```
10   OPEN 1,1,1
20   FOR I=100 TO 300
30   PRINT#1,I
35   IF PEEK(166)=18 THEN POKE 166,191
40   NEXT
50   CLOSE 1
```

Wir haben jetzt die Abfrage der Speicherzelle 166 des Programms # 2 von vorhin in das Programm # 1 eingebaut. Spulen Sie bitte das Band zurück und lassen Sie das Programm laufen.

Nun wollen wir die dadurch neu abgespeicherte Datei ganz normal auslesen. Dazu nehmen wir das Programm #2, also ohne die Zeile 45. Das sieht dann so aus:

Programm #2.a

```
10 OPEN 1,1,0
20 GET#1,X$
30 PRINT X$;
40 PRINT CHR$(28)PEEK(166)CHR$(154);
50 GOTO 20
```

Wir starten es mit RUN, nachdem das Band wieder zurückgespult ist. Der Vorgang ist im Prinzip der gleiche wie bei Programm #2; halten Sie das Programm bitte auch wieder an, so wie vorher.

Wir sehen aber einen großen Unterschied im Ausdruck. Es erscheinen nur die ersten drei Zahlen, 100 bis 102, danach steht nichts mehr im ganzen Block, bis der Inhalt von 166 die Endzahl 190 erreicht hat. Erst danach, nach dem Loslaufen des Kassettenmotors und dem Einlesen des nächsten Schubes, erscheinen die nächsten drei Zahlen.

Schlußfolgerung:

Durch POKEn der Zahl 191 in die Speicherzelle 166 zu einem beliebigen Zeitpunkt können wir sowohl beim Abspeichern, als auch beim Einlesen einer Datei dem Computer vorgaukeln, der Kassetten-Puffer sei bereits abgearbeitet. Dadurch wird der Kassettenmotor eingeschaltet und der nächste Schub ein- beziehungsweise ausgelesen.

Texteinschub #3 Fehlererkennung mit Parity-Bits

Bei der Datenübertragung zwischen Peripheriegeräten, insbesondere zwischen Datasette und dem Computer kommt es recht häufig vor, daß Fehler auftreten. Diese Fehler haben alle möglichen Ursachen und trotz aller Anstrengungen der Ingenieure lassen sie sich leider nicht völlig vermeiden.

An besonderen Schwachstellen werden daher Maßnahmen getroffen, um Fehler wenigstens zu erkennen und Programme abzubrechen, bevor größerer Schaden entsteht. Die mißlichen »LOAD ERROR«-Meldungen sprechen da eine deutliche Sprache. Die einfachste Art, Fehler zu erkennen — ich sollte genauer sagen: einzelne Bitfehler zu erkennen — geschieht über sogenannte »Parity-Bits«. Die Methode besteht darin, daß zu einem Datenwort, zum Beispiel einem Byte, ein zusätzliches Bit hinzugefügt wird und zwar so, daß die Quersumme immer eine gerade oder auch eine ungerade Zahl ergibt.

Bevor ein Wort übertragen wird, errechnet der Sender das Parity-Bit und fügt es dem Wort als zusätzliches Bit hinzu. Der Empfänger, der diese Prüfmethode natürlich auch kennen muß, rechnet die Quersumme aus. Wenn sie stimmt, nimmt er das Parity-Bit weg und arbeitet mit dem richtigen Wort weiter. Wenn die Quersumme nicht stimmt, schlägt er Alarm.

Sie werden sicher schon bemerkt haben, daß in meinem Beispiel natürlich ein Doppelfehler, nämlich zwei falsche Bits, natürlich nicht erkannt werden. Um das zu erreichen, müßte man zwei Parity-Bits einführen. Sie sehen natürlich auch, wohin das letztlich führt, nämlich zu einer Vergrößerung der Wortlänge. Man nennt das auch »Redundanz«, vielleicht haben Sie dieses Wort schon einmal gehört. Nun, da gibt es für jeden Anwendungsfall ein Optimum, abhängig von der Wahrscheinlichkeit, welche Art von Fehlern in welcher Häufigkeit auftreten. Im Extremfall gibt es Codiersysteme — zu denen die Parity-Bit-Methode auch gehört — welche in der Lage sind, Fehler nicht nur zu erkennen, sondern gleich zu korrigieren.

Logeleien (Teil 3)

Diese letzte Folge der Logeleien wird zum krönenden Abschluß unseren Commodore 64 als »Schlußfolgerungs-Maschine« vorstellen und den Zusammenhang mit der sogenannten Künstlichen Intelligenz beleuchten.

Das Aschenputtel unseres Basic-Sprachvorrates, den WAIT-Befehl, hatten wir in der letzten Folge untersucht. Dabei kam heraus, daß er — mit nur einem Argument versehen — testet, ob in einer spezifizierten Speicherstelle ein bestimmtes Bit gesetzt ist. Im Gegensatz dazu prüft WAIT mit zwei Argumenten, ob ein bestimmtes Bit gelöscht ist.

