Wirschaftsinformatik

Informatik Grundlagen

2.0 Logische Grundlagen

2.1 Aussagenlogik

Definition: Eine Aussage ist ein Satz, der entweder wahr oder falsch ist.

Beispiele: „Dies ist ein schönes Land“ ist keine Aussage, da es von dem Geschmack einer Person abhängig ist, ob ein Land schön ist, oder nicht. Daher kann diese Aussage nicht eindeutig „Wahr“ oder „Falsch“ zugeordnet werden.

„Es regnet draussen“ dagegen kann überprüft und objektiv beurteilt werden. Diesem Satz kann ich „Wahr“ oder „Falsch“ zuordnen und damit handelt es sich um eine Aussage.

Definition: Eine Aussageform ist ein Satz, der eine Variable enthällt. Durch Belegung dieser Variablen mit einer Konstante entsteht eine Aussage. Für die Variable ist ein Definitionsbereich anzugeben.

Beispiele:

„x ist eine Primzahl“ mit $x \in N$		(Aussageform)
„16 ist eine Primzahl“	f	(Aussage)
„x>25“ mit $x \in R$		(Aussageform)
„37,25>25“	w	(Aussage)

<u>Quantifizierung</u> <u>einer Aussageform</u>

Eine Aussageform A(x) wird zur Aussage durch Angabe eines Sog. **Quantors.**

All-Quantor : $\forall$ „für alle“

Existenz-Quantor : $\exists$ „es gibt“ (mind. 1 also)

Beispiele: $\forall x$: x ist Primzahl; $x \in N$

Die Ausage ist nicht wahr, da nicht alle $x \in N$ Primzahlen sind.

$\exists x$: x ist Primzahl; $x \in N$

Diese Aussage ist wahr.

Definition: Wenn A und B Aussagen sind, dann gelten für die Verknüpfungen von A und B folgende Wahrheitswerte, die in Wahrheitstafeln dargestellt werden :

<u>Konjunktion</u> : $A \wedge B$ (A und B)
Tafel dazu :

A	B	$A \wedge B=f(A,B)$
w	w	w
f	w	f
w	f	f
f	f	f

Disjunktion : $A \vee B$ (A oder B)
Tafel dazu :

A	B	$A \wedge B$=f(A,B)
w	w	w
f	w	w
w	f	w
f	f	f

Negation : $\overline{A}$, ~A
Tafel dazu :

A	~A
w	w
f	w
w	f
f	f

Regeln von D'Morgan

1. $\overline{\overline{A}} = A$
2. $\overline{A \wedge B} = \overline{A} \vee \overline{B}$
3. $\overline{A \vee B} = \overline{A} \wedge \overline{B}$

Vorrangregeln : $\overline{\ }$ vor $\wedge$ vor $\vee$
z.B. $\overline{A} \wedge B \vee C = C \vee \overline{A} \wedge B$

Test durch eine Wahrheitstafel

A	B	$\overline{A \wedge B}$	$\overline{A \vee B}$	$\overline{A} \vee \overline{B}$	$\overline{A} \wedge \overline{B}$
w	w	f	f	f	f
f	w	w	f	w	f
w	f	w	f	w	f
f	f	w	w	w	w

Als Beispiel der Negation

$\overline{\text{„Die Lampe brennt“} \wedge \text{„Es regnet“}}$ = „Die Lampe brennt nicht“ $\vee$ „Es regnet nicht“

$\overline{(A \vee B) \wedge C} = \overline{(A \vee B)} \vee \overline{C} = (\overline{A} \wedge \overline{B}) \vee \overline{C}$
$\overline{(A \wedge B \vee C)} = \overline{(A \wedge B)} \wedge \overline{C} = (\overline{A} \vee \overline{B}) \wedge \overline{C}$

Bei dem letzten Beispiel sieht man, wie wichtig Klammern sind, denn andernfalls wäre durch die Reihenfolge der Zeichen ein anderer Bool'scher Ausdruck entstanden.

Es ist wichtig, darauf zu achten, daß die Distributiv- und Kommutativ-Gesetzte entsprechend Anwendung finden. Es ist daher nicht unbedingt egal, in welcher Reihenfolge ein Ausdruck ausgewertet wird und welche Elemente dabei zusammen genommen werden.

Test durch eine Wahrheitstafel

A	B	C	$\overline{(A \wedge B)} \vee C$	$(\overline{A} \vee \overline{B}) \wedge \overline{C}$	$\overline{A} \vee \overline{B} \wedge \overline{C}$	
w	w	w	f	f	f	
f	w	w	f	f	f	
w	f	w	f	f	w	X
f	f	w	f	f	w	X
w	w	f	f	f	f	
f	w	f	w	w	w	
w	f	f	w	w	w	
f	f	f	w	w	w	

In dieser Tabelle ist ebenfalls zu erkennen, daß bei der Konjunktion gilt, ist bereits ein Mitglied der Konjunktion „falsch“, ist die ganze Konjunktion „falsch“.
Entsprechend gilt bei der Disjunktion, daß bereits nur ein Mitglied „wahr“ sein muß, um den gesamten Term „wahr“ werden zu lassen.

Es ist weiterhin möglich, jede Konjunktion durch Disjunktion und Negation darzustellen, wie es auch möglich ist, die Disjunktion mit der Konjunktion und der Negation darzustellen. Man nutzt dabei die D’Morgan’sche Gesetze aus:

$A \vee B = \overline{\overline{A \vee B}} = \overline{\overline{A} \wedge \overline{B}}$ (Die doppelte Negation verändert das Ergebnis nicht!!)

