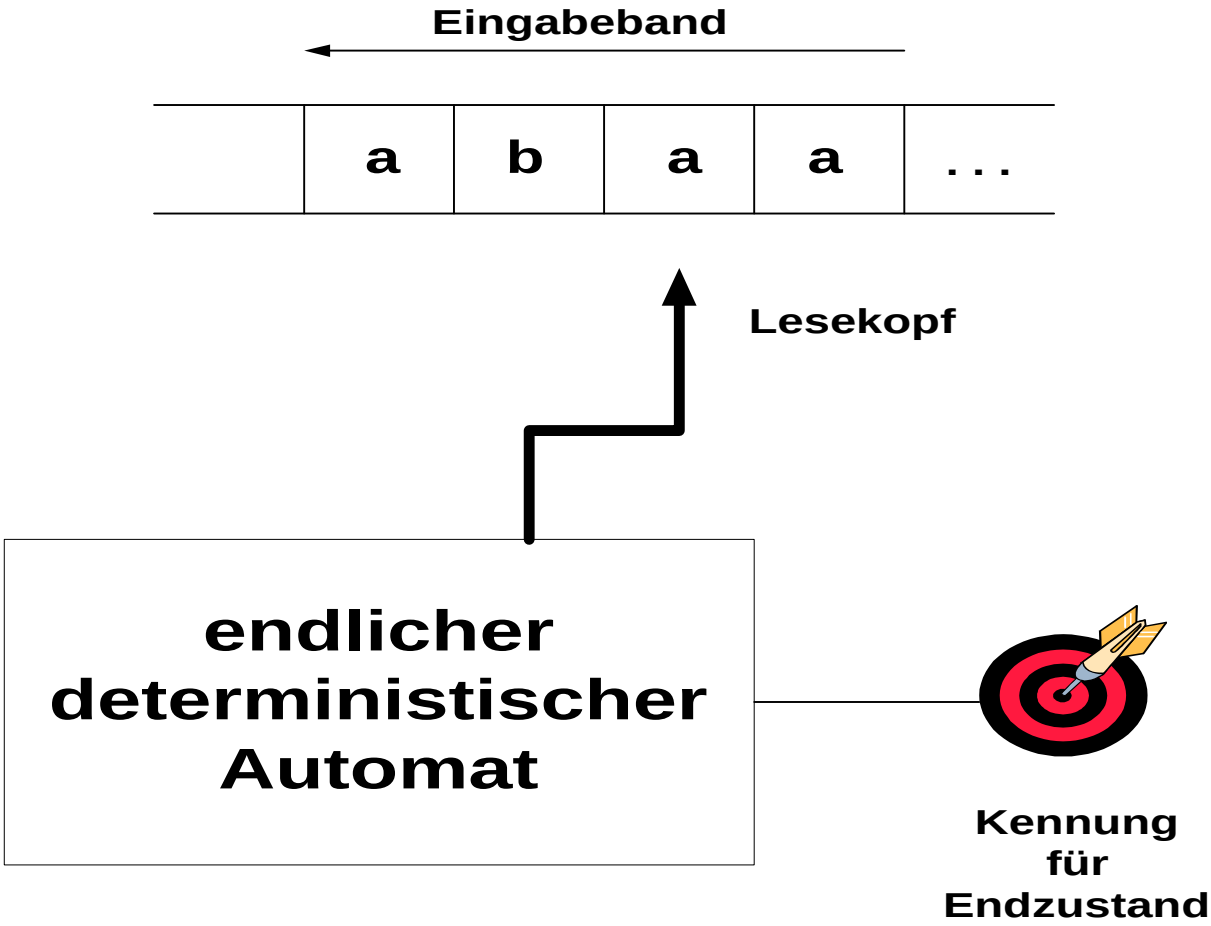
Automaten des Alltags:

- Kaffeemaschine
- Getränkeautomat
- Waschautomat
- Geldautomat
- . . .



Endlicher deterministischer Automat

- abstraktes Modell
- endlich viele Zustände
- es existiert jeweils genau ein Folgezustand
- Akzeptor für *reguläre Sprachen*.



