Projektarbeit Mathe

Inhalt

Einleitung

Im Wahlpflichtfach „MATLAB II“ wurden, aufbauend auf den Inhalten von Mathe IV, weitere Funktionalitäten des Programmes MATLAB behandelt. Als Prüfungsleistung war eigenständig eine komplexe Aufgabenstellung zu bearbeiten.

Aufgabenstellung

Die Aufgabe bestand darin, einen Aussagenlogischen Ausdruck auszuwerten. Dabei sollte das Programm Ausdrücke mit mindesten vier aussagenlogische Variablen unterstützen können. Im Auswerteprozess soll der Ausdruck als Zeichenkette ausgegeben werden. Dabei sind die Operatoren wie folgt darzustellen:

nicht a	~a
a und b	a&b
a oder b	a|b
wenn a, so b	a->b
a genau dann, wenn	a<->b
entweder a oder b	a#b

Die Eingabe des Ausdrucks soll als String erfolgen, die Form ist jedoch offengelassen.

Wahl der Software

Im Kurs wurde die Software MATLAB genutzt. Der Name leitet sich dabei von MATrix LABoratory ab. Mit MATLAB lassen sich also komplexe Matrizen-Operationen durchführen, es bietet allerdings auch diverse andere Möglichkeiten, um komplexe numerische Rechnungen zu lösen. Das ist insbesondere bei Iterativen Rechnungen wie Simulationen sinnvoll, kann aber auch in vielen anderen Bereichen ein Ergebnis mit einer ausreichenden Genauigkeit liefern. Die Erstellung von MATLAB Programmen erfolgt über die Programmierung über das GUI (Guided User Interface) und bietet neben der Deklaration von numerischen Variablen (a=2) oder dem Ausführen mathematischer Operatoren (a=2+3) auch die Möglichkeit eingaben vom Benutzer entgegenzunehmen oder Schleifen aus zu führen[1].

MATLAB ist ein proprietäres Programm von der Firma „The MathWork“. Ich habe mich darum für die Bearbeitung des Projektes für die Software „GNU Octave“ entschieden. Octave bietet in der Funktion die gleichen Möglichkeiten wie MATLAB, wird jedoch von der Community spendenfinanziert unter der „GNU General Public License“ entwickelt[2]. Der Quellcode von Octave ist damit für jeden einsehbar. Das hat den Vorteil, dass Fehler in der Software mit dem nötigen Wissen von dem Anwender selbst behoben werden können. Außerdem wird so nachvollziehbar, wie das Programm auf die Ergebnisse kommt, die es ausgibt. Zudem führt es zu einer niedrigeren Einstiegshürde für den Nutzer. So kann man ohne hohe Investitionskosten Programme, die mit Octave erstellt wurden, bearbeiten was zu einem breiten Zugang zu Wissen, insbesondere in weniger entwickelten Ländern führt. Für das Projekt habe ich Octave in der Version 6.4.0 (Windows) bzw. 5.2.0 (Linux) genutzt.

Aussagenlogik

„Eine Aussage ist die gedankliche Widerspiegelung eines Sachverhaltes in Form eines Satzes einer natürlichen oder künstlichen Sprache.“ (Bronstein 2000:297). Jede Aussage kann dabei entweder wahr

[1] https://www.mathworks.com/help/pdf_doc/matlab/rn.pdf
[2] https://www.gnu.org/software/octave/about

oder falsch sein (1 oder 0). In jeder Aussage gibt es Variablen die selbst entweder wahr oder falsch sein können. Variablen können mit Operatoren verknüpft sein. Durch diese Verknüpfung ergibt sich eine Aussage, die je nach den Wahrheitswerten der einzelnen Variablen Wahr oder Falsch sein kann. Um nun zu prüfen, ob ein Ausdruck Wahr ist, muss der Ausdruck mit jeder Kombination von Wahrheitswerten der Variablen geprüft werden[3].

So lässt sich der aussagenlogische Ausdruck „a und b“ wie folgt auswerten:

a	b	a&b
0	0	0
0	1	0
1	0	0
1	1	1

Anhand der Tabelle erkennt man, dass der Ausdruck nur dann wahr ist, wenn a und b wahr sind. Auf diese Weise kann man beliebig lange Ausdrücke mit beliebig vielen Variablen auswerten.