$A \wedge B = \overline{\overline{A \wedge B}} = \overline{\overline{A} \vee \overline{B}}$

Durch dieses Verfahren kann man auch komplizierte Terme in eine Normalform bringen, so daß z.B. nur noch Konjunktionen und Negationen auftauchen. Diese Form nennt man dann die „Konjunktive Normalform“. Gleiches gilt entsprechend für die „Disjunktive Normalform“

Wie wirkt sich nun die Negation auf die Quantisierer aus?

$$\overline{\forall x : A(x)} = \exists x : \overline{A(x)}$$

z.B. „Alle Studenten sind Anwesend.“ = „es gibt (wenigstens) einen Studenten, der nicht anwesend ist.“

und

$$\overline{\exists x : A(x)} = \forall x : \overline{A(x)}$$

z.B. „Es gibt einen Studenten, der anwesend ist.“ = „Alle Studenten sind weg.“

Exklusives Oder, Implikation, Äquivalenz, Umkehrschluß:

Ein kleines Tablau:

		XOR	Implikation	Äquivalenz	Umkehrschluß
A	B	A >—< B	A => B	A <=> B	$\overline{B} \Rightarrow \overline{A}$
w	w	f	w	w	w
f	w	w	f	f	f
w	f	w	w	f	w
f	f	f	w	w	w

Das Exclusive Oder :

$$A \succ\prec B = A \; xor \; B = (\overline{A} \wedge B) \vee (A \wedge \overline{B})$$

Die Implikation :

Bei der Implikation wird „gefolgert“. Dabei ist es wichtig, ob bereits die Startaussage „wahr“ ist. Ist die Startaussage bereits „falsch“, braucht man keine weitere Implikation mehr anzustellen, da man schließlich aus einer „falschen“ Ausage alles folgern kann. Daher ist eine Implikation aus einer „falschen“ Aussage auch konsequent „wahr“, wie nicht nur das Tablau zeigt, sondern auch die Gesetzmäßigkeit :

$$A \Rightarrow B = \overline{A} \vee B$$

Beispiele:

„Es regnet => es sind Wolken da“

$9=10 \overset{-8}{\Rightarrow} 1=2 \Rightarrow$ „ich bin Papst“ Die Folgerungen sind aber immer „wahr“!!
f f f

$-1=1 \overset{x^2}{\Rightarrow} 1=1$ aber: Aus einer falschen Startaussage wurde eine „wahre“ Aussage.

Falsch ist z.B. auch folgende Implikation:

$a^2+b^2+2ab \geq 0 \Rightarrow a^2+b^2 \geq -2ab \Rightarrow (a+b)^2 \geq 0$

Keiner kann über die erste Aussage etwas sagen, da nicht bekannt ist, ob sie „wahr“ ist, oder nicht.

Richtig dagegen ist die andere Reihenfolge, da man über die erste Aussage durch die Regeln der Quadration die Aussage bestetigen kann:

$(a+b)^2 \geq 0 \Rightarrow a^2+b^2+2ab \geq 0 \Rightarrow a^2+b^2 \geq -2ab$

Die Äquivalenz :

$$A \Leftrightarrow B = (A \Rightarrow B) \wedge (B \Rightarrow A)$$

Um die richtigen Bool’schen Operatoren für die Äquivalenz zu finden, kann man in der Tabelle bei „xor“ mal die Werte vergleichen. Man stellt dann fest, daß die Äquivalenz die Umkehrfunktion zu „xor“ ist und daher auch so darstellbar ist:

$$A \Leftrightarrow B = \overline{(\overline{A} \wedge B) \vee (A \wedge \overline{B})} = \overline{(\overline{A} \wedge B)} \wedge \overline{(A \wedge \overline{B})}$$
$$= (A \vee \overline{B}) \wedge (\overline{A} \vee B) = (B \Rightarrow A) \wedge (A \Rightarrow B)$$

Man kann nun das logische Und als Malpunkt und das logische Oder als Pluszeichen schreiben. Das erleichtert auf jeden Fall die Einsicht bei folgenden Operationen, denn dann ergibt sich folgende Umformungsmöglichkeit:

$$\left(A \vee \overline{B}\right) \wedge \left(\overline{A} \vee B\right) = \left(A + \overline{B}\right) \cdot \left(\overline{A} + B\right) = A \cdot \overline{A} + A \cdot B + \overline{A} \cdot \overline{B} + \overline{B} \cdot A$$
$$= A \wedge B \vee \overline{A} \wedge \overline{B} = \left(A \wedge B\right) \vee \overline{\left(A \wedge B\right)}$$

Der Umkehrschluß :

$A \Rightarrow B = \overline{B} \Rightarrow \overline{A}$

„Es regnet => es sind Wolken am Himmel.“ =
„Es sind keine Wolken am Himmel => es regnet nicht.“

Boolsche Algebra

Das Alphabet der Boolschen Algebra :
B={0,1}
Operatoren :$\vee$, $\wedge$, $\overline{\ }$
Rechenregeln: seien x,y,z $\in$ B, dann gilt

Kommutativgesetz:
$x \vee y = y \vee x$
$x \wedge y = y \wedge x$
Assoziativgesetz:
$x \vee (y \vee z) = (x \vee y) \vee z$
$x \wedge (y \wedge z) = (x \wedge y) \wedge z$
Distributivgesetz:
$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$
$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$
Absorbtionsgesetz:
$x \vee (x \wedge y) = x$
$x \wedge (x \vee y) = x$
Weitere Gesetzmäßigkeiten:
$0 \vee x = x$; 0 als neutrales Element des OR ($\vee$)
$1 \wedge x = x$; 1 als neutrales Element des AND ($\wedge$)
$x \vee \overline{x} = 1$; Tautologie
$x \wedge \overline{x} = 0$

aus $x \wedge y = 0$ und $x \vee y = 1$ folgt : $y = \overline{x}$

sei y=1 => x=0
sei y=0 => x=1
} => $y = \overline{x}$

D'Morgan allgemein:

$$\overline{\bigvee_{i=1,..,n} x_i} = \overline{x_1 \vee x_2 \vee x_3 \vee ... \vee x_n} = \bigwedge_{i=1,..,n} \overline{x_i}$$

$$\overline{\bigwedge_{i=1,..,n} x_i} = \overline{x_1 \wedge x_2 \wedge x_3 \wedge ... \wedge x_n} = \bigvee_{i=1,..,n} \overline{x_i}$$

Boolsche Funktionen :

$$f : (e_1,...,e_n) \to a$$
$$e_i \in B \;\; ; \forall i \in \{1,...,n\} \;\; ; a \in B$$

z.B. : $f(x) = \overline{x}$

$f(x) = \overline{x \wedge y}$ NAND-Funktion

$f(x) = \overline{x \vee y}$ NOR-Funktion

$f(x,y) = (x \wedge y) \vee \overline{(x \vee y)}$ äquivalenz Funktion

$f(x,y) = (x \vee y) \wedge \overline{(x \wedge y)}$ antivalenz (XOR)

$f(x,y,z) = (x \Rightarrow y) \wedge (x \Leftrightarrow y)$

*Neuformulierung von Boolschen Funktionen (**Disjunktive Normalform**):*

Def.: Eine Vollkonjunktion ist ein Ausdruck, in welchem alle vereinbarten Variablen konjunktiev verbunden sind. Die Variablen können auch negiert sein.
Analog wird die Volldisjunktion definiert.