16. Nochmal WAIT:
Diesmal mit zwei Argumenten.

Wie funktioniert das? Gehen wir aus vom Befehl

WAIT C,A,B

Dabei ist C die abzufragende Speicherstelle, A eine AND-Maske und B eine EOR-Maske. Sogar im Programmers Reference Guide findet man eine falsche Funktionsbeschreibung. Tatsächlich geschieht folgendes:

1) Der Wert der Speicherstelle C wird gelesen.
2) Dieser Wert wird mit B als EOR-Maske exklusiv-oder-verknüpft
3) Das Ergebnis davon wird mit A als AND-Maske verknüpft.

Anhand eines Beispiels wollen wir uns ansehen, was passiert. Der Befehl

WAIT 1,32,32

hält ein Programm an, bis eine Datasettentaste gedrückt wird (H. Hauck, 64'er, Ausgabe 11/1984, Seite 135). Bit 4 der Speicherstelle 1 ist in dem Fall nämlich Null. Der normale Inhalt dieses Registers ist 55 (binär 0011 0111), also Bit 4 = 1. Die Zahl 32 lautet binär

0001 0000

Mit WAIT1,32,32 geschieht folgendes:

keine Taste:

```
 0011 0111    55
 0001 0000    32
_________   EOR
 0010 0111
 0001 0000    32
_________   AND
 0000 0000     0
```

Das Ergebnis ist Null, der Computer wartet weiter.

Taste gedrückt:

```
 0010 0111    39
 0001 0000    32
_________   EOR
 0011 0111
 0001 0000    32
_________   AND
 0001 0000    32
```

Das Ergebnis ist ungleich Null, der Computer geht im Programm weiter.

Im Programmers Reference Guide steht es genau umgekehrt: Danach soll erst die AND und dann die EOR-Operation stattfinden. Lassen Sie uns diese Variante ebenfalls einmal durchspielen:

keine Taste:

```
 0011 0111    55
 0010 0000    32
_________   AND
 0010 0000
 0010 0000    32
_________   EOR
 0000 0000     0
```

Ergebnis also wieder Null. Der Computer wartet weiter.

Taste gedrückt:

```
 0010 0111    39
 0010 0000    32
_________   AND
 0010 0000
 0010 0000    32
_________   EOR
 0000 0000    0!
```

Auch wenn eine Taste gedrückt ist, würde bei dieser Funktionweise der Computer weiter warten bis zum jüngsten Gericht.

Wir sollten uns das merken: Es kommt bei mehreren logischen Verknüpfungen auch auf die Reihenfolge an.

Daß der WAIT-Befehl so selten benutzt wird, hängt zum einen sicherlich mit seiner relativ komplexen Handhabung zusammen, zum anderen aber auch damit, daß meist ein äußeres Ereignis eintreten muß, um den zu überprüfenden Bit-Wert zu verändern. Dafür kommen Kontrollregister in Frage, die für Ein-/Ausgabe-Operationen jeglicher Art eine Rolle spielen, einige VIC-II-Chip-Register und auch Speicherstellen des SID-Chip. Die beiden letzteren sind allerdings von der Firmware her ohnehin auf allerlei Ereignisse hin programmiert, so daß eine Behandlung mittels des WAIT-Befehls lediglich manchmal als Alternative gesehen werden kann. Bleiben also die CIA-Register, de-

ren komplexe Funktion aber meist eher zur Bearbeitung in Assembler zwingt als in Basic.

Ein Anwendungsbeispiel soll aber noch vorgestellt werden. Es stammt von H. Kohlen und wurde im 64'er, Ausgabe 10/1984, Seite 92 abgedruckt. Es handelt sich um eine Joystickabfrage (Port 2), die allerdings immer nur eine Joystickbewegung erfaßt:

WAIT56320,16,16 wartet auf Feuerknopf
WAIT56320,4,4 wartet auf Linksbewegung
WAIT56320,1,1 wartet auf Hochbewegung
WAIT56320,2,2 wartet auf Abwärtsbewegung
WAIT56320,8,8 wartet auf Rechtsbewegung

Das gleiche mit der Speicherstelle 56321 bezieht sich dann auf den Joystickport 1. Wenn also ein Programmabschnitt warten soll, bis der Feuerknopf gedrückt wird, ist WAIT56320,16,16 sicher eine Möglichkeit, so etwas zu programmieren.

17. Logische Sätze

Im Grunde haben wir bisher lediglich das logische Handwerkszeug kennengelernt. Zwar erbrachte das auch einige direkte Anwendungen, aber richtig logische Probleme zu lösen, dazu bedarf es noch einiger Überlegungen. Genau das wollen wir versuchen: Kann ein Computer verzwickte logische Aufgabenstellungen befriedigend lösen?