Projektdurchführung

Weil die Eingabe des Aussagenlogischen Ausdrucks in der Aufgabe nicht definiert wurde, habe ich mich dazu entschieden, die Eingabe in einem „alltagsdeutsch“ zu gestalten. Es sollen also Namen oder Gegenstände als Variablen dienen. Bei der Umsetzung in der Programmierung müssen die Operatoren eindeutig erkannt werden. Außerdem muss das Programm wissen, welche Worte als variablen dienen. Das ist allerding bei der Alternation schwierig, weil „entweder a, oder b“ ebenfalls den Operator „oder“ enthält. Darum ist es schwierig, Operatoren, die aus mehr als einem Wort bestehen klar zu erkennen. Ich habe mich darum entschieden, die Operatoren, die aus mehr als einem Wort bestehen so zu verändern, dass sie nur aus einem Wort bestehen, bei dem vor und hinter dem Operator die jeweilige Variable steht:

Alter Operator:	Operator in kurzschreibweise:	Neuer Operator:
nicht a	~a	nicht a
a und b	a&b	a und b
a oder b	a|b	a oder b
wenn a, so b	a->b	a impliziert b
a genau dann, wenn	a<->b	a gleichwertig b
entweder a oder b	a#b	a alterniert b

Somit wäre eine mögliche Eingabe „*bier oder nicht schnitzel impliziert bier und schnitzel oder wein*“

Mit den Namen der Variablen: „*bier*“, „*schnitzel*“ und „*wein*“.

Diese Aussage muss nun in die Kurzschreibweise übertragen werden. Außerdem muss jede mögliche Kombination der Variablen geprüft werden, um dann den Ausdruck auszuwerten.

Es soll ebenfalls eine Option sein, einen Aussagenlogischen Ausdruck mit Klammern in Kurzschreibweise einzugeben, um komplexere Ausdrücke auswerten zu können.

[3] Vgl. Bronstein, Taschenbuch der Mathematik, S.297

Um eine einfachere Programmierung zu gewährleisten habe ich die einzelnen Teile des Programms in Funktionen unterteilt.

Eine Funktion ist ein eigenständiges Programm, das mit Parametern aufgerufen wird. Diese Parameter müssen in einer Form sein, die die Funktion erkennt. Eine Funktion kann beliebig viele Parameter haben. In der Funktion werden dann dinge berechnet, die an den Programmteil zurückgegeben werden, der die Funktion aufgerufen hat. Auch hier kann eine Funktion beliebig viele Rückgabewerte besitzen.

Funktion „stringInAussagenlogik“

An die Funktion wird ein Cell-Array mit den Namen der Variablen und der Aussagenlogische Ausdruck als String übergeben.

In der ersten Schleife der Funktion werden die Stellen der Variablen in dem Text gesucht. Weil Variablen unterschiedlich oft vorkommen können, müssen die Stellen, an denen die Variablen auftauchen, in einem weiten Cell-Array gespeichert werden. Außerdem werden in den Vektor „variablen“ die jeweiligen Anfangsbuchstaben der Variablen geschrieben.

Der aussagenlogische Ausdruck wird nun nach den Operatoren durchsucht. Auch hier wird die Position, an denen ein Operator genutzt wird, in einem Cell-Array gespeichert. Über eine for-Schleife wird jedes Zeichen des Aussagenlogischen Ausdrucks durchlaufen. Bei jedem Zeichen wird sowohl das Cell-Array der Variablen als auch das der Operatoren durchlaufen. Kommt entweder ein Operator oder eine Variable an der jeweiligen Stelle im Ausdruck vor, wird an diese Stelle das jeweilige Kurzzeichen geschrieben. Ist die for-Schleife vollständig durchgelaufen wurden alle Worte durch die jeweiligen Kurzschreibweisen ersetzt. In einer zweiten Schleife werden alle ungenutzten Zeichen gelöscht.

Die Funktion gibt den Vektor der Variablen und den neu entstandenen Aussagenlogischen Ausdruck zurück.

Übergibt man den oben genannten Aussagenlogischen Ausdruck

„*bier oder nicht schnitzel impliziert bier und schnitzel oder wein*“

mit den Variablen

{"*bier*";"*schnitzel*";"*wein*"}

And die Funktion, so gibt sie den Aussagenlogischen Ausdruck in kurzschreibweise

„b|~s->b&s|w“

sowie die Variablen

„bsw „

zurück.