In der Disjunktiven Normalform einer Boolschen Funktion werden Vollkonjunktionen mit Disjunktionen verknüpft.

x	y	z	x=>y	x<=>z	f(x,y,z)
0	0	0	1	1	1
0	0	1	1	0	0
0	1	0	1	1	1
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	0	1	0
1	1	0	1	0	0
1	1	1	1	1	1

Bestimmung der Disjunktiven Normalform (DNF):

„Bestimme in der Spalte der Funktionswerte die einsen, erzeuge für die einsen die entsprechende Vollkonjunktion und verknüpfe diese konjunktiv."

$$\Rightarrow f(x,y,z) = (\overline{x} \wedge \overline{y} \wedge \overline{z}) \vee (\overline{x} \wedge y \wedge \overline{z}) \vee (x \wedge y \wedge z)$$

Bestimmung der Konjunktiven Normalform (KNF):

$$\Rightarrow f(x,y,z) = \overline{(\bar{x} \wedge \bar{y} \wedge z)} \wedge \overline{(\bar{x} \wedge y \wedge z)} \wedge \overline{(x \wedge \bar{y} \wedge \bar{z})} \wedge \overline{(x \wedge \bar{y} \wedge z)} \wedge \overline{(x \wedge y \wedge \bar{z})}$$
$$(KNF) \quad = (x \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y \vee z) \wedge (\bar{x} \vee y \vee \bar{z}) \wedge (\bar{x} \vee \bar{y} \vee z)$$
$$(DNF) \quad = (\bar{x} \wedge \bar{y} \wedge \bar{z}) \vee (\bar{x} \wedge y \wedge \bar{z}) \vee (x \wedge y \wedge z)$$

*Vereinfachung von Boolschen Funktionen nach dem Verfahren von **Quine+McClusky***

$$(\bar{x} \wedge \bar{y} \wedge \bar{z}) \vee (\bar{x} \wedge y \wedge \bar{z}) = \bar{x} \wedge z \vee (x \wedge y \wedge \bar{z})$$

Da gilt:

$$(\bar{x} \wedge \bar{y} \wedge z) \vee (\bar{x} \wedge y \wedge z) = \bar{x} \wedge z \wedge (\bar{y} \vee y) = \bar{x} \wedge z \wedge 1 = \bar{x} \wedge z$$

Das y ist für beide Ausdrücke nicht mehr von Bedeutung und wird daher weggelassen. (Die Klammern im ersten Term sind nicht grundsätzlich nötig, vereinfachen aber die Lesbarkeit)

1. Teil zusammenfassen:

Die Vollkonjunktionen werden nach Anzahl der Negationen, die sie enthalten, zusammengefasst, dann werden die Konjunktionen benachbarter Gruppen systematisch auf Zusammenhänge untersucht. D.h. Konjunktionen, die sich in nur einer Negation unterscheiden, werden zusammengefasst. (Quine+McClusky).
Die reduzierten Konjuktionen sind wieder zu Gruppen zusammenzufassen. Mehrfach vorkommende Konjunktionen werden nur einmal aufgeführt. Dieses Verfahren ist so oft wie möglich anzuwenden.

Normalisierungsprozess:

Beispiel:

$$f(x,y,z) = \bar{x} \vee (y \wedge z) \vee (x \wedge y)$$

Wertetabelle:

x	y	z	f(x,y,z)
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

DNF:

$$f(x,y,z) = (\bar{x} \wedge \bar{y} \wedge \bar{z}) \vee (\bar{x} \wedge \bar{y} \wedge z) \vee (\bar{x} \wedge y \wedge \bar{z}) \vee (\bar{x} \wedge y \wedge z) \vee (x \wedge y \wedge \bar{z}) \vee (x \wedge y \wedge z)$$

KNF:

$$f(x,y,z) = (x \vee y \vee z) \wedge (x \vee y \vee z)$$
$$= \bar{x} \vee y \vee (z \wedge \bar{z}) \quad \text{;Distributivgesetz}$$
$$= \bar{x} \vee y \vee 0 = \bar{x} \vee y$$

Wollen wir nun die ***DNF***, welche oben steht, nach dem Verfahren von Quine+McClusky etwas vereinfachen. Dazu stellen wir erst einmal ein Tablau auf. Wir sortieren dabei die Vollkonjunktionen nach der Anzahl der Negationen:

1	$x \wedge y \wedge z$
2	$\bar{x} \wedge y \wedge z$
3	$x \wedge y \wedge \bar{z}$
4	$\bar{x} \wedge \bar{y} \wedge z$
5	$\bar{x} \wedge y \wedge \bar{z}$
6	$\bar{x} \wedge \bar{y} \wedge \bar{z}$

Nun könne wir beginnen, Konjunktionen mit hilfe des Distributivgesetzes zu verknüpfen, wobei immer ein Glied herausfallen sollte:

1,2	$y \wedge z$	(Unsortiert!)
1,3	$x \wedge y$	
2,4	$\bar{x} \wedge z$	
2,5	$\bar{x} \wedge y$	
3,5	$y \wedge \bar{z}$	
4,6	$\bar{x} \wedge \bar{y}$	
5,6	$\bar{x} \wedge \bar{z}$	

Nun müssen diese verkürzten Konjunktionen noch nach dem Vorkommen ihrer Variablen und Negationen sortiert werden.

