Seminararbeit

99 Bottles of Beer (LOGO)

Florian Bergsleitner
csaf7492@uibk.ac.at

2. Februar 2011

Betreuer: DI Friedrich Neurauter

Zusammenfassung

Im Zuge des Vertiefungsseminars "99 Bottles of Beer" wird die Programmiersprache LOGO vorgestellt. Auf den einleitenden Teil, über den Werdegang der Sprache, folgen der grundlegende Befehlssatz und die Syntax. Anwendungsbeispiele sollen diese verdeutlichen. Durch weiterführenden Programmcode werden Vergleiche zu (modernen) Programmiersprachen gezogen, Turing-vollständigkeit gezeigt sowie ein Ausblick auf Einsatzmöglichkeiten geboten.

Inhaltsverzeichnis

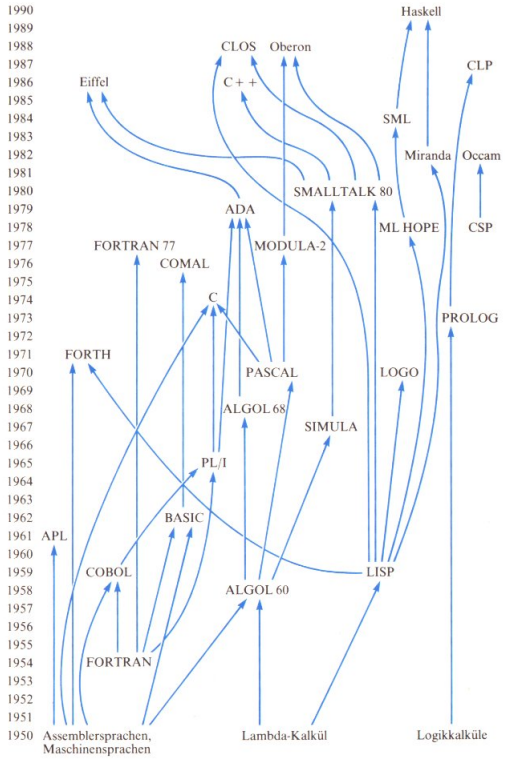
1 Geschichte der Programmiersprache LOGO

1.1 Entstehung/Idee hinter der Sprache

LOGO wurde 1968 von Seymour Papert[1] (Professor für Mathematik und Erziehungswissenschaften) erfunden. Papert beschäftigte sich gerne mit dem Thema „Kinder und Computer“ beziehungsweise dem Zugang zu neuen Technologien bereits im Schulalter. LOGO sollte eine Schnittstelle darstellen. Er war zu der Zeit am Massachusetts Institute of Technology (MIT) tätig. Dort forschte er um Kindern die Möglichkeit zu geben mit Computern zu arbeiten – speziell: damit zu schreiben bzw. Grafiken zu erstellen.

1.2 Wie hat sich die Sprache im Lauf der Zeit entwickelt?

LOGO lässt sich nur schwer einem klassischen Bereich zuordnen (Imperativ, Logisch, Funktional, Objektorientiert). Es ist eine funktionale Programmiersprache, besitzt aber imperative Sprachkonzepte. Laut unten stehender Grafik [8] entstand es aber aus dem Lambda Kalkül (also klassisch „funktional“) und ist eine Weiterentwicklung aus LISP. Die Verwendung von LOGO erfuhr Mitte der 80er–Jahre seinen Höhepunkt (Apple II Computer, TI 99/4A und im schulischen Bereich).



Oft wird LOGO in der Literatur (z.Bsp.[6]) gar als “LISP without parentheses“ bezeichnet. Bis dato gibt es keinen “LOGO-Standard“. 2009 gab es 197 ver-

schiedene Implementierungen bzw. Dialekte von LOGO. Stark beigetragen zur Verwirrung hat die Tatsache, dass viele „Turtle-Varianten“ sich fälschlicherweise „LOGO“ nannten. Die meisten sind nicht mehr in Verwendung – andere wiederum stehen nach wie vor noch in der Entwicklung.