Die meisten logischen Fragestellungen bestehen nicht nur aus der Anwendung einer einzigen Verknüpfungsart, sondern — wie schon der WAIT-Befehl mit zwei Argumenten — aus mehreren verkoppelten Aussagen. Wie stellt man die Wahrheitswerte solcher Kombinationen her?

Zunächst ein Beispiel für solch einen logischen SATZ (so nennt man diese verkoppelten Aussagen auch häufig):

NOT(A AND (NOTB))

Zum Ergebnis gelangt man wieder über eine Wahrheitstabelle. Abhängig von der Anzahl der Variablen (hier also zwei: A,B) muß eine Anzahl von Zeilen vorgesehen werden, die alle möglichen W/F-Kombinationen enthält. Bei zwei Variablen sind das dann vier Zeilen, allgemein aber bei N Variablen 2^N Zeilen. Die Anzahl der Spalten ergibt sich aus der Menge der Variablen und der vorzunehmenden Verknüpfungen. In Bild 1 sehen Sie die Wahrheitstabelle für unser Beispiel.

Links finden Sie die Kombinationen der Wahrheitswerte der Variablen A und B. Daneben wurde NOTB entwickelt, dann (wie bei algebraischen Rechnungen Schritt für Schritt) die AND-Verknüpfung und schließlich die Verneinung der Spalte davor. Diese letzte Spalte ist auch eine Zusammenstellung der Ergebnisse.

A	B	NOT B	A AND (NOT B)	NOT (A AND (NOT B))
W	W	F	F	W
W	F	W	W	F
F	W	F	F	W
F	F	W	F	W

Bild 1. So sieht die Wahrheitstabelle des logischen Satzes NOT(A AND (NOTB)) aus

A	B	A IMP B
W	W	W
W	F	F
F	W	W
F	F	W

Bild 2. Das ist die Wahrheitstabelle der IMP-Verknüpfung

Das kann auch unser C 64 für uns rechnen. Das Programm »Logelei-1« erledigt das in der Weise, daß es erst nach der logischen Funktion fragt, deren Wahrheitstabelle berechnet werden soll, sich dann selbst verändert (Zeilen 100-130) und schließlich die Berechnungen durchführt.

Zu beachten ist, daß »Logelei-1« nur die normalerweise in unserem Basic-Sprachvorrat enthaltenen logischen Operatoren kennt.

Zwei Sonderfälle von logischen Sätzen sollen Sie kennenlernen. Zunächst die sogenannte »Tautologie« (hochdeutsch: Gleichbedeutung). Davon spricht man, wenn sämtliche Wahrheitswerte des Ergebnisses W (wahr) sind (beziehungsweise 1). Das ist der Fall in Hamlets berühmten Satz (wenn er damit ein inklusiv-Oder gemeint hat) »sein oder nicht sein«. Probieren Sie mal aus:

A OR (NOT A)

(Das Programm Logelei-1 ist übrigens darauf nicht eingerichtet). Das Gegenteil der Tautologie ist die »Kontradiktion« (zu hochdeutsch: Widerspruch). Sämtliche Wahrheitswerte des Ergebnisses sind F (falsch oder 0). Das findet man zum Beispiel bei:

A AND (NOT A)

Als Beispiel sei A die Aussage »es regnet«. Dann ist es eine Kontradiktion zu behaupten: »Es regnet und es regnet nicht«.

Bisher fehlen uns aber noch einige entscheidende Werkzeuge: Wir können noch keine Schlußfolgerungen ziehen. Das steuern wir nun an.

Es gibt in einigen Basic-Dialekten den Operator EQV. Das kommt von »äquivalent«, also »gleich«. Wir dürfen in unserem C 64 getrost dafür das Gleichheitszeichen (=) verwenden. Äquivalenz zweier logischer Sätze liegt vor, wenn die Wahrheitswerte der Ergebnisse gleich sind. Probieren Sie mal mittels unseres Programmes:

a) NOT (A AND B)
b) (NOT A) OR (NOT B)

Beide Sätze sind äquivalent, weil die Ergebnisse gleiche Wahrheitswerte besitzen. Nebenbei: Das ist eines der sogenannten Gesetze von DeMorgan. Das Programm Logelei-1 zeigt sich — wie Sie nun sehen werden — auch schon als »Beweismaschine«. Bei der Frage nach der Funktion geben Sie doch mal dieses DeMorgan-Gesetz ein:

(NOT(AANDB))=
((NOTA)OR(NOTB))

Das Ergebnis zeigt lauter -1, also W. Für alle Variablenkombinationen ist dieser Satz also richtig. Vielleicht ahnen Sie jetzt schon, was ein Computer in Fragen Logik zu leisten imstande ist. Probieren Sie doch mal selbst einfach ein paar Äquivalenzen aus. So können Sie selbst noch zum Entdecker werden.