1	$x \wedge y$
2	$\bar{x} \wedge y$
3	$\bar{x} \wedge \bar{y}$
4	$\bar{x} \wedge z$
5	$\bar{x} \wedge \bar{z}$
6	$y \wedge z$
7	$y \wedge \bar{z}$

Jetzt kann erneut zusammengefasst werden, sofern noch möglich:

1,2	y
2,3	$\bar{x}$
4,5	$\bar{x}$
6,7	y

Da ist nun nichts mehr zusammenfassbar. Daher kann man nun (s. Folie) mit einer Kreuztabelle feststellen, wo die reduzierten Konjunktionen in den Vollkonjuntkionen vorkommen:

2.Teil	$x \wedge y \wedge z$	$\bar{x} \wedge y \wedge z$	$x \wedge y \wedge \bar{z}$	$\bar{x} \wedge \bar{y} \wedge z$	$\bar{x} \wedge y \wedge \bar{z}$	$\bar{x} \wedge \bar{y} \wedge \bar{z}$
y	X	X	X		X	
$\bar{x}$		X		X	X	X

Diese Tabelle führt nun zu dem Schluß (wer hätte das gedacht!?), daß alle Konjuktionen durch die beiden Variablen *x* und *y* darstellbar sind, da beide zusammen alle Konjunktionen abdecken. Die Ergebnisfunktion sieht nun so aus :

$$f(x,y,z) = \bar{x} \vee y$$

was bereits die Vereinfachung der *KNF* gezeigt hatte.

3.0 Zahlensysteme

3.1 Stellensysteme

Der Wert einer Ziffer hängt von seiner Position innerhalb seiner Zahl ab.
z.B. $283 = (2 \cdot 10 + 8) \cdot 10 + 3$
Das ganze erinnert an das *Horner Schema*. Nach dem Einmultiplizieren erhalten wir:

$$283 = 2 \cdot 10^2 + 8 \cdot 10^1 + 3 \cdot 10^0$$

Jede Stelle hat also ihren eigenen Potenzwert zur Zahlenbasis. Daher kann man eine Zahl auch allgemein so ausdrücken: (z_i ist dabei die jeweilige Stelle/Ziffer)

$$z = \sum_{i=0}^{n-1} z_i \cdot b^i = (z_{n-1} z_{n-2} z_{n-3} ... z_0)_b \quad ; b \in N \quad ; b > 1 \quad ; 0 \le z_i \le b-1$$

Wichtige Basen sind:

b=10	Dezimalsystem;	Alphabet *A*={0,...,9}
b=2	Dualsystem;	Alphabet *A*={1,2}
b=8	Oktalsystem;	Alphabet *A*={0,...,8}
b=16	Hexadezimalsystem; (auch: Sedezimalsystem)	Alphabet *A*={0,...,9,A,...,F}

Beispiele:

$$101101_2 = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$
$$= 1 \cdot 32 + 1 \cdot 8 + 1 \cdot 4 + 1$$
$$= 45_{10}$$

$$372_8 = 3 \cdot 8^2 + 7 \cdot 8^1 + 2 \cdot 8^0$$
$$= 192 + 56 + 2$$
$$= 250_{10}$$

$$1E9_{16} = 3 \cdot 16^2 + 14 \cdot 16^1 + 9 \cdot 16^0$$
$$= 256 + 224 + 9$$
$$= 489_{10}$$
$$FF_{16} = 255_{10}$$

Das ganze funktioniert natürlich auch mit Brüchen oder mit den Zahlen der Menge der rationalen Zahlen Q:

Allgemein:

$$z = \sum_{i=-m}^{n-1} z_i \cdot b^i = z_{n-1}b^{n-1} + ... + z_0 b^0 + z_{-1}b^{-1} + ... + z_{-m}b^{-m}$$
$$= (z_{n-1}z_{n-2}...z_0 z_{-1}z_{-2}...z_m)$$

Beispiel:

$$101{,}11_2 = 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 + 1\cdot 2^{-1} + 1\cdot 2^{-2}$$
$$= 5{,}75_{10}$$
$$1{,}6_8 = 1\cdot 8^0 + 6\cdot 8^{-1} = 1{,}75_{10}$$

3.2 Zahlenumwandlung

Wir versuchen uns an der Umwandlung von Dezimal zu Dual. Dabei suchen wir die höchste binäre Zahl, die in unseren Kandidaten noch hineinpasst. Mit dem übrigen Rest verfahren wir ebenso, bis der Rest gleich null ist:

$45_{10} = 1\cdot 2^5 + 1\cdot 2^3 + 1\cdot 2^2 + 1\cdot 2^0 = 101101_2$

$45 - 32 = 13 \qquad 32 = 2^5$

$13 - 8 = 5 \qquad 8 = 2^3$

$5 - 4 = 1 \qquad 4 = 2^2$

1 ist trauriger Rest.

Das Verfahren ist mühselig und nicht effektiv. Man entschließt sich daher dazu, das *Horner Schema* wieder zu bemühen und dieses mal in umgekehrter Reihenfolge auszuführen:

z.B.

$$11001_2 = (((1\cdot 2 + 1)\cdot 2 + 0)\cdot 2 + 0)\cdot 2 + 1 = 25_{10}$$

Und nun zurück:

$25 \div 2 = 12$ Rest 1 ↑

$12 \div 2 = 6$ Rest 0

$6 \div 2 = 3$ Rest 0

$3 \div 2 = 1$ Rest 1

$1 \div 2 = 0$ Rest 1

Daraus ergibt sich 11001 (von unten nach oben)

Beispiel:

998 ÷ 2 = 499	Rest 0
499 ÷ 2 = 249	Rest 1
249 ÷ 2 = 124	Rest 1
124 ÷ 2 = 62	Rest 0
62 ÷ 2 = 31	Rest 0
31 ÷ 2 = 15	Rest 1
15 ÷ 2 = 7	Rest 1
7 ÷ 2 = 3	Rest 1
3 ÷ 2 = 1	Rest 1
1 ÷ 2 = 0	Rest 1

Daraus folgt $998_{10} = 1111100110_2$

Andere Zahlensysteme funktionieren analog.