Beispiel: *Schwimmbadautomat* A_{swim}

Automat soll den Eintritt in ein Schwimmbad kontrollieren:

Eintrittspreis: 2 DM,

Automat akzeptiert:

50-Pfennig-, 1 DM-, 2 DM- Münzen ,

Endzustand:

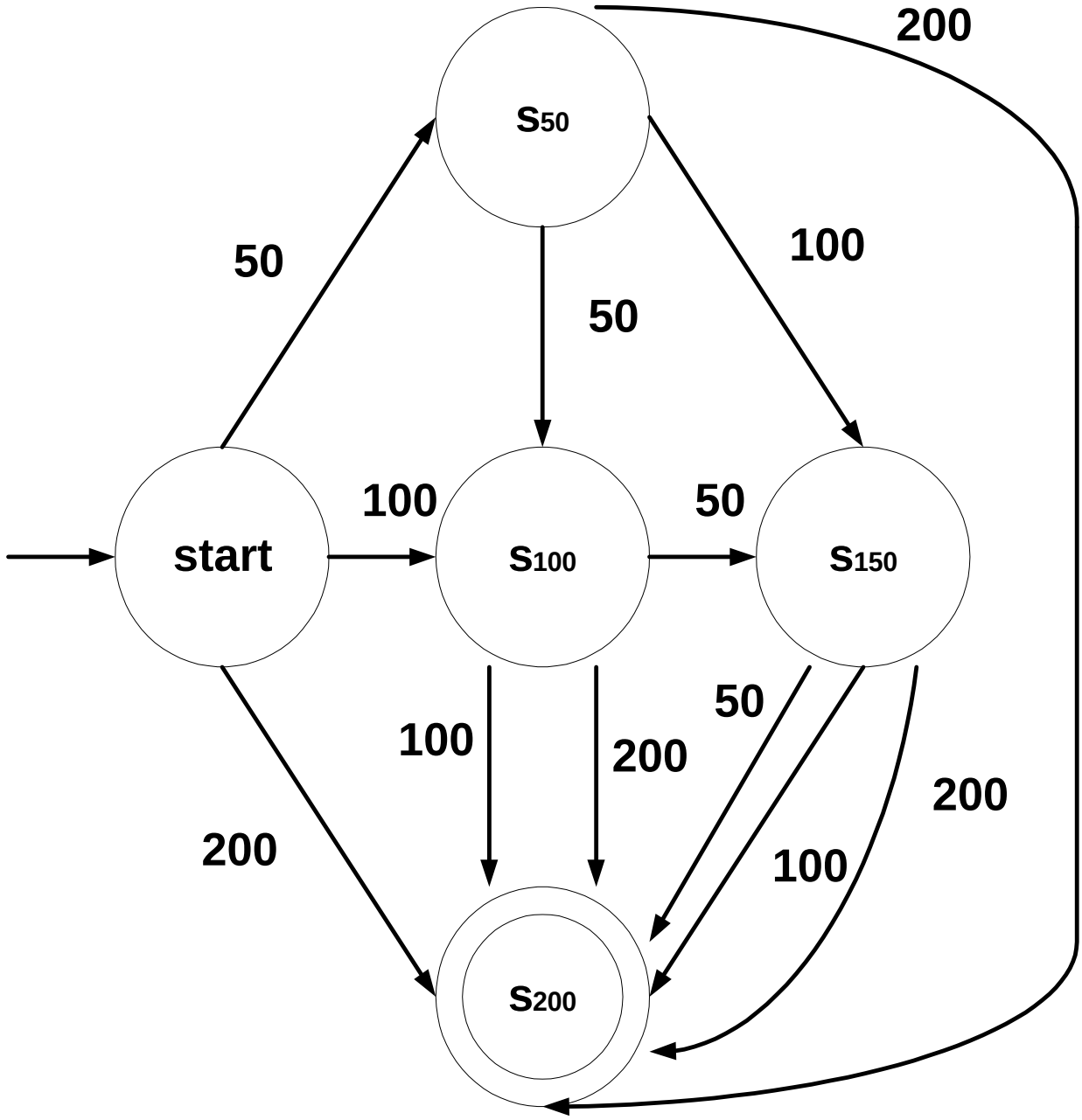
nach Eingabe eines Gesamtwertes von mindestens 2 DM öffnet der Automat eine Barriere,

Nebenbedingung:

Automat akzeptiert Überbezahlungen.