1.3 Die wichtigsten Varianten von LOGO

- UCB-LOGO (Berkeley-LOGO) LOGO, welches einem „de-facto-Standard“ entsprechen würde - das Programm, mit welchem auch hier gearbeitet wurde.
- MSWLOGO und dessen Nachfolger FMSLOGO (Freeware) sind andere Varianten, welche noch im schulischen Bereich verwendet werden. Mit diesen werden gerne einfache GIF-Animationen erstellt. MSWLOGO unterstützt dabei mehrere Turtles und 3D Grafiken. Zudem gewährt es Eingaben über COM, LPT sowie Hardware – Ports.
- MicroWorlds LOGO und Imagine LOGO (kommerzielle Varianten) unterstützen Funktionen im Multimedialen Bereich.
- StarLogo oder NetLogo sind modernere Variationen von Logo und arbeiten mit tausenden unabhängigen Turtles. Sie werden auch gerne für Sozialstudien und Experimente im Bereich der Biologie und der Physik eingesetzt.
- ObjectLogo (objektorientierte Variante)

1.4 Ist Logo Turing-vollständig?

Turing-Vollständigkeit bezeichnet in der Berechenbarkeitstheorie die Eigenschaft einer Programmiersprache oder eines anderen logischen Systems, sämtliche Funktionen berechnen zu können, die eine universelle Turingmaschine berechnen kann[2]. Hierbei spielt weder der Aufwand (um Berechnungen zu erstellen), noch physikalische Größen (Speicherbedarf, Zeit, ...) eine Rolle. Moderne Programmiersprachen (C++, Java, ..) sind Turing-vollständig. Auch funktionale Vertreter (LISP, Haskell). Aufgrund der Tatsache, dass LOGO eine Erweiterung von LISP ist, könnte bereits rückgeschlossen werden, dass die Sprache ebenfalls Turing-Vollständig ist. LOGO verfügt über sämtliche Funktionen, welche (moderne) Programmiersprachen definieren.

2 Befehlssatz und Syntax

2.1 Wie schwierig ist es Logo zu lernen?

Logo ist eine Interpretersprache - die Analyse des Quellcodes erfolgt somit zur Laufzeit des Programms (wie z.Bsp. auch in BASIC). Durchaus bekannt ist die „Turtle-Grafik“: Eine (dreieckige) Schildkröte wird über den Bildschirm bewegt und erzeugt dadurch Linien. Ein eingegebenes Kommando zeigt sofort seine Auswirkung. Ein paar Grundbefehle:

#Bewegen der Turtle
forward (Länge)/fd (Länge)
Beispiel:

forward 100

#Drehen der Turtle nach rechts/links
rt/lt (degree)
Beispiel:

rt 100

#Rückkehr zum Startpunkt
home

#Clean screen and return home
cs

Es können mehrere Befehle (hintereinander) eingegeben werden – zur Veranschaulichung:

```
? fd 100
? rt 90
? fd 100
? rt 90
? fd 100
? rt 90
? home
```

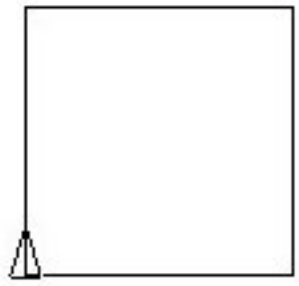


Abbildung 1: Quadrat in LOGO

2.2 Stärken und Schwächen der Sprache (Was kann LOGO?)

- Ausgaben:
  LOGO kann über den Befehl "print (bzw. pr)"die Eingabe(n) ausgeben:

```
? pr [Hello World]
Hello World
```

Listing 1: Ausgaben mit print/pr

- Arithmetik:
  Es verfügt über eine Vielzahl mathematischer Operationen. Auch Präzedenzen werden berücksichtigt:

```
? print sin 90
1
? print sqrt 144
12
? print 2*3+4
14
```

Listing 2: Arithmetische Berechnungen

- Bool'sche Operationen:
  Boolsche Funktionen werden in Präfix-Notation angegeben:

```
? and "true (or "false not "false)
```

Listing 3: Boolsche Operationen

- Variablen deklarieren:
  Frei definierbare Variablen mit: make "variablenname wert

```
? make "zehn 10
? print 3*zehn
30
```

Listing 4: Variablen deklarieren

- Konkatenieren von Strings:

```
? print "LOGO\ kann\ alles
LOGO kann alles
```

Listing 5: String Konkatenation

- Anzeigen aktueller Daten:
  Sämtliche Variablen, welche zum Zeitpunkt angelegt sind, können mit ihrem aktuellen Wert über den 'pons'-Befehl angesehen werden:

```
? pons
Make "counter 0
Make "zehn 10
```