$998_{10} = 3E6_{16}$

998 ÷ 16 = 62	Rest 6
62 ÷ 16 = 3	Rest E (= 14)
3 ÷ 16 = 0	Rest 3

Das Umwandeln von Brüchen:

Allgemein: $z = \sum_{-m}^{-1} z_i b^i = z_{-1} b^{-1} + \ldots + z_{-m} b^{-m}$

In diesem Fall verwendet man das fortgesetzte Multiplizieren mit der Basis und schneidet die Vorkommastellen ab. Nur die Nachkommastellen werden weiter verwendet, bis man nicht mehr weiter kann:

z.B.

$0{,}1011_2 = 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} = 0{,}6875_{10}$

$0{,}6875 \cdot 2 = 1{,}375$	Fällt raus : 1
$0{,}375 \cdot 2 = 0{,}75$	Fällt raus : 0
$0{,}75 \cdot 2 = 1{,}5$	Fällt raus : 1
$0{,}5 \cdot 2 = 1$	Fällt raus : 1
0	

Dieses mal müssen die Zahlen aber von oben nach unten gelesen werden : 0,1011

Dual->Oktal:

z.B.

$$
\begin{aligned}
&011010110{,}011011_2 \\
&= 0\cdot 2^8 + 1\cdot 2^7 + 1\cdot 2^6 + 0\cdot 2^5 + 1\cdot 2^4 + 0\cdot 2^3 + 1\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0 \\
&\quad + 0\cdot 2^{-1} + 1\cdot 2^{-2} + 1\cdot 2^{-3} + 0\cdot 2^{-4} + 1\cdot 2^{-5} + 1\cdot 2^{-6} \\
&= \left(2^3\right)^2\cdot\left(0\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0\right) + \left(2^3\right)^1\cdot\left(0\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0\right) + \left(2^3\right)^0\cdot\left(1\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0\right) \\
&\quad + \left(2^3\right)^{-1}\cdot\left(0\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0\right) + \left(2^3\right)^{-2}\cdot\left(0\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0\right) \\
&= 3\cdot 8^2 + 2\cdot 8^1 + 6\cdot 8^0 + 3\cdot 8^{-1} + 3\cdot 8^{-2} \\
&= 326{,}33_8
\end{aligned}
$$

Man faßt also Binär immer Bits zu dreien ($2^3 = 8$) zusammen, die dann Wertigkeiten im Bereich von 0 bis 7 haben, so wie es die Oktalzahl nun mal hat. Dann hat man die binäre Zahl auch schon oktal dargestellt. Hexadezimal funktioniert das analog, nur daß man hier immer die Bits zu Vierergruppen ($2^4 = 16$) zusammenfasst. Das passt aber sehr gut zu der Byte-Darstellung von jeweils 8 Bit, so daß das Hexadezimalsystem auf den Rechnern sehr oft verwendet wird. Es ist kürzer als eine binäre Darstellung, aber genauso Aussagekräftig.

Dual->Hexadezimal:

$000110101101{,}11011100_2 = 1AD{,}DC_{16}$
$AFFE_{16} = 1010\ 1111\ 1111\ 1110_2$

Oktal->Hexadezimal:

Diese Umwandlung könnte problematisch werden, wenn man die Dreiergruppen des Oktalsystem zu Vierergruppen umgruppieren will. Man macht es sich einfacher, wenn man den Weg über das Binärsystem geht.

$3A7_{16} = 001110100111_2 = 1647_8$

3.3 Rechnen im Dualsystem

Die Addition:

Die Addition wird zurückgeführt auf:

$$
\begin{aligned}
0 + 0 &= 00 \\
0 + 1 &= 01 \\
1 + 0 &= 01 \\
1 + 1 &= 10
\end{aligned}
$$

Man sollte immer nur zwei Zahlen auf einmal addieren, da man dann nur mit Überträgen von 1 zu rechnen hat. Mehr wird riskant (ist aber auch möglich!).

Nun verwendet man Boolsche Funktionen als Mittel zur Addition. Dabei erzeugt man eine Funktion für die Addition und eine für den Übertrag.
Fassen wir die Addition zweier Binär-Digits einfach in einer Tabelle zusammen:

x	y	$\ddot{U}(x,y)$	$W(x,y)$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Es fällt auf, daß die Funtkion $W(x,y)$ offensichtlich auch mit XOR erreicht werden kann. Eine Boolsche Funktion, die Addieren kann, nennt man auch ***Halbaddierer***.

$\ddot{U}(x,y) = x \wedge y$
$W(x,y) = x \text{ xor } y = \left(x \wedge \overline{y}\right) \vee \left(\overline{x} \wedge y\right) = \left(x \vee y\right) \vee \overline{\left(x \vee y\right)}$

Verwendet man Boolsche Funktionen zur Addition, so wird nicht wirklich addiert, sondern nur dafür gesorgt, daß zwei Bits so verknüpft werden, daß das Ergebnis einer Addition gleich kommt.

Beispiel einer Addition:

```
  1 0 1 0 1 1
+ 0 1 1 0 1 1 1 1
-----------------
1 0 0 0 0 1 0
```

Die Subtraktion:

0 - 0 = 00
0 - 1 = (11) Hier muß man aufpassen. (->Zweierkomplement)
1 - 0 = 01
1 - 1 = 01

Beispiel:

```
  1 1 0 1
- 0 1 1 1 0
-----------
  0 1 1 1
```

Probe:

```
  0 1 1 1
+ 0 1 1 1 0
-----------
  1 1 0 1
```

Die Mutliplikation:

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

Beispiel der Multiplikation:

```
10110111·101
10110111
 00000000
  10110111
---------
1110010011
```

Die Division:

Bei der Division geht man, ebenso wie bei der Addition, Subtraktion und Multiplikation grundsäzlich so vor, wie man es im Dezimalsystem gewohnt ist.

```
110101:101 = 1010,100...
101
  110
  101
    110
    101
      100...
```

Die Subtraktion durch Komplementbildung und Addition:

527 – 368 = 159
527 + 632 = 1159 = 527 + (1000 - 368)

oder

527 + 631 = 1158 => 1158 + 1 = 1159 (nachträgliche Korrektur des Neunerkomplements)

632 ist das Zehnerkomplement von 368,
631 ist das Neunerkomplement.