Funktion „klammernauswerten“

Wird die Funktion in normalem deutsch eingegeben geht das Programm davon aus, dass Klammern nicht vorkommen. Es gibt allerdings auch die Möglichkeit, einen Ausdruck in einer kürzeren, abstrahierten Art einzugeben. Sie muss aufgebaut sein, wie die Ausgabe aus der Funktion „stringInAussagenlogik“. Wenn der Ausdruck allerdings so eingegeben wird, kann er auch Klammern

enthalten. Diese Klammern verändern die Priorität der Operatoren. Ausdrücke in Klammern müssen zuerst ausgewertet werden. Mit dem Ergebnis kann dann weiter gerechnet werden.

An die Funktion wird der Auszuwertende String (entweder mit Klammern oder ohne Klammern) übergeben. Außerdem wird ein Vektor mit den Variablen und ein weiterer Vektor mit den Werten, die die Variablen haben, übergeben. Die Funktion gibt dann entweder eine logische eins zurück, wenn der Ausdruck mit den eingesetzten Variablen wahr ist, oder eine logische 0, wenn der Ausdruck falsch ist.

Ein möglicher Aufruf der Funktion würde wie folgt aussehen:

klammernauswerten(‘a&(b|c)‘,[‘a‘,‘b‘,‘c‘],[1,1,0])

Die Funktion soll also den Ausdruck a&(b|c) mit den werten a=1, b=1 und c=0 auswerten.

Dafür ersetzt die Funktion als erstes die Variablen mit ihren jeweiligen Werten:

1&(1|0)

Anschließend wird der String wird nach dem Zeichen ‚)‘ durchsucht. Es wird davon ausgegangen, dass es für jede geschlossene Klammer ebenfalls eine vorherige geöffnete Klammer gibt. Die Position der geschlossenen Klammern wird in einem Vektor gespeichert. In diesem Fall hätte der Vektor an der ersten Position den Wert sieben, weil die geschlossene Klammer an der siebten Stelle im String auftaucht.

Kommen in dem Ausdruck keine Klammern vor, so wird die Funktion „logikAuswerten“ direkt mit dem oben entstandenen String aufgerufen. Kommen Klammern vor, wird der Vektor, in dem die jeweiligen Positionen der geschlossenen Klammern gespeichert sind, durchgegangen. Von der ersten geschlossenen Klammer wird nach der vorangegangenen geöffneten klammer gesucht. Der Teilausdruck, der von den Klammern eingeschlossen wird (in diesem Fall ‘1|0‘) wird mit der Funktion „logikAuswerten“ Ausgewertet. Das Ergebnis wird an die Stelle der geschlossenen Klammer geschrieben und alle Zeichen davor bis einschließlich zur geöffneten Klammer werden gelöscht. Außerdem wird die Stelle jeder weiteren geschlossenen Klammer um die Anzahl der gelöschten Zeichen verringert.

Nachdem die Schleife die erste Klammer in dem Beispiel aufgelöst hat, ergibt sich der Ausdruck:

1&1

Gäbe es weitere Klammern würden sie nach dem gleichen Prinzip aufgelöst werden, bis keine Klammer mehr vorhanden ist. Der Ausdruck, der nun keine Klammern mehr enthält, wird ein letztes Mal mit der Funktion „logikAuswerten“ Ausgewertet. Das Ergebnis wird an die Aufrufende Funktion übergeben. In diesem Beispiel also die Logische eins.

Funktion logikAuswerten

Um die aussagenlogischen Ausdrücke auszuwerten, habe ich eine weitere Funktion erstellt. An sie muss der String in kurzschreibweise, mit den eingesetzten Wahrheitswerten für die einzelnen Variablen, übergeben werden. Ein Aufruf der Funktion würde mit dem Beispiel so aussehen:

logikAuswerten(‘0|~0->0&0|1 ‘)

Die Funktion soll dann entweder eine logische 1 zurückgeben, wenn der Ausdruck wahr ist oder eine logische 0, wenn der Ausdruck falsch ist.