Listing 6: Aktuelle Variablen ausgeben (pons = print out names)

- Schleifen:
  Schleifen werden über "REPEAT Anzahl Befehlsfolge"realisiert. Dabei wird die Wiederholungsanzahl angegeben und eine zu wiederholende Befehlsfolge:

```
? repeat 4 [fd 100 rt 90]
```

Listing 7: Schleifen – Beispiel Quadrat

- Hilfestellung:
  Selbstverständlich stehen auch Hilfsmöglichkeiten zur Verfügung um die Syntax von Befehlen zu betrachten:

```
? help "forward
FORWARD dist
FD dist

moves the turtle forward, in the direction that it's fac-
ing, by the specified distance (measured in turtle steps).
```

Listing 8: Hilfe (man-pages) mit: help “keyword

In der Hilfe sieht man die Vielzahl der Möglichkeiten – vor allem aber wichtig: Mit LOGO können eigene Funktionen deklariert werden.

2.3 Funktionen (Definition, Hintereinanderausführung)

- Funktionsdefinition:

```
to Funktionsname (:eingabeparameter)
Befehlsabfolge
end
```

Listing 9: Syntax von Funktionen

Im untenstehenden Beispiel wird die Funktion ‘quadrat‘ erstellt mit Eingabeparameter ‘laenge‘. Anschliessend kann die Funktion ausgeführt werden z.Bsp.: quadrat 100:

```
? to quadrat :laenge
> fd :laenge
> rt 90
> fd :laenge
> rt 90
> fd :laenge
> rt 90
> fd :laenge
> end
```

Listing 10: Funktion ‘quadrat‘ ohne Schleife

Um die Übersicht zu wahren, wird weiterer Quellcode (sowie dieses Beispiel) im Kapitel 4 ‘Anhang der Codebeispiele‘ aufgeführt.

Die Turtle schaut nach ausführen dieses Programmes “Richtung Westen”. Durch viermaliges Ausführen des Programmes zeigt sich ein neues Bild (siehe

Abbildung 3). Erneutes ausführen ergibt also eine andere Ausgabe (das Quadrat startet in anderer Richtung) – dies wird im Beispiel 4.2 ausgenutzt. Weiters wird auch die bereits erzeugte Funktion „quadrat“ innerhalb der neuen Funktion (nquadrat) aufgerufen.

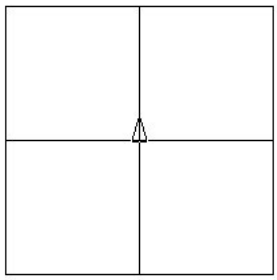


Abbildung 2: Quadrat mit Eingabeparameter und Schleife (Bsp. 4.2)

Es kann recht einfach ein beliebiges n-Eck gezeichnet werden – auch mit mehreren Eingabeparametern, wie das Beispiel 4.3 veranschaulicht: Eingaben sind die Anzahl der Wiederholgungen (n) und die Länge (laenge). Die Turtle dreht sich also jeweils um 360/n Grad:

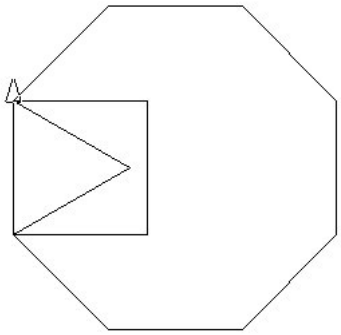


Abbildung 3: Beliebiges n-Eck (Bsp. 4.3)

Basierend auf vorhergehende Funktionen, können eindrucksvolle graphische Elemente erstellt werden; für das untenstehende Beispiel wird das n-Eck aus Bsp. 4.3 m-mal wiederholt:

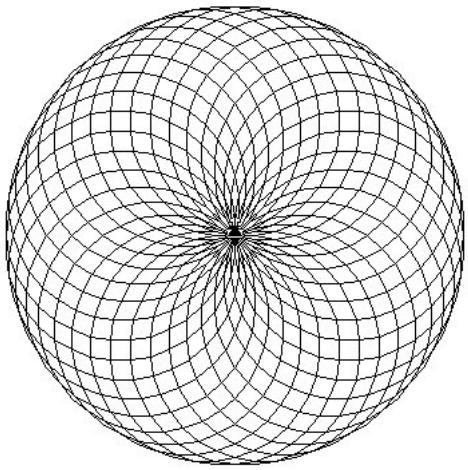


Abbildung 4: mxn-Eck (Bsp. 4.4)