In Dualer Form:

z.B. $z = 101001$

$-z = 010110$ Das Einerkomplement wird gebildet, indem jedes Bit von *z* schlicht negiert wird.

$-z = 010111$ Das Zweierkomplement schließlich ist das Einerkomplement plus eins. (wie oben.)

Beispiele:

```
 1 0 1 1          1 0 1 1
-0 1 0 1         +1 0 1 1 1
---------        ----------
 0 1 1 0          1 0 1 1 0
```

```
 1 1 0 1 1 1 0 0 0          1 1 0 1 1 1 0 0 0
-0 0 0 1 0 1 1 1 1 1       +1 1 1 1 0 1 0 0 0 1
-----------------------    ---------------------
 1 1 0 0 0 1 0 0 1          1 1 1 0 0 0 1 0 0 1
```

Würde man sich auf Bitkonstellationen von nur 8 oder 16 Bit beschränken und alle höherwertigen Bits als Vorzeichenbit interpretieren und entsprechend setzen, so würde das Zweierkomplement und das Ergebnis vorzeichenrichtig sein.

Beispiele:

```
       -1 = 11111111 (2er)
-10101111 = 01010001 (2er)

-58 + 33 = -25
  | 2er Komplement ^
  v               |
 42 + 33 =  75
```

$-58_{10} = -111010_2 = 000110_2$
$33_{10} = 100001_2 = \underline{+100001_2}$
100111_2 (=39)
=> 011001 = 25

Rückkomplementierung notwendig, da kein Übertrag aufgeteten ist!

Wir brauchen ein Vorzeichen:

1. Bit als Signal: 1 für –
0 für +

z.B. 01101001 = $+105_{10}$
komplement: 10010111 = -105_{10}

oder 00000111 = $+7_{10}$
11111001 = -7_{10}

$x = 179_{10} = 010110011_2$
$y = 109_{10} = 001101101_2$
$-x = -179_{10} = 101001101_2$
$-y = -109_{10} = 110010011_2$

```
   x 010110011
  -y 110010011
-------------
  70 001000110 = 70

   y 001101101
  -x 101001101
-------------
 -70 110111010 => (2er) [-]001000110 = [-]70

  -x  101001101
  -y  110010011
-------------
-288 1011100000=> (2er) [-]0100100000=[-]288
```

Das große Problem mit dem letzten Beispiel ist der Überlauf. Dieser dürfte unter normalen Umständen nie auftreten und ist daher auch mit äußerster Vorsicht zu genießen! Sollte dieses Phänomen einmal auftauchen und man kennt die Zahlen nicht (kann also nicht zwingend auf einen Überlauf schließen!), so sollte man sich das Ergebnis ansehen. Wie im obigen Beispiel zu erkennen, würde die Zahl wider erwarten positiv sein, schmisse man den Übertrag wie gewohnt einfach weg. Das ist das einzige Anzeichen (dessen Zuverlässigkeit mir nicht bekannt ist), das man meines Erachtens verwenden kann, um den tödlichen Überlauf zu erkennen. Danach reagiert man darauf schlicht mit dem erneuten 2er-Komplement und tut so, als gehöre die vorderste Eins als Negativkennzeichen wirklich dazu.
Der Zahlenbereich bei 8-Bit Zahlen inklusive 2er-Komplement: **-128 – +127**

3.4 Die Gleitkommadarstellung reeller Zahlen

Die reelle Zahl *z* hat folgende Gleitkommadarstellung (halblogarithmische Darstellung):

$$z = \pm m \cdot b^{\pm d}$$

m = Mantisse
d = Exponent
b = Basis des Exponenten; z.B. 2, 8, 10, 16

Hat man sich auf eine feste Basis geeinigt, kann man sie natürlich auch weglassen. Daher ist die übliche Rechnerdarstellung : $z = (\pm m\ ;\ \pm d)$
z.B.:

$$32{,}579 \cdot 10^7$$
$$= 3{,}2579 \cdot 10^8$$
$$= 32579 \cdot 10^4$$

<u>*Normalisierung:*</u>

$$\frac{1}{b} \le |m| < 1$$

z.B. : $b = 10 \Rightarrow \frac{1}{10} \le |m| < 1$ $\qquad$ $b = 2 \Rightarrow \frac{1}{2} \le |m| < 1$

Beispiel:

$$0{,}1101 \cdot 2^{ex}$$
$= 0{,}0000000000000001101$ ist aber nicht sonderlich gut gewählt. Besser:
$$= 0{,}1101 \cdot 2^{-13}$$
$$= 1{,}101 \cdot 2^{-14}$$

Man kann, wenn die erste Stelle immer eine eins sein soll, sich darauf einigen, auch noch diese wegzulassen.

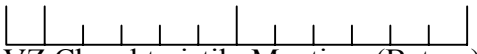
Die Exzeß-Darstellung:

Der Exponent wird durch Addition des Betrages vom kleinsten negativen Exponenten größer oder gleich Null gemacht. Man erreicht dadurch eine Verschiebung aller Exponenten in den positiven Bereich.
Der Exponent wird zur **Charakteristik**:

Charakteristik = Exponent + Exzeß

Beispiel:

Charakteristik hat 5 Stellen, Mantisse 6 Stellen ($b = 2$):



VZ Charakteristik Mantisse (Betrag)

Exzeß = $2^{e-1} - 1$; e = Stellenzahl des Exponenten

Konkretes Beispiel:

$5{,}75_{10} = 101{,}11$

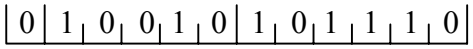
Normalisierung:

$5{,}75_{10} = 0{,}10111 \cdot 2^3$

mit Exzeß : $= 2^{5-1} - 1 = 15 = 01111$ (das ist kein Zufall!)