Der gesamte Ausdruck für jeden Operator durchgegangen. Die Reihenfolge ist dabei durch die Operatoren Priorität gegeben[4]:

1. Negation
2. Konjunktion
3. Disjunktion
4. Konditional
5. Bikonditional

In einer while-Schleife wird nacheinander jedes Zeichen in dem aussagenlogischen Ausdruck durchgegangen. Entspricht das Zeichen dem Operator, wird der Operator mit der „eval"-funktion ausgewertet. So arbeitet sich das Programm für jeden Operator durch den gesamten String. Nachdem jede Negation angewandt wurde, sieht der String folgendermaßen aus:

„0|1->0&0|1"

Nun wird dieser String ein weiteres Mal durchlaufen, um die Konjunktionen anzuwenden. Jedes Zeichen wird mit dem und-Operator „&" verglichen. Ist das Zeichen im String gleich dem „&" Zeichen, wird der String „temp" erzeugt. Er ist ein Zeichenvektor mit dem Zeichen, das vor dem Operator kommt, dem Operator und dem Zeichen nach dem Operator. Also:

temp='0&0'

Mit der „eval" Funktion kann dieser String nun ausgewertet werden. Das Ergebnis (in diesem Fall „0") wird an die Stelle des Operators geschrieben. Das Zeichen vor und nach dem Operator wird gelöscht. Analog hierzu werden die anderen Operatoren der Reihe nach ausgewertet:

„0|1->0|1"

„1->1"

„1"

Bei den Operatoren der Implikation, Äquivalenz und Alternation konnte nicht auf bereits vorhandene Funktionen zurückgegriffen werden. Sie wurden also in separaten Funktionen implementiert.

Wenn der String nach jedem Operator durchsucht wurde und der resultierende Ausdruck ausgewertet wurde, bleibt ein einzelner Wert über, der Aussagt, ob der Ausdruck mit den übergebenen Werten für die Variablen wahr oder falsch ist. Dieser Wert wird an die aufrufende Funktion übergeben.

Erstellung der Wahrheitstabelle-allgemeiner Ansatz

In der Wahrheitstabelle muss jede mögliche Kombination aus Variablen geprüft werden. Dabei ist es üblich, in der ersten Spalte in jeder Zeile den Wert zu wechseln, in der zweiten Spalte jede zweite Zeile, in der dritten Spalte jede vierte Zeile usw. Die einfachste Möglichkeit so einen aussagenlogischen Ausdruck auszuwerten ist das Nutzen von for-Schleifen. Dies würde für den Ausdruck „a|b&c" wie folgt aussehen:

```
clc
clear all
logik=[];
```

[4] Vgl. Jürgen Weiß, Taschenbuch der Mathematik, Band 1, S. 115–120

```
n=1;
for a=0:1
  for b=0:1
    for c=0:1
    logik(n,1:3)=[a,b,c];
    logik(n,4)=a|b&c;
    n++;
    endfor
  endfor
endfor
logik
```

```
    logik =

       0   0   0   0
       0   0   1   0
       0   1   0   0
       0   1   1   1
       1   0   0   1
       1   0   1   1
       1   1   0   1
       1   1   1   1
```

Die schleifen laufen durch jede Kombination der Variablen a,b und c, diese werden in die ersten drei Spalten der Matrix geschrieben. In die vierte wird das Ergebnis der logischen Auswertung geschrieben. Um diese Vorgehensweise anwenden zu können muss man allerdings wissen, wie viele Variablen in dem aussagenlogischen Ausdruck vor Kommen. Die Variablen müssen außerdem in Octave als solche gespeichert werden. Das oben gezeigt Programm funktioniert also nur bei drei Variablen die a, b und c heißen. Ich wollte mir offenlassen, wie viele Variablen in dem Ausdruck vor Kommen und wie diese heißen. Die oben umrissene Methode ist also für mich nicht sinnvoll. Für meine Anwendung benötige ich eine andere Möglichkeit, um auf die Werte der Wahrheitsvariablen zu kommen. Sie sollen dabei das gleiche Muster haben wie mit der Methode oben.

Exkurs: Binäres Zählen

Die kleinste Speichereinheit in einem Computer ist das Bit. Es kann entweder den zustand 1 oder den zustand 0 annehmen. Damit ein Computer mit zahlen arbeiten kann muss eine Zahl in mehreren Bits abgespeichert werden. Dabei haben die einzelnen Bits unterschiedliche Wertigkeiten:

2^0 2^1 2^2

Sind in diesem Beispiel alle drei Bit aktiv, so wird die Zahl

$1*2^0+1*2^1+1*2^2=7$

dargestellt.

Ist keines der 3 Bit aktiv stellt das die Zahl 0 dar. Mit 3 Bit lassen sich also Zahlen von 0 bis 7 darstellen. Wenn man nun alle Zahlenwerte eines 3-Bit-Zählers darstellt und die aktiven Bits mit einer 1 und die inaktiven Bits mit einer 0 markiert ergibt sich folgende Tabelle:

Dargestellte Zahl	2^0	2^1	2^2
0	0	0	0
1	1	0	0

2	0	1	0
3	1	1	0
4	0	0	1
5	1	0	1
6	0	1	1
7	1	1	1

Man erkennt, dass sich beim binären Zählen das gleiche Muster wie bei den Wahrheitstabellen ergibt. Dabei ist die Anzahl der gebrauchten Variablen analog zu der Anzahl der Bits beim Zähler. Die Reihenfolge der Bit Werte, also, ob das höchstwertige Bit (2^2) an erster oder letzter Stelle steht, ist in vielen Programmiersprachen davon abhängig, ob es sich bei dem System um ein „Big-Endian-System“ oder „Little-Endian-System“ handelt. Bei MATLAB und Octave werden die Bit Werte allerdings immer nach dem Big-Endian-System erzeugt. Ich habe mich in der obigen Darstellung ist für ein Little-Endian-System entschieden.

Erzeugung der Wahrheitstabelle-konkreter Ansatz

Um die Wahrheitstabelle zu erzeugen, wird in einer while-Schleife so lange hoch gezählt, bis der maximale Wert (2^variablen) erreicht ist. Bei jedem durchlauf wird mit der Funktion „dec2bin“ die aktuelle schleifen Iteration in einen binären Wert umgewandelt. Die Funktion gibt char-Werte zurück. Diese müssen in logische Werte umgewandelt werden und können dann in einer Matrix gespeichert werden. Ist das Geschehen kann an die Funktion „klammernauswerten“ die Zeile der Matrix mit den jeweiligen Wahrheitswerten, dem verkürzte String des aussagenlogischen Ausdrucks sowie den Namen der Variablen übergeben werden. Das Ergebnis der Funktion wird dann in die letzte Spalte der Matrix gespeichert und sagt aus, ob der Ausdruck wahr oder falsch ist.
Ist jede mögliche Kombination durchlaufen wird die Matrix mit allen Variablenkombinationen sowie der Auswertung ausgegeben.

Erzeugt man sich die Wahrheitstabelle zu dem obigen Beispiel

"bier oder nicht schnitzel impliziert bier und schnitzel oder wein"

So erhält man das folgende Ergebnis:

```
wahrheitswerte =

   0   0   0   0
   0   0   1   1
   0   1   0   1
   0   1   1   1
   1   0   0   0
   1   0   1   1
   1   1   0   1
   1   1   1   1
```

Die Aussage ist also nur dann nicht wahr, wenn alles falsch ist, oder dann, wenn nur Bier wahr ist. In allen anderen Fällen ist sie immer wahr. Über die Tatsächliche Aussage des Ausdrucks muss man dann eine kontextuelle Aussage treffen.

Verbesserung der Nutzerfreundlichkeit

Um die Nutzerfreundlichkeit zu verbessern, gibt es die Möglichkeit, den aussagenlogischen Ausdruck in der Konsole einzugeben. Dabei ist darauf zu achten, dass die Operatoren klein geschrieben werden. Das Programm versteht zudem nur die oben genannten Operatoren. Der Text muss nicht in Anführungszeichen gesetzt werden. Wenn man einen String in kurzschreibweises Eingeben möchte, ist darauf zu achten, dass man bei der ersten Eingabeaufforderung das Zeichen ‚0' eingibt. So gelangt man in das Eingabefenster für die Ausdrücke in Kurzform. Bestätigt man mit „Enter" wird man dazu aufgefordert die Variablen einzugeben. Sie müssen ebenfalls nicht in Anführungszeichen gesetzt werden. Außerdem darf man nicht von der Schreibweise des vorher eingegebenen Textes abweichen. Man hat jede Variable mit „Enter" zu bestätigen. Hat man alle genutzten Variablen eingegeben beendet man die Eingabeaufforderung durch das Eingeben der „0" und anschließendes bestätigen mit „Enter". Das Programm wertet anschließend den Ausdruck aus und gibt ihn in der oben beschriebenen Form zurück.

Wenn man die aussagenlogischen Ausdrücke und die Variablen lieber im Programmcode eingibt, kann man die Variable „texteingabe_ueber_console" von 1 auf 0 setzen. Möchte man dann den Ausführlichen Text eingeben ist die Variable „eingabe_in_alltagssprache" in Zeile 73 auf „1" setzen. Dann kann in Zeile 77 der Text, und in Zeile 78 die Variablen eingegeben werden. Ist die Variable „eingabe_in_alltagssprache" auf „0", so kann man in Zeile 85 den String und in Zeile 86 die Variablen eintragen.

Anwendung des Programms

Aussagenlogik ist Teil von Ing. Mathe I. Eine der Übungsaufgaben zum 31.10.2019 lautete:

„Stellen Sie für die folgenden Aussagen aussagenlogische Ausdrücke auf:

a) Wenn die Ernte gut war, ist das Brot billig
b) Das Brot ist teuer, wenn die Ernte schlecht war
c) Wenn das Brot nicht billig ist, war die Ernte schlecht
d) …"

Außerdem war zu untersuchen, welche der Aussagen logisch gleichwertig sind.

Um das Programm anwenden zu können müssen als erstes die variablen definiert werden.

Am einfachsten ist es, gute Ernte als „erntegut", sowie billiges Brot als „brotbillig" zu definieren. Nun müssen die Ausdrücke auf die im Programm verwendeten Operatoren angepasst werden:

„erntegut impliziert brotbillig"

„nicht erntegut impliziert nicht brotbillig"

„nicht brotbillig impliziert nicht erntegut"

e->b	~e->~b	~b->~e
0 0 1	0 0 1	0 0 1
0 1 1	0 1 0	0 1 1
1 0 0	1 0 1	1 0 0
1 1 1	1 1 1	1 1 1

Man erkennt so, dass a gleichwertig mit c ist (die Letzten Spalten sind identisch).

Aufgabe 2 vom 07.11.2019 lautet so:

„Worin besteht das Geheimnis Ihres langen Lebens?“ wurde ein Hundertjähriger gefragt. „Ich halte mich streng an die Diätregel: Eis esse ich höchstens, wenn ich vorher Fleisch hatte. Fisch esse ich dann und nur dann, wenn ich auch Eis bestelle. Wenn ich kein Bier trinke, so nehme ich immer Fisch. Entweder esse ich Fleisch und verzichte auf Eis oder ich bestelle weder Bier noch Fisch.

Der Fragesteller fand diesen Ratschlag ziemlich verwirrend. Können Sie ihn vereinfachen?

Eine Vereinfachung wäre in diesem Fall sicherlich sinnvoll. Man kann jedoch auch einfach den Ausdruck auswerten. Dafür können die einzelnen Sätze mit und-Operatoren verbunden werden. Es ergibt sich der Folgende Ausdruck:

Eis=a

Bier=b

Fleisch=c

Fisch=d

(c->a)&(c<->d)&(~b->d)&((c&~a)#(~b&d))

Nun lässt man sich zu diesem Ausdruck die Wahrheitstabelle ausgeben:

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

```
1  1  0  0  0
1  1  0  1  0
1  1  1  0  0
1  1  1  1  0
```

Man erkennt an der letzten Spalte, dass die Aussage nur mit einer möglichen Kombination wahr ist. Das Geheimnis des hundertjährigen ist also, dass er zu jeder Mahlzeit Eis, Fleisch und Fisch isst und dabei auf Bier verzichtet.

Fazit

An dem letzten Beispiel erkennt man gut, dass die Anzahl an auszuwertenden Kombinationen exponentiell steigt. Bei zwei Variablen hat man nur 4 Kombinationen auszuwerten. Im letzten Beispiel muss man bei vier Variablen schon 16 mögliche Kombinationen Auswerten. Gäbe es nur eine mehr wären es schon 32 Kombinationen usw. Dazu kommt, dass die Ausdrücke mit mehr Variablen eher komplexer werden. In Anbetracht all dessen ist es sehr sinnvoll, solche Ausdrücke mit einem Computerprogramm auswerten zu lassen.