Im folgenden Abschnitt wird ein Überblick an ausgewählten (auch bereits vordefinierten) Funktionen gezeigt:

- vordefinierte Funktionen (z.Bsp.: 'random')

Die vordefinierte Funktion erwartet eine positive (Integer-)Zahl, welche die Obergrenze der Zufallszahl darstellt.

```
pr random 10
```

Listing 11: Zufallszahl in einem angegebenen Bereich

- Selbst definierte: max-Funktion

```
to max :x :y
ifelse (:x > :y) [output :x] [output :y]
end
```

Listing 12: Das Maximum von 2 Zahlen wird ausgegeben

- Hintereinanderausführung von Funktionen (mit Zeitverzögerung):

Zunächst 2 Funktionen (chair und erasechair)

```
to chair
repeat 4 [fd 100 rt 90] fd 200
end
```

Listing 13: Zeichnet Stuhl

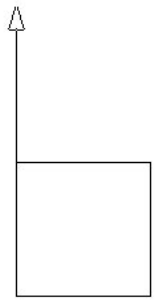


Abbildung 5: Chair

```
? to erasechair
> pe
> bk 200 repeat 4 [fd 100 rt 90]
> ppt
> end
```

Listing 14: löscht Stuhl (pe = pen erase, ppt= pen paint)

```
#Start Programm
? chair wait 200 erasechair
```

Listing 15: erzeugt und löscht Stuhl nach Wartezeit

Dabei wird durch ‘wait 200‘ 200*1/60 Sekunden gwartet, also ca. 3,3 Sekunden.

2.4 Rekursionen

Es wird anhand einiger Beispiele gezeigt wie Rekursionen in LOGO funktionieren. Diese sind sowohl aus dem graphischen, als auch aus dem mathematischen Bereich gewählt worden. Ein abschliessendes Beispiel mit Rekursionen bei Listen verdeutlicht die Mächtigkeit der Sprache auch in diesem Gebiet. Abbruchbedingungen bei rekursiven Prozeduren werden über ’STOP’ oder ’OUTPUT’ formuliert.

- Ein graphisches Rekursionsbeispiel

```
to spiral :size
if :size > 30 [stop] ; #Abbruchbedingung
fd :size rt 15;
spiral :size * 1.02; #rekursiver Aufruf
end
```

Listing 16: Rekursiver Aufruf

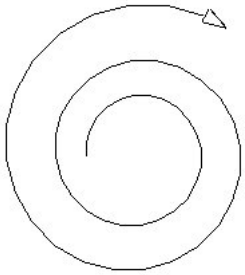


Abbildung 6: spiral 10

- Aufsummieren von n Gliedern

```
? to summe :n
> output ifelse :n=0 [0][:n + summe :n-1]
> end
```

Listing 17: Aufsummieren von 0 bis n

- Fibonacci-Folge

```
to fibonacci :n
ifelse :n < 2 [output 1] [output sum fibonacci :n-1 fibonacci :n-2]
end
```

Listing 18: Fibonacci-Reihe

- Listen

Das folgende Beispiel soll erstellt werden:

```
? updown "LOGO
L
LO
LOG
LOGO
LOGO
LOG
LO
L
```

Listing 19: Rekursion bei Listen - Aufruf z.Bsp: updown "LOGO

Dies benötigt: Eine Funktion 'up', welche ein Wort Buchstabe für Buchstabe (beginnend mit einem) erweitert.
Beispiel:

```
? up "test
t
te
tes
test
```

Listing 20: Ausgabe der Funktion 'up'

Die Funktion 'down', welche mit dem gesamten Wort beginnt und jeweils den hintersten Buchstaben abschneidet:

```
? down "test
test
tes
te
t
```

Listing 21: Ausgabe der Funktion 'down'

Der Befehl 'emptyp' gibt 'true' aus, wenn ein eingegebenes Wort leer ist ('false' sonst).