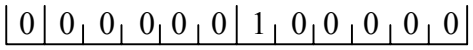
$\Rightarrow 5{,}75_{10} = 0{,}10111 \cdot 2^{18}$

Das sieht in der Graphik folgendermaßen aus:



Zahlenbereiche:

Die Betragsmäßig kleineste Zahl:



allgemein: $z_{10} = 0{,}5 \cdot 2^{-(2^{e-1}-1)}$ $\left[\frac{1}{2^1} = 2^{-1}\right]$

$= 2^{-1} \cdot 2^{-(2^{e-1}-1)}$

$= 2^{2^{e-1}}$

mit $e = 5 \Rightarrow z = 2^{-16}$ und das ist eine sehr kleine Zahl!

Die Betragsmäßig größte Zahl:

0	1	1	1	1	1	1	1	1	1	1	1	1

allgemein: $ex = 2^{e-1}$

$$z = \left(2^{-1} + 2^{-2} + \ldots + 2^{-m}\right) \cdot 2^{2^{e-1}} \cong 2^{2^{e-1}}$$

aufgrund der Konvergenzkriterien von Reihen ist der Klammerausdruck 1.

$$\text{Mit } e = 5 \Rightarrow z < 2^{2^4} = 2^{16} = 65536$$

Je mehr Bitstellen die Mantisse hat, je genauer wird die Zahl darin abgebildet. Für die Charakteristik gilt, daß mit vielen Stellen hier auch große Exponente stehen können, also die Zahl sehr viele Stellen vor oder nach dem Komma aufweisen kann.

Das IEEE-Format:

Wortlänge	VZ	C	M	
32	1	8	23	bit
64	1	11	52	bit

Normalisierung:

1,xxxxx und die erste eins wird weggelassen, da sie ja sowieso dort stehen muß. Näheres siehe Folie.

3.5 BCD-Zahlen als Schmankerl

BCD : Binary Coded Dezimal

0	0000	6	0110
1	0001	7	0111
2	0010	8	1000
3	0011	9	1001
4	0100	+	1010
5	0101	-	1011

```
     4739 = 0100 0111 0011 1001
```

Addition:

```
     4739 = 0100 0111 0011 1001
+    1287 = 0001 0010 1000 0111
-------------------------------
     6026 = 0101 1001 1100 0000
Korrektur:          0110 0110 0110
-------------------------------
           0110 0000 0010 0110
              6    0    2    6
```

4. Der Aufbau eines Computers

4.1 Die komponenten der Zentraleinheit

s. Kopie!

4.2 Der Befehlssatz und Addressierungsarten

Assembler:

Unter der Assemblersprache versteht man das Ersetzen der Codezahlen der Maschinensprache durch mnemonischen Code und auch das Übersetzertool, daß diesen mnemonischen Code wieder in die dazugehörigen Zahlen übersetzt. Mnemonisch bedeutet, daß man Akronyme wählt, die kurz und bündig den Befehl beschreiben. Also z.B. ADD, INC, JMP, MOV usw.
Diese kurzen Befehle ersetzen einen oder auch mehrere Codes der Maschinensprache, wobei die Vielfältigkeit nur durch kleine Veränderungen in der Verschlüsselung der Adressierungsarten und der verwendeten Register entsteht. Man könnte auch für jeden Maschinenbefehl ein neues Mnemonic erfinden, aber das ergibt sich meist auch aus dem Kontext.

Z.B.:

```
     ADD ax, 5
gegen ADD ax, word ptr [Adresse]
```

Der Unterschied in der Adressierungsart ist unverkennbar. Im ersten Fall wird eine direkte Zahl angegeben, wärend im zweiten Fall eine externe Referenz, also der Inhalt einer Speicheradresse verwendet wird. Trotzdem verwendet man nicht ADDR für ADdiereDiRekt und ADIDR für ADdiereInDiRekt als Mnemonic, sondern das eingängige ADD. Es gibt aber durchaus beide Arten.

Der Übersetzer wird ebenfalls als Assembler bezeichnet, da er im Gegensatz zu einem Compiler nicht mehrere Befehle für einen Übersetzungsschritt zusammenfasst (to compile=zusammenstellen) sondern nur in einer Tabelle nachsieht, wie der OP.-Code des Befehls sei und diesen einsetzt (to assemble=übersetzen).

Adressierungsarten:

Die **effektive Adresse** ist die Adresse, unter der das Datenwort, das angesprochen werden soll, tatsächlich steht.

Implizierte Adresse:
Ein Befehl spricht immer die selbe Adresse an, so daß sie nicht genannt werden muß. (z.B. CLC = Clear Carry: Im PSW wird das Bit des Carryflags gelöscht.)

Unmittelbare Adressierung:
Der Operant ist direkt als Wert angegeben.

direkte Adressierung:

Die effektive Adresse ist im Befehlswort enthalten. Der Wert steht direkt an dieser Adresse.

Indirekte Adressierung:

Im Operandenfeld steht die Adresse, unter der die effektive Adresse zu finden ist. Die indirekte Adressierung ist mehrfach möglich.

Basisregister Adressierung oder indizierte Adressierung:

Effektive Adresse ist = Basisregister + Operandenfeld. Das Operandenfeld enthält das Offset bezüglich des Basisregisters (-> Relative Adressierung)

4.3 Befehlssatz einschließlich CPUSIM-Übung

s. Kopie!

4.4 Peripherie

s. Referate und Kopien!

5. Schaltungen

5.1 Grundschaltungen

s. Kopie!

5.2 Schaltungsanalyse und Schaltungssynthese

Def.: Schaltungssynthese
In der *Schaltungssynthese* wird eine Schaltung aus einer Funktion bzw. der Wertetabelle entwickelt. Die Synthese erfolgt ggf. durch forgegebene Gatter (NOR- oder Nand-Gatter)

Def.: Schaltungsanalyse
Bei der *Schaltungsanalsys* wird aus einer gegebenen Schaltung die Wertetabelle oder die Funktion ermittelt.

5.3 Minimierung von Schaltungen

s.Kopie.

Es wird zur Schaltungsminimierung das Verfahren der Logischen Optimierung von Quine und McClusky angewendet

5.4 Schaltungsbeispiele

s.Kopie!