18. Bedingungen

Auf dem Weg zum Computer als »Schlußfolgerungs-Maschine« fehlt uns noch etwas Wichtiges: Bedingungen. Das sind Verknüpfungen der Form »Wenn... dann...«. Man nennt diese Verknüpfungen »Implikationen« und findet in manchen Basic-Dialekten dafür den Operator IMP. Wir dürfen statt dessen > = verwenden.

(Der Vollständigkeit halber sei noch angemerkt, daß wir im C 64 anstelle von EOR »< >« verwenden können. Der C 128 kennt diesen Befehl als XOR.)

Im weiteren soll für diese Wenn-/Dann-Verknüpfung immer IMP verwendet werden. Sie können dann in einem Programm leicht > = dafür einsetzen. Für eine Aussagen-Operation der Form A IMP B gibt es wieder genaue Vorschriften, wie die einzelnen Wahrheitswerte miteinander verrechnet werden. Bild 2 zeigt Ihnen die Wahrheitstabelle der IMP-Operation:

Nur dann also, wenn A richtig und B falsch ist, ist auch A IMP B falsch. Diese Wahrheitstabelle erstellt Ihnen auch ohne weiteres Logelei-1. Wenn Sie schon am Probieren sind, dann geben Sie doch noch mal diese Äquivalenz ein:

(A > = B) = ((NOTA)ORB)

Das Ergebnis zeigt: Anstelle von A IMP B kann auch (NOTA) ORB verwendet werden. Solche Äquivalenzen sind sehr brauchbar. Deshalb werden wir uns noch eine weitere IMP-Operation ansehen. Überprüfen Sie doch mal mittels Logelei-1 die Wahrheitswerte dieses Satzes:

(NOTB) IMP (NOTA)

Sie sehen, daß genau dieselben Wahrheitswerte herauskommen wie für A IMP B. Auch diese Sätze sind demnach äquivalent. Man nennt den letzteren »Kontraposition« von A IMP B. Als Beispiel für eine Anwendung dieser Erkenntnis wollen wir einen Beweis führen:

Der Satz »Wenn x^2 ungerade ist, dann ist x ungerade« soll bewiesen werden. Wenn wir für A einsetzen »x^2 ungerade« und für B »x ungerade«, dann ist also zu beweisen A IMP B. Die Kontraposition davon wäre dann der Satz: »Wenn x gerade ist, dann ist auch x^2 gerade«. Das zu zeigen, ist relativ einfach. Eine gerade Zahl x ist auszudrücken durch 2n: x = 2n. Dann ist für x^2 zu schreiben:

$x^2 = (2n)\,(2n) = 2\,(2n^2)$,

was wieder eine gerade Zahl ist. Damit haben wir die Kontraposition bewiesen. Weil aber die Kontraposition äquivalent ist zu unserem Ausgangssatz, ist so auch dieser bewiesen.

Was damit zu zeigen war, ist die Tatsache, daß man zu Beweisen häufig nur einen geeigneten äquivalenten Satz beweisen muß, was unter Umständen sehr viel leichter sein kann.

19. Argumente, Schlußfolgerungen

Wenn jemand in einer Diskussionsrunde einen anderen Teilnehmer abbügelt, indem er behauptet, dieser habe keine gültigen Argumente, dann kommt er dem, was ein Logiker unter diesem Begriff versteht, schon ziemlich nahe. Für diesen ist ein Argument eine Beziehung zwischen logischen Sätzen, die er »Prämissen« (Voraussetzungen) nennt und einem Satz, der »logischer Schluß« heißt. Solche Argumente können dann »gültig« sein oder »irrig« (also ungültig).

Gültigkeit liegt nur dann vor, wenn alle Prämissen den Wahrheitswert W aufweisen und außerdem gleichzeitig auch der logische Schluß den Wahrheitswert W hat. Das ist vielleicht so

trocken etwas unverständlich. Wir sehen uns daher ein Beispiel an.

Unser Beispiel beinhaltet ein grundlegendes Gesetz, das Gesetz der logischen Schlußfolgerung (Gesetz der Syllogistik, aufgestellt von Aristoteles). Man kann es so ausdrücken: »Aus A folgt B. Aus B folgt C. Dann folgt auch aus A die Aussage C.« Die beiden ersten Sätze sind die Prämissen. Der letzte Satz ist der logische Schluß. In Formeln:
Prämissen: A IMP B
B IMP C
Schluß: A IMP C

Das Ganze ist ein Argument, und wir müssen nun nachprüfen, ob es gültig ist. Dazu bauen wir uns eine Wahrheitstabelle, die nun (weil drei Variable vorhanden sind) $2^3 = 8$ Zeilen enthalten muß. Außerdem soll jedem Satz und jeder Variablen eine Spalte zugeordnet sein. In Bild 3 ist diese Wahrheitstabelle zu sehen.

A	B	C	A IMP B	B IMP C	A IMP C
W	W	W	W	W	W
W	W	F	W	F	F
W	F	W	F	W	W
W	F	F	F	W	F
F	W	W	W	W	W
F	W	F	W	F	W
F	F	W	W	W	W
F	F	F	W	W	W

Bild 3. Wahrheitstabelle eines gültigen Argumentes. Hier das syllogistische Gesetz des Aristoteles.

Links finden Sie alle möglichen Kombinationen der Variablen-Wahrheitswerte, rechts daneben dann die Werte, die bei den drei Sätzen herauskommen. In den markierten Zeilen sind die Werte beider Prämissen W und — wie Sie unschwer feststellen werden — auch der Wahrheitswert des Schlusses ist dort W. Damit ist dieses Argument gültig.

Eine andere, etwas computergerechtere Form, lernen wir nun noch kennen. Dabei verknüpft man alle Prämissen durch die AND-Operation und dies so entstandene Gebilde wird durch die IMP-Operation mit dem Schluß verkoppelt. Das Ergebnis ist eine Tautologie, wenn das Argument gültig ist. Unser Beispiel lautet in dieser Form dann:
((A IMP B)AND(B IMP C))
IMP (A IMP C)

Unser Computer wird also eine »Schlußfolgerungs-Maschine«, wenn wir das Programm Logelei-1 etwas erweitern und umbauen. Als »Logelei-2« sehen Sie diese Schlußfolgerungsmaschine hier abgedruckt.

Im Grunde genommen tut dieses Programm nichts anderes als Logelei-1, nur sind jetzt unterschiedliche Variablen-Anzahlen zugelassen (von einer bis zu vier Variablen). Ebenso wie vorhin wird hier auch nach Anzeigen der vorhandenen Funktion gefragt, ob eine andere gewünscht sei. Danach muß die Anzahl der Variablen eingegeben werden und unter Umständen die neue Funktion. Wenn eine Schlußfolgerung zu untersuchen ist, wird sie in der eben erklärten Form (Prämissen AND-verknüpft etc.) eingegeben. Ansonsten verfahren Sie wie bei Logelei-1. Der Computer erstellt nun eine Wahrheitstabelle. Liegt eine Tautologie vor, dann sind alle Ergebnis-Wahrheitswerte gleich -1 und damit das Argument gültig, die Schlußfolgerung also richtig. Um das Programm überschaubar zu halten, sind einige Eleganzen nicht verwendet worden. Aber vielleicht reizt Sie das ja zum Erweitern, Verfeinern, etc.

Versuchen Sie es mal mit dem obigen Beispiel. Sie sehen, wir können mit Logelei-2 ein logisches Gesetz beweisen. Natürlich kann man diese Schlußfolgerungsmaschine auch im alltäglichen Bereich einsetzen. Das soll das folgende einfache Beispiel zeigen:

Ein Student erzählt seine Situation: »Wenn ich jobbe, dann kann ich nicht gleichzeitig studieren. Wenn ich studiere, dann schaffe ich das Examen. Wenn ich nicht jobbe, dann muß ich am Hungertuch nagen. Ich habe mich entschlossen, trotzdem zu studieren. Dann habe ich mein Examen und nage am Hungertuch.« Das sind 4 Sätze mit 4 Variablen:
A = ich jobbe
B = ich studiere
C = ich schaffe das Examen
D = ich nage am Hungertuch

In der verkürzten Form lautet die Äußerung des Studenten dann:
Prämissen: A IMP NOTB
B IMP C
NOTA IMP D
B
Schluß: C AND D

Nach dem Starten der Schlußfolgerungs-Maschine geben Sie also ein:
((A IMP (NOTB)) AND (B IMP C) AND ((NOTA) IMP C) AND B) IMP (C AND D)

Anstelle von IMP verwenden Sie die oben genannten Zeichen. Die Zwischenräume können wegbleiben, denn sie sind nur der Deutlichkeit halber eingefügt. Außerdem sagen Sie bitte dem Computer, daß er es mit vier Variablen zu tun hat. Das Ergebnis zeigt, daß die Schlußfolgerung richtig ist: Lauter -1 treten auf.

So, nun können Sie dieses Programm auf eigene Probleme anwenden. Ein paar nette Beispiele finden Sie in einem Artikel von D. Herrmann in der Zeitschrift computer colleg, Ausgabe 2/1985, Seite 26. Wenn Sie öfters versuchen, Schlüsse auf diese Weise per Computer zu überprüfen, werden Sie feststellen, daß der schwierigste Schritt die Umwandlung von Sätzen in die mathematische Kurzform darstellt. Das kennen Sie vielleicht schon von Textaufgaben in der Mathematik her.

20. Künstliche Intelligenz?

Wenn Sie diese Folge bis hierher gelesen und vielleicht einige Schlüsse selbst ausprobiert haben, ist bei Ihnen möglicherweise der Eindruck entstanden, daß Sie hier so etwas wie den Keim der Künstlichen Intelligenz (auch mit dem Kürzel KI bedacht) vorliegen haben. Ein Programm, das in der Lage ist, mehr Variable — vielleicht sogar Hunderte — zu verarbeiten, welches die Schlüsse aus allen möglichen Variablenkombinationen selbst erstellt und mittels der Tautologien überprüft auf Richtigkeit, solch ein Programm wäre im Prinzip auf der gleichen Basis wie »Logelei-2« zu erstellen, also relativ einfach. Zwar hätte es ziemlich große Ausmaße und es bedürfte sicherlich eines wesentlich schnelleren Prozessors, um in absehbaren Zeiten zu Resultaten zu kommen, aber es wäre technisch realisierbar. Ist das Künstliche Intelligenz, und — diese etwas utopisch anmutende Zusatzfrage stellt sich zwangsläufig — wäre das Programm nicht in der Lage, Firmen, Gesellschaften, ja auch dem Staat eine wichtige Hilfe bei Entscheidungen zu sein?

Versuchen wir zunächst, die Frage nach KI zu beantworten, dann erkennen wir sehr schnell, daß eine Antwort nicht möglich ist. Noch gibt es keine allgemein anerkannte Definition, was Intelligenz eigentlich sei. Dann kann auch kein Begriff einen sinnvollen Inhalt haben, der daraus abgeleitet ist.

Die zweite Frage ist nicht ganz so leicht zu klären. Solch eine »Super-Schlußfolgerungs-Maschine« wäre sicherlich in einigen Anwendungsbereichen sehr nützlich. Aber vielleicht ist Ihnen schon bei der Formulierung von Problemstellungen für unser Programm »Logelei-2« aufgefallen, daß es da manchmal Schwierigkeiten gibt. Irgendwie fällt es ab und zu schwer, einen Satz, in dem Adjektive oder Adverbien eine große Rolle spielen, in die Kurzform umzusetzen. Nun haben Sie recht, wenn Sie sagen, daß das sicher eine Frage der Feinheit der Methode sei. Wie haben hier ja auch nur einen kleinen Zipfel der Logik zu fassen bekommen. Aber trotzdem bleiben Probleme. Im täglichen Leben werden Sie höchst selten Situationen vor sich haben, die mit JA oder NEIN (Wahr oder Falsch) zu bewältigen sind. Da gibt es zum Beispiel noch ein JA,ABER oder ein VIELLEICHT oder ein JEIN etc. Die zweiwertige Logik, auf der all das fußt, was wir in diesen drei Folgen behandelt haben, spielt in der Realität nur eine untergeordnete Rolle! Sie ist deshalb auch nur höchst selten und jedesmal mit einer gehörigen Portion Skepsis auf reale Problemstellungen anwendbar.

Allen, die neugierig auf dieses Thema geworden sind, möchte ich zwei Bücher ans Herz legen, die ebenso unterschiedlich wie nützlich sind:

1) Seymour Lipschutz: »Essential Computer Mathematics«. McGraw-Hill 1982. ISBN 0-07-037990-4. Das ist einer der Bände der Serie »Schaum's Outline Series«, der auch noch andere mathematische Themen anpackt und das alles mit vielen Beispielen würzt.

2) Douglas R. Hofstadter: »Gödel, Escher, Bach ein Endloses Geflochtenes Band«. Klett-Cotta 1985. ISBN 3-608-93037-X. Das ist für jeden, der sich für Computer, Logik, KI und das Denken an sich interessiert, eine Art Bibel.

(Heimo Ponnath/gk)

```
Y=((A)=B)AND(B)=C)AND(C)=D)))=(A)=D)

-------------------------------
 A   B   C   D        ERGIBT
-------------------------------

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

Bild 4. Das Programm »Logelei-2« überprüft Aussagen. Wenn alle Ergebnisse —1 lauten, ist die Aussage richtig (so wie hier).

```
1 REM ********************************    <132>
2 REM *                              *    <051>
3 REM *      LOGELEIEN FUER ZWEI     *    <224>
4 REM *            VARIABLE          *    <026>
5 REM *                              *    <054>
6 REM *  HEIMO PONNATH  HAMBURG 1985 *    <131>
7 REM *                              *    <056>
8 REM ********************************    <139>
9 REM                                     <071>
10 PRINT CHR$(147):POKE 53280,0:POKE 53281
   ,0:POKE 646,5:GOTO 30                  <086>
15 REM **** UP CURSOR SETZEN ****         <067>
20 POKE 211,S:POKE 214,Z:SYS 58640:RETURN <220>
25 REM **** HP-1: FUNKTIONENEINGABE ***   <148>
30 S=11:Z=5:GOSUB 20:POKE 646,7:PRINT"LOGE
   LEIEN FUER ZWEI"                       <076>
35 S=16:Z=6:GOSUB 20:POKE 646,2:PRINT"VARI
   ABLE"                                  <227>
40 S=6:Z=10:GOSUB 20:POKE 646,5:PRINT"IM P
   ROGRAMM IST DIE FUNKTION:"             <136>
45 K=1:GOSUB 100:K=0:S=0:Z=15:GOSUB 20:PRI
   NT"C="L$                               <028>
50 S=6:Z=20:GOSUB 20                      <188>
55 PRINT CHR$(18)"A"CHR$(146)"LTE ODER "CH
   R$(18)"N"CHR$(146)"EUE FUNKTION ?"     <227>
60 GET A$:IF A$<>"A"AND A$<>"N"THEN 60    <218>
65 IF A$="A"THEN 95                       <055>
70 S=0:Z=22:GOSUB 20:PRINT"C(A,B)=";:INPUT
    L$:POKE 781,22:SYS 59903              <176>
75 GOSUB 20:PRINT"C="L$:FOR I=0 TO 500:NEX
   T:POKE 646,0                           <002>
80 PRINT CHR$(147)CHR$(17):PRINT"100L$="CH
   R$(34)L$CHR$(34)                       <182>
85 PRINT"110DEFFNL(A)="L$:PRINT"120DEFFNM(
   B)="L$:PRINT"RUN95"CHR$(19);           <023>
90 POKE 631,13:POKE 632,13:POKE 633,13:POK
   E 634,13:POKE 198,4:END                <195>
94 REM **** HP-2: BERECHNUNGEN ****       <056>
95 POKE 646,5:PRINT CHR$(147)             <227>
100 L$="NOT(A AND(NOT B))"                <206>
110 DEF FN L(A)=NOT(A AND(NOT B))         <020>
120 DEF FN M(B)=NOT(A AND(NOT B))         <110>
130 IF K=1 THEN RETURN                    <139>
140 S=5:Z=5:GOSUB 20:POKE 646,2:PRINT"HIER
     IST DIE WAHRHEITSTABELLE:"           <230>
145 PRINT:PRINT:PRINT:POKE 646,1:PRINT TAB
    (5)"A"TAB(10)"B"TAB(25)"ERGIBT C"     <085>
150 PRINT TAB(4)"-------------------------
    -------":POKE 646,5:PRINT             <215>
155 FOR A=-1 TO 0:FOR B=-1 TO 0:PRINT TAB(
    5)A TAB(10)B TAB(27)FN M(B)           <078>
160 NEXT B:NEXT A                         <255>
165 PRINT:POKE 646,1:PRINT TAB(4)"--------
    ------------------------":POKE 646,5  <227>
170 S=0:Z=20:GOSUB 20:PRINT"C="L$         <035>
```

© 64'er

Listing Logelei-1. Eine Hilfe zum Erstellen von Wahrheitstabellen bei zwei Variablen

```
1 REM ********************************    <132>
2 REM *                              *    <051>
3 REM * DIE SCHLUSSFOLGERUNGSMASCHINE *   <217>
4 REM *       BIS ZU 4 VARIABLE      *    <062>
5 REM *                              *    <054>
6 REM *  HEIMO PONNATH  HAMBURG 1985 *    <131>
7 REM *                              *    <056>
8 REM ********************************    <139>
9 REM                                     <071>
10 PRINT CHR$(147):POKE 53280,0:POKE 53281
   ,0:POKE 646,5:GOTO 30                  <086>
15 REM **** UP CURSOR SETZEN ****         <067>
20 POKE 211,S:POKE 214,Z:SYS 58640:RETURN <220>
25 REM **** HP-1: FUNKTIONENEINGABE ***   <148>
30 S=7:Z=5:GOSUB 20:POKE 646,7:PRINT"SCHLU
   SFOLGERUNGSMASCHINE                    <005>
35 S=10:Z=6:GOSUB 20:POKE 646,2:PRINT"BIS
   ZU 4 VARIABLE"                         <119>
40 S=6:Z=9:GOSUB 20:POKE 646,5:PRINT"IM PR
   OGRAMM IST DIE FUNKTION:"              <126>
45 K=1:GOSUB 100:K=0:S=0:Z=12:GOSUB 20:PRI
   NT"Y="L$                               <225>
46 Z=14:GOSUB 20:PRINT"MIT "N" VARIABLEN"  <113>
50 S=6:Z=16:GOSUB 20                      <199>
55 PRINT CHR$(18)"A"CHR$(146)"LTE ODER "CH
   R$(18)"N"CHR$(146)"EUE FUNKTION ?"     <227>
60 GET A$:IF A$<>"A"AND A$<>"N"THEN 60    <218>
62 Z=17:GOSUB 20:PRINT"WIEVIELE VARIABLE (
   MAX 4)";:INPUT B$                      <092>
63 N=VAL(B$):IF N<1 OR N>4 THEN 62        <140>
65 IF A$="A"THEN 95                       <055>
70 S=0:Z=19:GOSUB 20:PRINT"Y(A,B,..)=";:IN
   PUT L$:POKE 781,19:SYS 59903           <181>
75 GOSUB 20:PRINT"Y="L$:FOR I=0 TO 500:NEX
   T:POKE 646,0                           <024>
80 PRINT CHR$(147)CHR$(17):PRINT"100L$="CH
   R$(34)L$CHR$(34)                       <182>
85 PRINT"110DEFFNL("CHR$(64+N)")="L$:PRINT
   "120N="N:PRINT"RUN95"CHR$(19);         <254>
90 POKE 631,13:POKE 632,13:POKE 633,13:POK
   E 634,13:POKE 198,4:END                <195>
94 REM **** HP-2: BERECHNUNGEN ****       <056>
95 POKE 646,5:PRINT CHR$(147)             <227>
100 L$="((A>=B)AND(B>=C)AND(C>=D))>=(A>=D)
    "                                     <080>
110 DEF FN L(D)=((A>=B)AND(B>=C)AND(C>=D))
    >=(A>=D)                              <018>
120 N= 4                                  <137>
125 C$="{4SPACE}-------------------------
    ------"                               <146>
130 IF K=1 THEN RETURN                    <139>
140 S=2:Z=0:GOSUB 20:POKE 646,2:PRINT"HIER
     IST DIE WAHRHEITSTABELLE VON:"       <247>
145 S=0:Z=2:GOSUB 20:PRINT"Y="L$:POKE 646,
    1:PRINT:PRINT C$:POKE 646,5           <025>
150 ON N GOSUB 200,300,400,500:PRINT:POKE
    646,1:PRINT C$:POKE 646,5:PRINT       <207>
155 PRINT"WENN ALLE ERGEBNISSE -1 LAUTEN,
    DANN"                                 <193>
160 PRINT"IST DER SCHLUSS RICHTIG ! (TASTE
    )":POKE 198,0:WAIT 198,1              <202>
190 END                                   <192>
195 REM ***** 1 VARIABLE *****            <055>
200 PRINT TAB(15)"A"TAB(30)"ERGIBT":POKE 6
    46,1:PRINT C$:PRINT:POKE 646,5        <090>
205 FOR A=-1 TO 0:PRINT TAB(15)A TAB(30)FN
     L(A)                                 <173>
210 NEXT A:RETURN                         <090>
295 REM ***** 2 VARIABLE *****            <158>
300 PRINT TAB(5)"A"TAB(10)"B"TAB(30)"ERGIB
    T":POKE 646,1:PRINT C$:PRINT:POKE 646,
    5                                     <008>
305 FOR A=-1 TO 0:FOR B=-1 TO 0           <171>
310 PRINT TAB(5)A TAB(10)B TAB(30)FN L(B)  <092>
315 NEXT B:NEXT A:RETURN                  <071>
395 REM ***** 3 VARIABLE *****            <003>
400 PRINT TAB(5)"A"TAB(10)"B"TAB(15)"C"TAB
    (30)"ERGIBT"                          <172>
405 POKE 646,1:PRINT C$:PRINT:POKE 646,5  <070>
410 FOR A=-1 TO 0:FOR B=-1 TO 0:FOR C=-1 T
    O 0                                   <022>
415 PRINT TAB(5)A TAB(10)B TAB(15)C TAB(30
    )FN L(C)                              <030>
420 NEXT C:NEXT B:NEXT A:RETURN           <202>
495 REM ***** 4 VARIABLE *****            <104>
500 PRINT TAB(5)"A"TAB(10)"B"TAB(15)"C"TAB
    (20)"D"TAB(30)"ERGIBT"                <223>
505 POKE 646,1:PRINT C$:PRINT:POKE 646,5  <170>
510 FOR A=-1 TO 0:FOR B=-1 TO 0:FOR C=-1 T
    O 0:FOR D=-1 TO 0                     <132>
515 PRINT TAB(5)A TAB(10)B TAB(15)C TAB(20
    )D TAB(30)FN L(D)                     <231>
520 NEXT D:NEXT C:NEXT B:NEXT A:RETURN    <024>
```

© 64'er

Listing Logelei-2. Die Schlußfolgerungs-Maschine. Ein Programm zum Überprüfen der Stimmigkeit logischer Schlüsse.