Und schließlich die Funktion 'updown', in welcher nacheinander beide Funktionen aufgerufen werden:

```
\#Funktion up
to up :word
if emptyp :word [stop]
up butlast :word
print :word
end

\#Funktion down
to down :word
if emptyp :word [stop]
print :word
down butlast :word
end

\#Vereinigung der beiden Funktionen
to updown :word
up :word
down :word
end
```

Listing 22: Rekursion bei Listen - Aufruf z.Bsp: updown "LOGO

2.5 Anwendungsbeispiele und Vergleiche zu anderen Programmiersprachen

Vergleich mit Funktionalen Programmiersprachen: LISP, OCaML

- Vergleich mit LISP

Beispiel zur Berechnung der Fakultät in LOGO

```
? to ! :n
> output ifelse :n=0 [1][:n*! :n-1]
> end
```

Listing 23: Fakultätsberechnung (Bsp.: ! 6)

Beispiel zur Berechnung der Fakultät in LISP

```
(defun ! (n)
(cond ((zerop n) 1)
((>= n 1) (* n (! (1- n))))))
```

Listing 24: Fakultätsberechnung (Bsp.: (! 6))

Die Syntax ist also recht ähnlich – eine weitere (iterative) Möglichkeit für UCB-LOGO:

```
? to fac :n
> if :n = 0 [make "n 1]
> if :n > 1 [make "n :n * fac :n - 1]
> output :n
> end
```

Listing 25: Fakultätsberechnung (Bsp.: print fac 6)

Die Doppelpunkte im obigen Beispiel sind nicht zwingend. Die Variante hat starke Ähnlichkeit mit OCaML.

```
let rec fac n =
if n = 0 then 1
else n * (fac (n - 1))
```

Listing 26: Iterative Fakultätsberechnung in OCaML

Auch im Listen-Bereich (meiner Meinung nach eine Stärke von OCaML) kann LOGO mithalten: Ein Beispiel [7]:

```
? make "fluss "donau
? print first :fluss
d
? print first "fluss
f
? pr last :fluss
u
? pr bf :fluss
onau
? pr bl :fluss
dona
```

Listing 27: Grundbefehle für Listen

First liefert das erste Element eines übergebenen Objektes. Mit :fluss wird hierbei auf den Inhalt zugegriffen - mit "fluss auf die Variable selber. Die Funktionen "BUTFIRST (bf)"bezeihungsweise "BUTLAST (bl)"geben alles bis auf das erste beziehungsweise letzte Element zurück.

```
? make "Stadt [Linz liegt an der Donau]
? pr first :Stadt
Linz
? pr item 2 :Stadt
liegt
? show bf :Stadt
[liegt an der Donau]
? show bl :Stadt
[Linz liegt an der]
```

Listing 28: Ein weiterführendes Beispiel für Listen

Bei den beiden letzten Beispielen show, statt print (da print bei einer Liste nicht die eckigen Klammern anzeigt. BUTFIRST(bf) und BUTLAST(bl) geben nämlich eine Liste zurück, wenn das übergebene Objekt eine Liste war. First und Last geben in diesem Falle ein Wort zurück.

3 Ausblick

3.1 Warum sich die Sprache nicht durchgesetzt hat

LOGO ist schnell „verwendbar“. Mit wenigen Befehlen lassen sich schon beeindruckende Ergebnisse erzielen – speziell im graphischen Bereich. Weiters verfügt es über eine hohe Leistungsfähigkeit (dynamischen Listen aus Lisp), es können frei definierbare (und auch rekursiv aufrufbare) Funktionen erstellt werden und vieles mehr. Der Grund warum sich LOGO nicht durchgesetzt hat ist dennoch einfach: Eine Programmiersprache welche für Kinder entwickelt wurde, kann wohl kaum „erwachsene“ Anforderungen abdecken – LOGO wurde also lediglich unterschätzt. Ein weiterer plausibler Grund wäre auch, der damalige Drang zur strukturierten Programmierung und die damit vorherrschende Meinung, dass Rekursionen nicht nur Speicher- und Rechenzeit beanspruchen, sondern zudem schwer lesbar sind.

3.2 Wo wird/könnte die Sprache eingesetzt werden?

Auf alle Fälle kann es – um der ursprünglichen Idee gerecht zu werden im Unterrichtsbereich eingesetzt werden. Persönlich habe ich in der Mittelschule (im Freifach Informatik) das erste mal mit der Turtle zu tun gehabt. Die Wahl der Sprache war auch deshalb – ich habe bis heute die Sprache LOGO in bleibender Erinnerung. Geometrische/Räumliche Vorstellung kann geschärft werden, die Freude zum arbeiten am Computer wird forciert und die Neugier auf interessante Anwendungsmöglichkeiten einer Programmiersprache wird geweckt. Der Vorteil hierbei (keine koordinatenbasierte Grafikbeschreibung) wurde dem

Programm im professionellen Bereich zum Verhängnis – es wird jedoch weiterhin zur Darstellung von Fraktalen mittels Lindenmayer-Systemen (Ersetzung durch Produktionsregeln – z.BspChomsky-Gramatiken) eingesetzt[3]. Ein wichtiger Bereich, wo LOGO meiner Meinung nach nicht außer Acht gelassen werden sollte, wäre in der Robotik bei der Steuerung von Robotern beispielsweise bzw. allgemeiner bei autonomen Systemen. Und natürlich (als Basis) für andere Programmiersprachen. Ein bekannter Vertreter, welcher unter anderem durch LOGO beeinflusst wurde, ist Smalltalk (welcher wiederum die Java-Entwicklung beeinflusste).

4 Anhang der Codebeispiele

4.1 Quadrat mit Eingabeparameter

```
? to quadrat :laenge
> fd :laenge
> rt 90
> fd :laenge
> rt 90
> fd :laenge
> rt 90
> fd :laenge
> end
```

Listing 29: Beispielaufruf: quadrat 100

4.2 Funktionsaufruf und Schleife

```
to nquadrat :n
repeat :n [
quadrat 100
]
end
```

Listing 30: Beispielaufruf: nquadrat 4

4.3 Beliebiges n-Eck

```
? to n_eck :n :laenge
> repeat :n [
> rt 360 / :n
> fd :laenge
> ]
> end
```

Listing 31: Beliebiges n-Eck

```
? n_eck 3 100 #Dreieck
? n_eck 4 100 #Quadrat
? n_eck 8 100 #Oktogon
```

```
? n_eck 30 15 #nahezu ein Kreis
```

Listing 32: Beispielaufrufe

4.4 mxn-Eck

```
? to mn_eck :m :laenge
> repeat :m [
> rt 360 / :m
> n_eck :m :laenge
> ]
> end
```

Listing 33: Beispielaufruf: mn_eck 36 15

5 „Spielereien“

5.1 Programmcode: 99 Bottles of Beer

[4]

```
? to bottle :i
> if :i = 0 [output [No more bottles of beer]]
> if :i = 1 [output [One bottle of beer]]
> output se :i [bottles of beer]
> end

? to verse :i
> pr (se bottle :i [ on the wall,] bottle :i)
> pr [Take one down, pass it around]
> pr se bottle :i - 1 [ on the wall]
> end

? to sing :i
> if :i = 0 [stop]
> verse :i
> sing :i - 1
> end
#Programmstart:
? sing 99
```

Listing 34: 99 bottles beer on the wall

5.2 LOGO-Wettbewerb 15-word-challenge

[5]

```
repeat 8 [rt 45 repeat 6 [repeat 90 [fd 2 rt 2] rt 90]]
```

Listing 35: z.B.: Dahlie mit 14 Worten

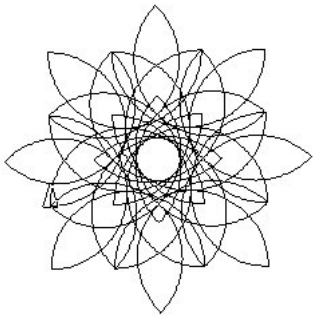


Abbildung 7: Dahlie

Literatur

[1] Quelle: http://www.papert.org/.

[2] Quelle: http://de.wikipedia.org/.

[3] Quelle: http://www.worldlingo.com/.

[4] Quelle: http://99-bottles-of-beer.net/.

[5] http://www.mathcats.com/gallery/15wordcontest.html.

[6] S. Fincher. *Computer Science Education Research*. Crc Pr Inc, 2004.

[7] G. Schättiger, 1999. Quelle: http://www.informatik.uni-hamburg.de/.

[8] E. von Puttkamer. *Wie funktioniert das? - Der Computer*. Meyers Lexikon, 1994. Quelle: http://www.sn.schule.de/~ti/ti85/Seiten/uebersicht.html.