Aufgabe:

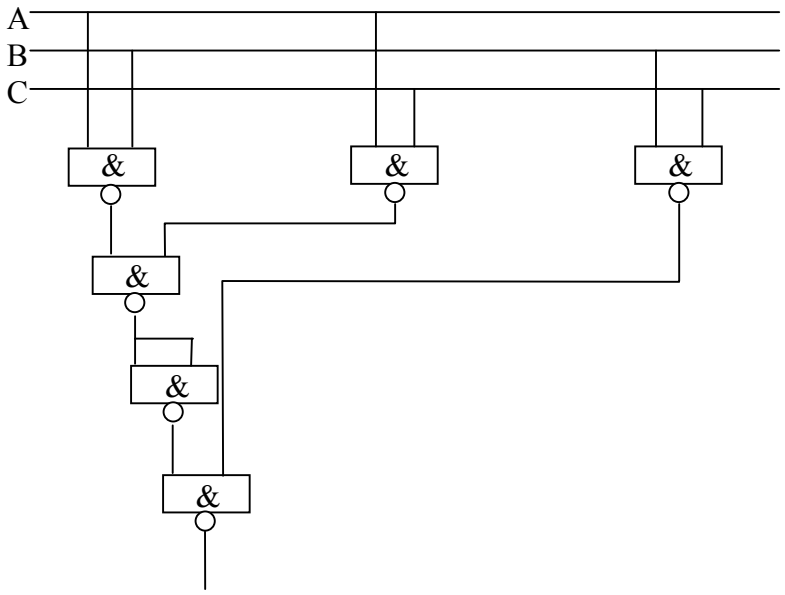
A	B	C	F
0	0	0	0
1	0	0	0
0	1	0	0
1	1	0	1
0	0	1	0
1	0	1	1
0	1	1	1
1	1	1	1

Die Aufgabe sei in einer Schaltung darzustellen und dabei seien nur NAND-Gatter zu verwenden:

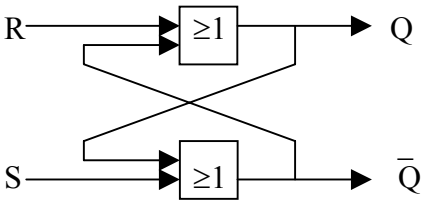
$$DNF = f(A,B,C) = \left(A \wedge B \wedge \overline{C}\right) \vee \left(A \wedge \overline{B} \wedge C\right) \vee \left(\overline{A} \wedge B \wedge C\right) \vee \left(A \wedge B \wedge C\right)$$
$$= \left(A \wedge B\right) \vee \left(A \wedge C\right) \vee \left(B \wedge C\right)$$
$$= \overline{\overline{\left(A \wedge B\right) \vee \left(A \wedge C\right) \vee \left(B \wedge C\right)}} = \overline{\overline{\left(A \wedge B\right)} \wedge \overline{\left(A \wedge C\right)} \wedge \overline{\left(B \wedge C\right)}}$$

Und für Später benötigen wir noch die Substitution: $\overline{x \wedge y \wedge z}$

Schaltung:



Das RS-Flip-Flop:



S	R	Q	$\overline{Q}$
1	0	1	0
0	0	1	0
0	1	0	1
0	0	0	1
1	1	0	0

Das RS-Flip-Flop kann, wie die Tabelle zeigt, nach dem Anlegen einer log. 1 auf R bzw. auf S einen Zustand annehmen, der auch nach der Wegnahme der Spannung immernoch erhalten bleibt. Die Leitung $\overline{Q}$ ist dabei nicht als Ausgangsleitung gedacht.

Durch die Speicherung des Zustandes eignet sich dieses Flip-Flop auch als Speichereinheit für ein Bit. Dazu ist aber noch eine Erweiterung der Schaltung mit einigen Und-Verknüpfungen nötig. Es werden die Leitungen Select und read/write über und-Gatter so eingebunden, daß die Selectleitung festlegt, ob eine Angelegte Spannung, die zu R oder S druchdringen soll, überhaupt durchkommen kann. Die read/write-Leitung sorgt ebenfalls für eine Sperre oder Entsperrung, so daß wahlweise der Inhalt des Flip-Flops über die Leitung Q abgegriffen werden kann, oder ob über R und S eine Spannung in das Flip-Flop fließt. Dann nennt man diese Schaltung eine Speicherzelle (Binary Cell, BC).

Diese BC kann man nun Cascadieren und in einer Matrix anordnen. (Dazu bitte auch die Folie betrachten.) Mit einem Decoder wird das Bitmuster einer Adressleitung verwendet, um eine entsprechende Selectleitung auszuwählen. Über diese Selectleitung wird eine Reihe von BCs (also z.B. genau 8 Stück für ein Byte) aktiviert oder scharf gemacht. Dann wird weiterhin über die read/write-Leitung festgelegt, ob gelesen oder geschrieben werden soll und demnach entweder das auf den Data-Input-Leitungen anliegende Byte eingespeichert, oder aber über die Data-Outputleitung der Inhalt ausgegeben. Dazu werden alle Outputleitungen übereinanderliegender BCs via OR-Gatter verknüpft, damit beliebig viele Zeilen auch genau eine Leitung ergeben.

Aufgabe:

x	y	z	f		
0	0	0	1	→	$\bar{x} \wedge \bar{y} \wedge \bar{z}$
1	0	0	1	→	$x \wedge \bar{y} \wedge \bar{z}$
0	1	0	0		
1	1	0	1	→	$x \wedge y \wedge \bar{z}$
0	0	1	0		
1	0	1	1	→	$x \wedge \bar{y} \wedge z$
0	1	1	1	→	$\bar{x} \wedge y \wedge z$
1	1	1	1	→	$x \wedge y \wedge z$

Nach der Optimierung folgt daraus:

$\Rightarrow DNF = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z) \vee (\bar{y} \wedge \bar{z})$ und als NOR dargestellt:

$\Rightarrow DNF = \overline{\overline{(x \wedge y)} \vee \overline{(x \wedge z)} \vee \overline{(y \wedge z)} \vee \overline{(\bar{y} \wedge \bar{z})}} = (\bar{x} \vee \bar{y}) \vee (\bar{x} \vee \bar{z}) \vee (\bar{y} \vee \bar{z}) \vee (y \vee z)$

Schaltbild: