Logik und Boolsche Algebra

Prof. Dr. Dörte Haftendorn: Mathematik mit MuPAD 4, 04.04.02 in Vers 2 und 3 genauso,
Update 7. juli 08
Web: http://haftendorn.uni-lueneburg.de www.mathematik-verstehen.de

---

Eine sprachliche oder mathematische **Aussage** ist ein Satz (oder ein mathematisches Gebilde),
zu dem der Begriff **wahr** oder der Begriff **falsch** gehört.
Die Regeln mit denen man aus Aussagen neue Aussagen bilden kann,
beschreibt man in der **Boolschen Algebra. (**Kurzform: "Rechnen mit Aussagen").
Bezeichnung nach George Boole 1854.
Computersprachen und Computer-Algebra-Systeme, CAS, beherrschen Boolsche Algebra.

Auswertung elementarer Aussagen (in MuPAD)

```
bool(3=4)
```

FALSE

```
bool(3<4)
```

TRUE

```
a:=3:  b:=4:
bool(a<b)
```

TRUE

Wenn a und b waren nicht bekannt sind, kann die letzte Ungleichung nicht ausgewertet werden.

Ein Satz oder mathematisches Gebilde mit Variablen heißt **Aussageform**,
wenn durch Belegung der Variablen eine Aussage entsteht.

Aussagenformen heißen auch **Prädikate,** der Umgang mit ihnen heißt **Prädikatenlogik.**
Diese wird in dieser Datei nur wenig vertieft.

Bezeichnung von Aussagen kann mit beliebigen Namen geschehen, meist nimmt man Großbuchstaben:

```
Unsinn:=3=4
```

3 = 4

```
bool(Unsinn)
```

FALSE

```
soIstEs:=3<4: bool(soIstEs)
```

TRUE

Im Folgenden sollen mit Aw, Bw, Cw.... wahre Aussagen und mit Af, Bf, Cf, ... falsche Aussagen bezeichnet werden.

```
Aw:= soIstEs: Af:= Unsinn:
Bw:= 10<=100: Bf:= 10>100:
Cw:=1<=1:Cf:=1<1:
```

Durch den := weist man zu, der die anderen Doppelpunkte unterdrücken die Ausgabe.

Durch den := weist man zu, der die anderen Doppelpunkte unterdrücken die Ausgabe.

Verknüpfungen von Aussagen, logische Operatoren

```
bool(3<4 and 10<=100)
```

TRUE

Das logische **Und** heißt in MuPAD und anderen Computersprachen **and , AND .....**
Als Zeichen ist es kleines Dach. Man schafft sich eine Übersicht mit einer **"Wahrheitstafel"**.
Hier ist eine Liste mit allen möglichen Kombinationen von wahren und falschen Aussagen.

```
[bool(Aw and Bw), bool(Aw and Bf),bool(Af and Bw),bool(Af and Bf
```

[TRUE, FALSE, FALSE, FALSE]

Die Und-Aussage ist also genau dann wahr, wenn beide beteiligten Aussagen wahr sind.
Eine Und-Aussage heißt auch **Konjunktion.**

Das logische **Oder** heißt in MuPAD und anderen Computersprachen **or , OR .....**
Als Zeichen ist es kleines v. Man schafft sich eine Übersicht mit einer **"Wahrheitstafel"**.
Hier ist eine Liste mit allen möglichen Kombinationen von wahren und falschen Aussagen.

```
[bool(Aw or Bw), bool(Aw or Bf),bool(Af or Bw),bool(Af or Bf)]
```

[TRUE, TRUE, TRUE, FALSE]

Die Oder-Aussage ist also genau dann falsch, wenn beide beteiligten Aussagen falsch sind.
Eine Oder-Aussage heißt auch **Disjunktion.**
"Genau dann" kann man auch als "dann und nur dann" aussprechen.

Die **Negation,** das logische **Nicht,** heißt in MuPAD und anderen Computersprachen **not , NOT .....**
Als Zeichen wird oft ein Haken verwendet oder über dem Buchstaben der Aussage wird ein Strich gemacht.

```
not(3<4)
```

$\neg\, 3 < 4$

Zusammensetzungen mit der Negation

Negiert man eine Und-Aussage erhält man folgendes:

```
not(A and B)
```

$\neg\, A \vee \neg\, B$

MuPAD "löst auf" : $\neg(A \wedge B) = \neg A \vee \neg B$, das 1. Gesetz von De Morgan
Man beweist ein solches Gesetz, indem man alle möglichen Belebungen von A und B mit wahren und falschen Aussagen prüft.
Übrigens: Der nicht-Operator bindet stärker als alle anderen logischenOperatoren.

```
bool(not(Aw and Bw)), bool(not(Aw and Bf));
bool(not(Af and Bw)),bool(not(Af and Bf));
```

FALSE, TRUE

TRUE, TRUE

```
TRUE, TRUE
```

```
bool(not Aw or not Bw), bool(not Aw or not Bf);
bool(not Af or not Bw),bool(not Af or not Bf);
```

```
FALSE, TRUE

TRUE, TRUE
```

Da sich dieselbe Wahrheitstafel ergibt, gilt das 1. Gesetz. von De Morgan. In Worten:
Die Negation einer Und-Aussage ist die Oder-Aussage mit den Negationen der einzelnen Aussagen.

Negiert man eine Oder-Aussage erhält man folgendes:

```
not(A or B)
```

$$\neg A \wedge \neg B$$

Die Negation einer Oder-Aussage ist die Und-Aussage mit den Negationen der einzelnen Aussagen.

MuPAD "löst auf" : $\neg(A \vee B) = \neg A \wedge \neg B$, das 2. Gesetz von De Morgan

```
bool(not(Aw or Bw)), bool(not(Aw or Bf));
bool(not(Af or Bw)),bool(not(Af or Bf))
```

```
FALSE, FALSE

FALSE, TRUE
```

```
bool(not Aw and not Bw), bool(not Aw and not Bf);
bool(not Af and not Bw),bool(not Af and not Bf)
```

```
FALSE, FALSE

FALSE, TRUE
```

Da sich dieselbe Wahrheitstafel ergibt, gilt das 2. Gesetz. von De Morgan.

```
bool(if Aw then Bw end_if), bool(if Aw then Bf end_if);
bool(TRUE),bool(TRUE)
```

```
TRUE, FALSE

TRUE, TRUE
```

```
bool(not Aw or Bw), bool(not Aw or Bf);
bool(not Af or Bw),bool(not Af or Bf)
```

```
TRUE, FALSE

TRUE, TRUE
```

Weitere logische Operatoren der Informatik

Da in der Informatik Schaltelemente für die logischen Operatoren konstruiert werden, sind auch die Zusammensetzungen **NichtUnd = nand=Nand = NAND=...** und **NichtOder=nor=Nor=NOR=...** gebräuchlich.
In machen Sprachen sind sie implementiert, in MuPAD kann man sie sich selbst definieren:

```
nand:=(A,B)->not(A and B)
```
$(A, B) \to \neg (A \wedge B)$
```
nor:=(A,B)->not(A or B)
```
$(A, B) \to \neg (A \vee B)$

Probe an einem Beispiel, dass das richtig funktioniert:

```
bool(nand(Aw,Bw)), bool(nand(Aw,Bf));
bool(nand(Af,Bw)),bool(nand(Af,Bf))
```
FALSE, TRUE

TRUE, TRUE

```
bool(nor(Aw,Bw)), bool(nor(Aw,Bf));
bool(nor(Af,Bw)),bool(nor(Af,Bf))
```
FALSE, FALSE

FALSE, TRUE

Aufbau der Logik mit NAND-Bausteinen

Wenn man die logischen Operatoren Nicht, Und und Oder mit Nand bauen kann, dann lässt sich jede logische
Schaltung allein aus Nand-Bausteinen aufbauen:

```
[nand(A,A),not A] //Aufbau von Nicht
```
$[\neg A, \neg A]$
```
[nand(nand(A,B),nand(A,B)),not(nand(A,B)), A and B]//Aufbau von
```
$[A \wedge B, A \wedge B, A \wedge B]$
```
[nand(nand(A,A),nand(B,B)),nand(not A, not B), A or B]//Aufbau v
```
$[A \vee B, A \vee B, A \vee B]$

Rechenregeln für logische Terme, Distributiv-Gesetze

```
(A and ( B or C))= ((A and B) or (A and C))
```
$(A \wedge (B \vee C)) = ((A \wedge B) \vee (A \wedge C))$

Ein Beispiel für die Richtigkeit:

```
[bool(Aw and ( Bw or Cf)),
bool((Aw and Bw) or (Aw and Cf))]
```
[TRUE, TRUE]

```
[bool(Aw and ( Bf or Cf)),
bool((Aw and Bf) or (Aw and Cf))]
```
[FALSE, FALSE]

Das 2. Distributivgesetz

```
(A or ( B and C))= ((A or B) and (A or C))
```
$(A \vee (B \wedge C)) = ((A \vee B) \wedge (A \vee C))$

Ein Beispiel für die Richtigkeit:

```
[bool(Aw or ( Bw and Cf)),bool((Aw or Bw) and (Aw or Cf))]
```
[TRUE, TRUE]

[TRUE, TRUE]

Für einen Beweis müsste man alle 8 Belegungen durchchecken.

Verwendung der logischen Terme in der Informatik

```
if 3=4 then doof else drei_nicht_vier end_if;
if 3<4 then kleiner_ok else ganzFalsch end_if;
```

$drei_{nicht_{vier}}$

$kleiner_{ok}$

Zusammengesetzte Abbruchbedingungen in Schleifen.

```
i := 1:   r := 10.0:
while i < 5 or r >1.5 do
   print(i ,r):
  r := sqrt(r):
  i := i + 1:
end_while:
```

```
1, 10.0
2, 3.16227766
3, 1.77827941
4, 1.333521432
```

```
i := 1:r:=10.0:
repeat
  print(i,r);
  i := i + 1; r:=sqrt(r):
until i >= 5 or r <1.5  end:
```

```
1, 10.0
2, 3.16227766
3, 1.77827941
```

```
i := 1:r:=10.0:
repeat
  print(i,r);
  i := i + 1; r:=sqrt(r):
until i >= 5 and r <=1.5  end: //Negation der While-Bedingung
```

```
1, 10.0
2, 3.16227766
3, 1.77827941
4, 1.333521432
```

```
r:=10.0: //Mit Zählschleife
for i  from 1 to 5   do
 print(i,r);
r:=sqrt(r):
end_for:
```

```
1, 10.0
2, 3.16227766
3, 1.77827941
4, 1.333521432

5, 1.154781985
```

```
5, 1.154781985
```

Direkte Boolsche Werte

```
if TRUE then Aw else Af end_if
```

```
3 < 4
```

```
if FALSE then Aw else Af end_if
```

```
3 = 4
```

```
not TRUE and TRUE, not TRUE or TRUE
```

```
FALSE, TRUE
```

In einigen (älteren) Computersprachen ist TRUE=1 und FALSE=0.
Variablen für logische Ausdücke (=logische Terme=boolsche Ausdrücke) sind vom Datentyp **boolean.**
Manchmal wird für UND ein Malzeichen und für ODER ein Pluszeichen geschrieben.
Das passt dazu, dass in allen Computersprachen UND Vorrang vor ODER hat.