Zustandsdiagramm des Schwimmbadautomaten

Graphische Darstellung des Automaten durch einen *gerichteten markierten Graphen*.



Zustandsdiagramm (Zustandsgraph)
gerichteter markierter Graph, wobei
- die Zustände die Knoten (Kreise) sind,
- der Knoten des Startzustandes wird durch einen hineinweisenden Pfeil besonders gekennzeichnet,
- alle Endzustände werden durch doppelte Kreise dargestellt,
- die beschrifteten Kanten bezeichnen die Übergänge zwischen den Zuständen.

Eingabealphabet: $G = \{ 50 , 100 , 200 \}$

Sprache des Automaten:
Die Menge aller Eingabefolgen, die der Automat akzeptiert, d.h. die den Automaten in den Endzustand überführen, heißt die *Sprache* des Automaten:

$L (A_{swim}) = \{ 50\ 50\ 50\ 50 , 50\ 50\ 50\ 100 , 50\ 50\ 50\ 200 , 50\ 50\ 100 , 50\ 50\ 200 , 50\ 100\ 50 , 50\ 100\ 100 , 50\ 100\ 200 , 50\ 200 , 100\ 50\ 50 , 100\ 50\ 100 , 100\ 50\ 200 , 100\ 100 , 100\ 200 , 200 \}.$

Alphabete, Wörter, Sprachen

› Endliche Automaten verarbeiten Wörter

› Wörter sind endliche Zeichensequenzen

› Zeichensequenzen werden aus *atomaren* Symbolen gebildet

[z.B. 100 50 100 beim Schwimmbadautomat]

8

atomares Symbol

Der Lesekopf des endlichen Automaten liest das Eingabewort symbolweise von links nach rechts auf einem nach rechts offenen Eingabe-Speicher oder Eingabeband.

› Das Eingabealphabet eines endlichen Automaten ist die Menge der atomaren Symbole, aus denen die Eingabewörter aufgebaut werden können.

Eingabealphabet A_{swim} : $G = \{ 50 , 100 , 200 \}$

Allgemein bezeichnet man die Elemente eines ***Alphabets*** als ***Symbole*** oder ***Buchstaben***.

Beispiele für Alphabete sind:

- die Ziffern des Dezimalsystems

  G = {0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9},

- die Menge der kleinen lateinischen Buchstaben

  G = {a , b , c , ... , x , y , z},

- der Zeichenvorrat der Programmiersprache C.

Die endlich langen Zeichenfolgen, die über einem Alphabet Σ gebildet werden können, bezeichnet man als ***Wörter* über Σ**.

Wörter werden aufgebaut, indem Symbole oder bereits erzeugte Wörter aneinandergereiht, d.h. miteinander verkettet (***konkateniert***) werden; Konkatenation bedeutet Verkettung.

Definitionsgemäß gilt für ***die Menge Σ^* aller Wörter***, die über einem Alphabet Σ gebildet werden können:

- jeder Buchstabe des Alphabets ist auch Wort über Σ

$$a \in \Sigma \Rightarrow a \in \Sigma^* \quad ,$$

- die Verkettung vorhandener Wörter liefert neue Wörter

$$v,w \in \Sigma^* \Rightarrow vw \in \Sigma^* ,$$

- das *leere Wort* ε ist ein Wort über jedem Alphabet Σ

$$\varepsilon \in \Sigma^* \text{ und } \varepsilon w = w\varepsilon = w \text{ für alle } w \in \Sigma^* .$$

Im weiteren bezeichnet E^+ die Menge aller Wörter über E ohne das leere Wort:

$$E^+ = E^* - \{\varepsilon\} .$$

Normalerweise ist das betrachtete Alphabet nicht leer, so dass E^* abzählbar unendlich viele Wörter als Elemente besitzt; falls $E = \{i\}$ resultiert $E^* = \{\varepsilon\}$.

Beispiele:

1. Sei $E = \{a,b\}$, dann folgt:

$$E^* = \{\varepsilon,a,b,aa,ab,ba,bb,aaa,aab,aba,...\}$$

2. Schwimmbadautomat
   $G = \{ \underline{50} , \underline{100} , \underline{200} \}$,

   G^* ist die Menge aller Geldstückfolgen, die mit $\underline{50}$, $\underline{100}$ und $\underline{200}$ gebildet werden können inklusive der leeren Folge - vgl. oben. #

Für u,v,w $\in \Sigma^*$ liefert die **Konkatenation** ein Wort

$$\mathbf{z = uvw \in \Sigma^*,}$$

wobei dann u als ***Präfix***, v als ***Infix*** und w als ***Postfix*** (***Suffix***) bezeichnet wird.

Abkürzende Notation:

$w = v\,v =: v^2$ für $w \in \Sigma^*$, $v \in \Sigma$

$w = vv \ldots v =: v^k$ für $w \in \Sigma^*$, $v \in \Sigma$

$\underbrace{\qquad}_{k\text{ - mal}}$

v^k heißt k-te Potenz von v; $v^0 = \lambda$.

Beispiel:

$\Sigma^* = \{\lambda,a,b,aa,ab,ba,bb,aaa,aab,aba,...\}$

und

$\Sigma^* = \{\lambda,a,b,a^2,ab,ba,b^2,a^3,a^2b,aba,...\}$

bezeichnen die gleiche Menge.

#

Lexikographische Ordnung auf Σ^*

Mit der Ordnungsrelation "$\prec$" (lies "$\prec$" als vor)

$$\prec \; f \; \Sigma^* \times \Sigma^*$$

definiert man eine lexikographische Ordnung auf Σ^* wie folgt:
Für

$$v = v_1 \ldots v_m \,, \; v_i \in \Sigma \,, \; 1 \leq i \leq m$$

und

$$w = w_1 \ldots w_n \,, \; w_i \in \Sigma \,, \; 1 \leq i \leq n$$

gilt

$$v \prec w$$

genau dann, wenn:

1. $m < n$

oder

2. $m = n$ und es existiert ein k ($1 \leq k \leq m$) mit
$v_1 \ldots v_k = w_1 \ldots w_k$ und $v_{k+1} < w_{k+1}$.

Beispiel:
$\Sigma = \{ a , b \}$, d.h a < b (lies " < " als vor);
dann folgt:

$bb \prec aab$ (wegen 1.),
$baab \prec babb$ (wegen 2.). #

Die Anzahl der Buchstaben eines Wortes legt die ***Länge*** des Wortes fest; die Länge eines Wortes läßt sich rekursiv anhand der Funktion

$$l: \Sigma^* \to \mathbb{N}$$

mit

$$l(\varepsilon) = 0$$

und

$$l(wa) = l(w) + 1 \text{ für } w \in \Sigma^* , a \in \Sigma$$

berechnen.

Die Länge des Wortes **aba** über $\Sigma = \{a,b\}$ ist 3; die rekursive Berechnung stellt sich wie folgt dar:

$$l(aba) = l(ab) + 1 = l(a) + 1 + 1 =$$

$$= l(\varepsilon) + 1 + 1 + 1 = 0 + 1 + 1 + 1 = 3.$$

Häufig wird die Länge l(w) eines Wortes w auch durch die Bezeichnung |w| dargestellt; diese Notation erfolgt in Anlehnung an die Bezeichnung für die **Mächtigkeit (Kardinalität) einer Menge**: Die Mächtigkeit |M| einer Menge M ist die Anzahl der Elemente, die zu M gehören.

Beispiel:

$$M = \{1,3,5,7,9\} , \quad |M| = 5;$$

$$A = \{a_1,a_2,a_3, \ldots ,a_{n+1}\} , \quad |A| = n+1 .$$

#

Formale Sprachen

Sei Σ ein Alphabet, jede Teilmenge

$$L \subseteq \Sigma^*$$

heißt formale Sprache über Σ.

Sprachen sind Mengen von Wörtern und können mit Hilfe der Mengenoperationen Vereinigung, Durchschnitt und Komplementbildung miteinander verknüpft werden;
die ***Konkatenation*** (Verkettung) zweier Sprachen L_1 und L_2 über Σ ist wie folgt definiert:

$$L_1 \circ L_2 := \{ vw \mid v \in L_1 , w \in L_2 \}.$$

<u>Beispiel:</u>
$L_1 = \{ \varepsilon , ab , abb \}$, $L_2 = \{ b , ba \}$ seien Sprachen über $\{ a , b \}$;

$L_1 \circ L_2 := \{b , ba , abb , abba , abbb , abbba \}$,

$L_2 \circ L_1 := \{b , bab , babb , ba , baab , baabb \}$ #

Sei L eine Sprache über Σ, dann definiert man die n-te Potenz von L rekursiv über:

$$L^0 = \{ \varepsilon \} ,$$

$$L^{n+1} = L^n \circ L \quad .$$

<u>Beispiel:</u>
$L = \{ a , ab \} \quad \subset \Sigma$

$$L^0 = \{ \varepsilon \} ,$$

$L^1 = L^0 \circ L = \{ \varepsilon \} \circ \{ a , ab \} = \{ a , ab \} = L ,$

$L^2 = L^1 \circ L = \{ a , ab \} \circ \{ a , ab \}$
$= \{ a^2 , a^2b , aba , (ab)^2 \} ,$

$L^3 = L^2 \circ L = \{ a^2 , a^2b , aba , (ab)^2 \} \circ \{ a , ab \}$
$= \{ a^3 , a^3b , a^2ba , \ldots , (ab)^3 \} ,$

usw.

#

Als Kleene-Stern-Produkt einer Sprache L bezeichnet man die Vereinigung aller ihrer Potenzen L^n , n $ 0 :

$$L^{*} := \bigcup_{n \geq 0} L^n = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \ldots$$

$$L^{+} := L^{*} \setminus L^0 \quad .$$

Zustände und Zustandsübergänge

Die möglichen Zustände eines Automaten bilden die *Zustandsmenge* S des Automaten;

die Zustandsmenge S sei im folgenden immer eine endliche Menge (6 endlicher Automat).

Der Schwimmbadautomat besitzt die Zustandsmenge:

$$S = \{ \text{start} , s_{50} , s_{100} , s_{150} , s_{200} \} .$$

Zu Beginn einer Verarbeitung befindet sich ein Automat immer in einem ausgezeichneten Zustand: dem *Startzustand*.

Ein Automat akzeptiert eine Eingabesequenz, falls er sich nach deren Verarbeitung in einem Endzustand befindet.

Die *Endzustände* eines Automaten werden durch die Menge F f S bezeichnet.

Der Schwimmbadautomat besitzt die Endzustände:

$$F = \{ s_{200} \} ,$$

d.h. es gibt nur einen Endzustand.

In Abhängigkeit vom aktuellen Zustand und dem nächsten zu verarbeitenden Symbol der Eingabefolge geht der Automat in einen neuen Zustand über. Diese Übergänge werden durch die *Zustandsüberführungsfunktion*

$$*: S \times G \; 6 \; S$$

beschrieben;

$$*(s , a) = s'$$

bedeutet, daß der Automat, wenn er sich im Zustand s und der Lesekopf sich unter einer Eingabe-Speicherzelle mit dem Inhalt a befindet, in den Zustand s' übergeht.

Gebräuchliche Darstellungen der Zustandsüberführungsfunktion sind das *Zustandsdiagramm* und die *Zustandstabelle*.

Zustandstabelle des Schwimmbadautomaten

Symbole des Eingabealphabets Ä

Zustände {

*	50	100	200
start	s_{50}	s_{100}	s_{200}
s_{50}	s_{100}	s_{150}	s_{200}
s_{100}	s_{150}	s_{200}	s_{200}
s_{150}	s_{200}	s_{200}	s_{200}
s_{200}	-	-	-

Beachte:

* muß keine totale Funktion sein, d.h. * muß nicht für alle Paare (s , a) 0 S × G definiert sein!

Def.: ***Deterministischer endlicher Automat***

Ein deterministischer endlicher Automat
(endlicher Akzeptor) ist ein System (5-Tupel)

$$A = (G , S , * , s_0 , F) .$$

Dabei ist:

1. G ein endliches Eingabealphabet; die Symbole von G sind die Eingabezeichen einer Eingabesequenz;

2. S eine endliche Menge von Zuständen, die Zustandsmenge; die Elemente von S sind die möglichen Zustände des Automaten;

3. $s_0 \in S$ der Startzustand des Automaten;

4. $F \subseteq S$ die Menge der Endzustände, in die der Automat bei Verarbeitung einer akzeptierbaren Zeichenkette übergeht;

5. $*: S \times G \to S$ die Zustandsüberführungs-funktion, * beschreibt die Arbeitsweise von A.

Schwimmbadautomat

A_{swim} = ({ 50 , 100 , 200 },

{ start , s_{50} , s_{100} , s_{150} , s_{200} }, * , start ,

{ s_{200} })

* wird durch die oben aufgeführte Zustandstabelle oder das Zustandsdiagramm definiert.

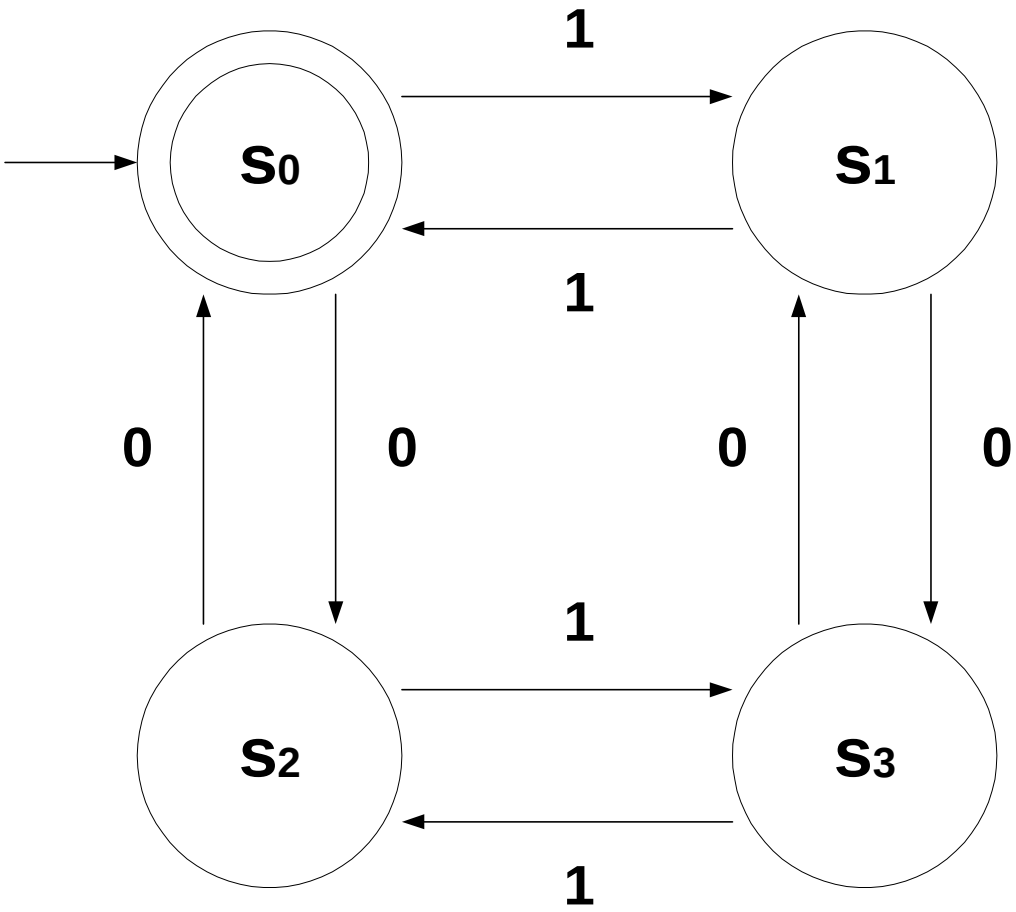
Beispiel:

$$A_{gerade} = (G , S , * , s_0 , F)$$

$G = \{ 0 , 1 \}$, $S = \{ s_0 , s_1 , s_2 , s_3 \}$, $F = \{ s_0 \}$,

s_0 ist sowohl Anfangs- als auch Endzustand;

Spezifizierung von * mittels Zustandsdiagramm:



Spezifizierung von * mittels Zustandstabelle:

*	0	1
s_0	s_2	s_1
s_1	s_3	s_0
s_2	s_0	s_3
s_3	s_1	s_2

Die Sprache des Automaten $L(A_{gerade})$ ist die Menge aller Wörter, die sowohl eine gerade Anzahl von Nullen (0) als auch eine gerade Anzahl von Einsen (1) aufweisen:

$L(A_{gerade}) = \{ 00 , 11 , 0101, 1010, 0011 , 1100 ,$

$0000, 1111, 000011 , . . . \} .$

#

Beispiel: *Akzeptor zur Paritätsprüfung*

Entwurf eines Automats A_{par} , der eine Bitsequenz

$$x = x_0\, x_1 \ldots x_{n-1}\,, \qquad x_i \in \{ 0 , 1 \}$$

liest und nach n Takten (Clock) im Zustand s_0 bzw. s_1 endet, abhängig davon, ob die Anzahl der Einsen in x gerade oder ungerade ist.

Dem jeweiligen Endzustand kann die *Parität* von x entnommen werden.

Der Automat ist ein Akzeptor, der alle möglichen Eingaben in die zwei Äquivalenzklassen

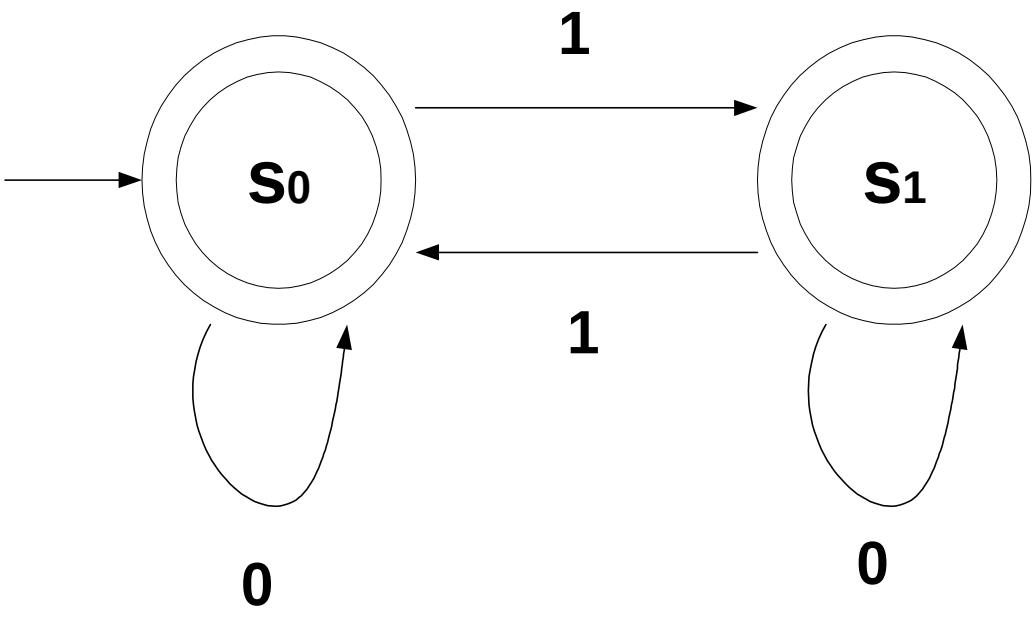
- gerade Anzahl von Einsen,

- ungerade Anzahl von Einsen

einteilt;

der erreichte Endzustand kennzeichnet die Klasse.

Zustandsdiagramm:



Zustandsüberführungsfunktion:

$$*(s_0 , 0) = s_0 , \qquad *(s_0 , 1) = s_1 ,$$

$$*(s_1 , 0) = s_1 , \qquad *(s_1 , 1) = s_0 .$$

Beachte: S = F

#

Erweiterte Zustandsüberführungsfunktion

Die Erweiterung der Zustandsüberführungs-funktion von Buchstaben auf Wörter für einen endlichen Automaten A = (G , S , * , s_0 , F) basiert auf der folgenden Definition:

$$*^*: S \times G^* \to S$$

mit

$$*^*(s , g) = s , \quad s \in S ,$$

$$*^*(s , aw) = *^*(*(s , a) , w) ,$$

$$s \in S , a \in G , w \in G^* .$$

$*^*$ beschreibt das sukzessive Abarbeiten eines Wortes v beginnend im Zustand s:

- für v = g verbleibt A im Zustand s,

- für v = aw wird der Folgezustand *(s , a) des ersten Buchstabens a von v berechnet, anschließend muß $*^*$ für diesen Folgezustand und den Rest w von v ermittelt werden,

- ist der Rest das leere Wort γ, so ist die Berechnung abgeschlossen.

$*^*(s , v) =$ s' besagt, daß, wenn A sich im Zustand s befindet, der Automat sich nach Abarbeitung des Wortes v in den Zustand s' übergeht.

Beispiel: *Schwimmbadautomat*

$*^*$ für den Zustand s_{50} und das Wort 50 50 100:

$$*^*(s_{50} , 50\ 50\ 100) = *^*(*(s_{50} , 50) , 50\ 100)$$

$$= *^*(s_{100} , 50\ 100)$$

$$= *^*(*(s_{100} , 50) , 100)$$

$$= *^*(s_{150} , 100)$$

$$= *^*(*(s_{150} , 100) , \gamma)$$

$$= *^*(s_{200} , \gamma)$$

$$= s_{200}$$ #

Reguläre Sprachen

Def.:

a) Sei A = (G , S , * , s_0 , F) endlicher Automat;
dann heißt die Sprache

$$L(A) = \{ w \in G^* \mid *^*(s_0 , w) \in F \}$$

die von A *akzeptierte Sprache*.

b) Eine Sprache $L \subseteq E^*$ heißt *regulär*, falls es einen endlichen Automaten gibt, der L akzeptiert, d.h. für den L = L(A) gilt.

c) Sei G ein Alphabet, dann bezeichnet:

- DFA_G die Klasse der von endlichen Automaten akzeptierten Sprachen über G,

- REG_G die Klasse der regulären Sprachen über G,

und es gilt wegen b): $DFA_G = REG_G$.

d) Zwei endliche Automaten A und A’ heißen äquivalent, falls sie dieselbe Sprache akzeptieren, d.h. falls L(A) = L(A’) gilt.

(DFA steht für ***D****eterministic* ***F****inite* ***A****utomaton*)

Beispiel: *Schwimmbadautomat*

$L(A_{swim}) = \{$ 50 50 50 50 , 50 50 50 100 , 50 50 50 200 , 50 50 100 , 50 50 200 , 50 100 50 ,50 100 100 , 50 100 200 , 50 200 , 100 50 50 , 100 50 100 , 100 50 200 , 100 100 , 100 200 , 200 $\}$.

#