TURBO-PROLOG

Michael Scheidt

PROLOG (PROgramming in LOGic), eine deklarative Programmiersprache, ist 1972 von Alain Colmerauer in Marseille entwickelt und für Anwendungen im Bereich der Verarbeitung und Analyse natürlicher Sprache konzipiert worden. Erst einige Jahre später wurde an der University of Edingburgh der erste effiziente Prolog-Compiler fertiggestellt. Anders als etwa in FORTRAN, PASCAL oder auch BASIC muß der Programmierer in seinen Prozeduren dem Rechner nicht mehr mitteilen, wie er das Problem zu lösen hat, er gibt vielmehr „nur" eine Beschreibung des Problems, das sogenannte „goal" und entsprechende Regeln an. Hiernach wählt Prolog selbst seinen Lösungsweg. Zunächst diente Prolog, welches auf J.A. Robinsons Resolutionsprinzip beruht, als Theorem-Beweiser. Das erste theoretische Modell von Kowalski und Van Emden basierte darauf, daß die vorhandenen Restriktionen äquivalent den Hornklauseln (Klauseln mit maximal einem positiven Literal) sind.[1]

Seitdem japanische Computerfirmen angekündigt hatten, Prolog zur Basisprache ihrer neuen Rechner, die der „fünften Generation", zu machen, stieg das Interesse an dieser Sprache gewaltig. Diese Projekte hatten u.a. sogenannte Prolog-Maschinen zum Ziel, auf denen Prolog als Systemsprache eingesetzt werden soll. Besonders im Zusammenhang mit KI (Künstlicher Intelligenz)[2] wird Prolog immer wieder genannt. So sind zahlreiche Expertensystem[3]-Shells in PROLOG geschrieben, wie APES, ESP/Advisor und Xi.

Prolog ist eine Programmiersprache auf der Basis prädikatenlogischer Ausdrücke und daher vor allem für symbolische Verarbeitung, analog zu LISP, geeignet. Ein Prolog-Programm besteht grundsätzlich aus:
— den Fakten,
— den Regeln
— und den Anfragen,
wobei Fakten und Regeln die Objekte und deren Relation beschreiben.

Die Fakten und Regeln stehen in der Wissensbank *(database)*. Zu deren Darstellung werden die sogenannten Horn-Klauseln *(Clauses)* verwendet. Es können Anfragen an das System formuliert werden, wobei die Wissensbank nach dem erfragten Faktum durchsucht wird *(matching)*. Wird ein Faktum (auch aufgrund abgeleiteter Regeln) gefunden, das die Anfrage erfüllt, so ist die Anfrage erfolgreich, sonst geht sie fehl. Ebenso können Anfragen auch direkt an die Wissensbank gerichtet werden.

Fast jeden Tag kommen neue Prolog-Implementierungen auf den Markt. Fast alle orientieren sich an dem Standardwerk von Clocksin und Mellish: Programming in Prolog. Anders Turbo-Prolog.

Turbo Prolog von Borland International Inc. läuft auf allen IBM PCs und zu diesen kompatiblen Rechnern, die über einen Arbeitsspeicher von mindestens 384 kByte verfügen, unter den Betriebssystemen PC-DOS und MS-DOS (Version 2.0 oder später). Als Einsatzgebiete gibt Borland an:

a) Entwicklung von Expertensystemen
b) Erstellung dynamischer relationaler Datenbanken
c) Bau natürlicher Sprach-Schnittstellen, auch zu bereits existierenden Programmen
d) KI-Programme, Theorembeweiser
e) Findung symbolischer Manipulationsketten zur Lösung von Gleichungen, Differentialen, Integralen uvm.
f) Untersuchung von Betriebsabläufen[4]

Turbo-Prolog weicht, wie schon vorausgeschickt, erheblich vom Clocksin und Mellish „Standard" ab. So fehlen einige wichtige Operatoren wie:

arg: Gibt ein bestimmtes Argument einer vorgegebenen Struktur aus:
arg (2,inhaber (auto,haus),Wert) **ergibt** Wert = haus

functor: Gibt den Prädikatsnamen und die Zahl der Objekte aus:
functor(leser(buch,zeitschrift),Name,Objektanzahl) **ergibt** Name = leer Objektanzahl = 2

univ: verwandelt eine Klausel in eine Liste und umgekehrt:

univ(inhaber (auto,haus))	**ergibt** [inhaber,auto,haus]
univ([vater,franz,helga])	**ergibt** vater(franz,helga)

So wird verständlich, daß viele Turbo-Prolog eher als intelligentes Pascal bezeichnen, denn als Prolog-System. Daß Turbo-Prolog dennoch recht guten Anklang

[1] Vgl. Kowalski, R., Van Emden, M.: The Semantics of Predicate Logic as Programming Language. JACM 23,783 (1986).

[2] Der Begriff der Artificial Intelligence (AI), der 1956 von Mc Carthy kreiert worden ist, hat im Englischen eine wesentlich weitere Bedeutung als im Deutschen. Zum Begriff KI vergleiche auch Dreyfus, Hubert L.: Die Grenzen der Künstlichen Intelligenz. Was Computer nicht können. Königstein/Ts. 1985 und Dreyfus, Hubert L./Dreyfus, Stuart E., Künstliche Intelligenz, Reinbeck 1987 (dazu Drücker, IuR 1987, S. 165ff., 205ff.). Vgl. auch noch Schneider, Künstliche Intelligenz und Wissenschaftspraxis, IuR 1987, S. 274ff., 317ff., 361ff., 419ff.

[3] Nach E. Feigenbaum ein intelligentes Computerprogramm, das mittels Wissen und Schlußfolgerungen Probleme löst.

[4] Turbo-Prolog ist hierzu dank der guten Zugriffsmöglichkeiten auf die I/0 (Eingabe/Ausgabe) Ports des Rechners prädestiniert.

bei vielen Anwendern findet, liegt zum einen an der ausgezeichneten Programmierumgebung, zum anderen an den zahlreichen Funktionen, die Turbo-Prolog dem Programmierer bietet. Vor allem aber liegt es darin begründet, daß Turbo-Prolog einen Compiler darstellt, wogegen die meisten anderen Prolog-Implementierungen für die MSDOS-Welt nur einen Interpreter aufweisen können.[5] Die Charakteristik des Herstellers Borland International ist daher kritisch zu relativieren:

„Turbo Prolog ist die erste Prolog-Implementierung für den IBM PC und Kompatible, die sich sowohl durch Leistungsstärke als auch durch Bescheidenheit in Bezug auf Speicheranforderung auszeichnet. Dabei verfügt das System über mehr Features als viele Großrechner-Implementierungen. Turbo Prolog ist ein ausgereifter Compiler mit Pulldown-Menüs sowie umfassenden arithmetischen und graphischen Möglichkeiten."[6]

Die erste Berührung mit Turbo-Prolog verläuft mehr als erfreulich. Beeindruckt von der Arbeitsumgebung, den Pulldown-Menüs, der Schnelligkeit, in der man das System installiert, ja sein erstes Programm adhoc laden und starten kann, „jubelten" die ersten Kritiker.

Sicherlich, das Compiler-System bietet viele Möglichkeiten, die man sich bei anderen Prolog-Varianten wünschte. So die Graphikbefehle, die Bildschirmfenster, das maschinennahe Programmieren, die Option, Programme, die in anderen Programmiersprachen geschrieben wurden, mit einzubinden. Zudem beeindruckt die außerordentlich hohe Ausführungsgeschwindigkeit der Turbo-Prolog Programme. Spätestens hier jedoch sollte der Benutzer mit einer gewissen Skepsis an das System herantreten. Gilt es doch als eine allgemein bekannte Tatsache, daß Prolog-Programme sich prinzipiell nicht besonders gut compilieren lassen.

Bei näherem Hinsehen zeigt sich, wie Borland International dieses Problem anging. Kurzerhand entschloß man sich, den Clocksin und Mellish-Standard zu verlassen. So muß der Systembenutzer alle in seinem Programm auftretenden Prädikate zuvor deklarieren. Das führt dazu, daß man während des Programmablaufs kaum neue Prädikate erzeugen kann. Analog verhält es sich mit Regeln. Auch diese müssen zuvor festliegen. Hier geht wohl eine der wesentlichsten Stärken von Prolog verloren. Gerade im Bereich der Künstlichen Intelligenz empfiehlt es sich, mit selbstmodifizierenden Programmen zu arbeiten. Auch die sogenannten „compound objects"[7] sind kein Ersatz hierfür.

Zur Struktur von Turbo Prolog-Programmen

Äußerungen, daß die deklarativen Programmiersprachen (LISP, PROLOG) wesentlich einfacher als die traditionellen seien, sind zu relativieren. Sicherlich erreicht der Programmieranfänger in Prolog eher Fortschritte, als in einer herkömmlichen Programmiersprache.

„Die Hoffnung, daß mit derartigen Sprachen geringere Anforderungen an den Benutzer gestellt würden, ist jedoch falsch. Die Anforderungen bleiben, weil mit Entlastung des Programmierers die Möglichkeiten steigen, neuartige Projekte zu realisieren, die angesichts der Komplexität bisher kaum in Angriff genommen werden konnten. Zugleich erfordern selbst einfache Probleme die Kenntnis des Lösungsverfahrens, mit dem Prolog arbeitet. (...) Diese neuen Methoden erfordern eine Anpassung der Denkweise des Programmierers, der sich daran gewöhnen muß, daß die Rekursion nunmehr das einzige Lösungsverfahren ist."[8]

Im folgenden soll der Aufbau von Turbo-Prolog Programmen und deren „Lösung" anhand simpler Beispiele erläutert werden. Sinn dieses Abschnitts kann und soll es jedoch nicht sein, einen Abriß eines Turbo-Prolog Kurses zu bieten. Zudem ist an dieser Stelle nochmals ausdrücklich auf die große Abweichung vom Clocksin-Mellish „Standard" zu verweisen. Ob Turbo-Prolog andererseits selbst einen Standard auf PC Ebene setzen wird, wie dies Turbo-Pascal erreichte[9], ist zu bezweifeln.

```
/*Beispiel 1*/

domains
   mensch, produkt = symbol
predicates
   kauft (mensch,produkt)
clauses
   kauft (regina, kleid).
   kauft (werner, hut).
   kauft (franz, auto).
   kauft (willi,X) if kauft (werner,X).
```

Betrachtet man die erste Zeile, so sieht man das Wort „Beispiel" eingeklammert in den Zeichenfolgen „/*" bzw. „*/". Die Zeichenfolgen markieren einen Kommentar. (analog zu den Klammern: „{", „}" in Pascal.)

In der nächsten Zeile folgt die Bezeichnung „domains". Hier werden die Bereiche, eben die Domains,

[5] Da ein Turbo-Prolog über einen eigenen integrierten Linker verfügt, erhält man, vorausgesetzt man hat in den Pulldown-Menüs den Punkt „EXEfile (auto link)" angewählt, nach dem Compilieren ein ausführbares Programm. Genauer gesagt, es wird eine OBJ Datei erzeugt und hiernach automatisch gelinkt. Hierzu sollte man beachten, daß die Prolog-Dateien PROLOG.LIB und INIT.OBJ sich im Turbo-Prolog-Directory befinden.

Lehnt man ein auto link ab, so können die OBJ-Dateien sowohl mit dem PC-DOS Linker Version 2.14 (oder später) als auch mit dem Overlay Linker PLINK86 Version 1.48 von Phoenix-Software Associates gebunden werden.

[6] Handbuch zu Turbo Prolog Seite 6.

[7] Compound objects sind Prädikate ohne Regeln, die, wenn sie als database deklariert sind, mittels ASSERT und RETRACT dynamisch erzeugt werden können.

[8] Armin Schwarz (Hrsg.): Prolog. Einstieg in die künstliche Intelligenz mit Turbo-Prolog. Würzburg 1987.

[9] Hierbei sollte man jedoch bedenken, daß Turbo-Pascal bei Erscheinen eine nahezu echte Obermenge zum damaligen Pascal Standard darstellte.

zu denen die Objekte in Beziehung gesetzt werden, festgelegt.

Standard-Domaintypen sind:

char: In Hochkommata eingeschlossene einzelne Zeichen wie 'a' oder '&'

string: Eine Zeichenkette, die von Anführungszeichen begrenzt wird, z.B. „Turbo-Prolog"

symbol: Dieser Typ ist im Grunde äquivalent zu dem Domaintyp string, eine Zeichenfolge aus Buchstaben, Zahlen oder „—" resp. Sonderzeichen.

integer: Eine ganze Zahl zwischen −32768 und +32767.

real: Eine Kommazahl im Bereich von ±1e−307 und ±1e+308. Ein weiterer Vorteil von Turbo-Prolog ist, daß integer, wenn nötig, automatisch in real verwandelt werden.

In unserem Fall sind sowohl „mensch" als auch „produkt" vom Typ „symbol".

Die unter predicates genannte Zeile

kauft (mensch,produkt)

stellt eine Aussage dar, die die zuvor beide als Domain-Typ „symbol" deklarierten Objekte „mensch" und „produkt" in eine Relation (: kauft) setzt.

Unter clauses folgen hiernach die Fakten und Regeln zu der zuvor definierten Relation.

Fakten:

```
kauft (regina, kleid).
kauft (werner, hut).
kauft (franz, auto).
```

Regel:

```
kauft (willi,X) if kauft (werner,X).
```

Nachdem unser Beispielprogramm compiliert worden ist, kann Turbo-Prolog ein „goal", ein Ziel vorgegeben werden. Wenn ihm als goal *kauft (regina,kleid)* vorgegeben wird, so erhalten sie die Antwort „true". Fragt man jedoch *kauft(franz,hut),* so wird Turbo-Prolog mit „false" antworten, da diese Aussage keinen Eintrag unter den clauses hat.

Betrachtet man nun die Regel

```
kauft (willi,X) if kauft (werner,X).,
```

so fällt auf, daß anders als bisher, hier Großbuchstaben auftreten. Variablennamen werden in Prolog stets mit einem großen Buchstaben eingeleitet. Befragen wir also unser kleines Programm nach dem „goal" *kauft (willi,hut),* so wird es mit „true" antworten, da, wenn „werner" einen „hut" kauft, nach der oben genannten Regel auch „willi" einen solchen kauft.

Ebenso kann man Variablen in ein „goal" mit einbeziehen. So führt die Frage *kauft (X, Y)* zu den Antworten:

```
kauft (regina, kleid).
kauft (werner, hut).
kauft (franz, auto).
kauft (willi, hut).
```

In dem folgenden Beispiel soll das Lösungsverfahren, dessen sich Prolog bedient, ein wenig näher vorgeführt werden.

Man könnte Prolog beispielsweise dazu nutzen, um die Paarungen für einen Judo Wettbewerb aufzustellen. Hierbei soll beachtet werden, daß jeder Kampf eine Rückrunde bedingt. Zudem sollen nur Kämpfer gleicher Kategorie, sprich Gürtelfarbe, berücksichtigt werden.

```
                    /*Beispiel2*/

domains
  name, guertel = symbol
predicates
  kaempfer (name,guertel)
clauses
  kaempfer (hans,gruen).
  kaempfer (fritz,braun).
  kaempfer (walter,gruen).
  kaempfer (horst,gruen).
```

Um die angestrebte Lösung zu erhalten, befragt man Turbo-Prolog wie folgt:

kaempfer (Sportler1, gruen)
and kaempfer (Sportler2, gruen)
and Sportler1 <> Sportler2

Dies bedeutet übersetzt in natürliche Sprache: Finde einen „kaempfer" der den grünen Gürtel hat und weise dessen Namen der Variablen *Sportler1* zu und finde ebenso einen „kaempfer", der den grünen Gürtel hat, und weise dessen Namen der Variablen *Sportler2* zu. Als dritte Bedingung gilt, daß die so besetzten Variablen *Sportler1* und *Sportler2* nicht gleich sein dürfen.

Wie löst nun Turbo-Prolog solch ein Problem: Zunächst versucht Turbo-Prolog, die erste Teilaufgabe *kaempfer (Sportler1, gruen)* zu lösen. Hiernach geht es zur nächsten Teilaufgabe und so fort.

Auf die erste Teilaufgabe hin wird „hans" der Variablen *Sportler1* zugewiesen. Ebenso wird die Variable *Sportler2* mit „hans" belegt, da Prolog die Wissensbasis wiederum von Beginn an durchsucht. Dies führt zu einem Scheitern der bisher erhaltenen Lösung an der dritten Prämisse „Sportler1 <> Sportler2". Nun setzt ein *Backtracking* ein: Turbo-Prolog geht zurück zur zweiten Teilfrage und bindet nun „walter" an die Variable *Sportler2.* Da diese Kombination auch die dritte Bedingung erfüllt, erhält man als erste Lösung:

Sportler1 = hans, Sportler2 = walter

Insgesamt ergeben sich noch folgende Lösungen:

Sportler1 = hans,	Sportler2 = horst
Sportler1 = walter	Sportler2 = hans
Sportler1 = walter,	Sportler2 = horst
Sportler1 = horst,	Sportler2 = hans
Sportler1 = horst,	Sportler2 = walter

Nach Darstellung der Grundstruktur von Turbo-Prolog soll im folgenden ein Verzeichnis der wichtigsten Turbo-Prolog Standardprädikatsnamen vorgestellt

werden. Hierbei belasse ich es bei einer Kurzerklärung und verzichte aus Platzgründen generell auf eine nähere Ausführung der Prädikate.

Standardprädikate und ihre Funktionen:

asserta: fügt ein Faktum am Anfang einer Datenbasis ein
assertz: fügt ein Faktum ans Ende einer Datenbasis ein
attribute: setzt und liest Bildschirmattribute
back: bewegt die Turtle rückwärts
beep: generiert einen kurzen Ton
bios: ermöglicht den Aufruf vonBIOS-Interrupts
bound: stellt die Bindung einer Variablen fest
char_int: wandelt char in integer um
clearwindow: löscht das aktuelle Bildschirmfenster
closefile: schließt eine Datei
comline: ermöglicht eine Parameterübergabe bei einem Programmaufruf
consult: fügt einen Textfile an die aktive Datenbank an
concat: verkettet Zeichenketten
config: liest eine Konfigurationsdatei ein
consult: fügt der Datenbasis eine Textdatei hinzu
cursor: bestimmt die Cursorposition
cursorform: bestimmt das Aussehen des Cursors
date: setzt und liest das Tagesdatum
deletefile: löscht eine Datei
dir: zeigt ein directory an
disk: bestimmt das aktuelle Laufwerk und den Pfad
display: gibt eine Zeichenkette am Bildschirm im aktuellen Fenster aus
dot: setzt einen Grafikpunkt resp. liest Farbinformationen
edit: ruft den Turbo-Prolog Editor auf.
editmsg: entspricht *edit,* zugleich kann jedoch ein string ausgegeben werden
eof: überprüft, ob das Dateiende erreicht wurde
existfile: ermittelt, ob eine bestimmte Datei existiert
exit: beendet die Ausführung eines Programms
fail: verursacht den Fehlschlag eines Beweisversuches
field_attr: bestimmt Feldattribute
field_str: schreibt und liest eine Zeichenkette auf dem Bildschirm
filemode: ermöglicht den Zugriff auf Binäre Dateien
filepos: setzt und liest die Datensatznummer bei Datenzugriff
file_str: schreibt und liest einen String auf dem Bildschirm
findall: generiert eine Liste aus Backtracking-Daten
flush: alle Datenpuffer werden auf Diskette geschrieben
forward: bewegt die Turtle vorwärts
free: überprüft, ob eine Variable ungebunden ist
frontchar: verkettet strings
frontstr: teilt strings
fronttoken: spezifische Stringoperation unter der Berücksichtigung der Turbo-Prolog-Schlüsselwörter
gotowindow: ermöglicht den schnellen Wechsel zweier Fenster, die sich überlappen
graphics: initialisiert dan Graphik-Bildschirm
inkey: überprüft ob nach dem letzten Lesevorgang eine Taste gedrückt wurde
isname: stellt fest, ob ein string einen gültigen Namen darstellt
keypressed: stellt fest, ob eine Taste gedrückt wurde, ohne das entsprechende Zeichen einzulesen.
left: dreht die Turtle nach links
line: zieht eine Linie
makewindow: generiert ein Bildschirmfenster
membyte: schreibt resp. liest ein Byte aus dem Hauptspeicher
memword: schreibt oder liest ein Byte in bzw. aus dem Speicher
nl: bewirkt einen Zeilevorschub
not: Negation
openappend: öffnet eine Datei, um Daten anzuhängen
openmodify: gibt eine Datei für Änderungen frei
openread: Datei wird zum Lesen geöffnet
openwrite: Datei wird zum Schreiben geöffnet
pencolor: bestimmt die Turtle-Farbe
pendown: Turtle zeichnet
penpos: legt die Position der Turtle fest
penup: Turtle zeichnet nicht
portbyte: schreibt oder liest ein Zeichen eines I/O Ports
prt_dword: erzeugt einen Langwortzeiger auf einer Zeichenkette
readdevice: bestimmt das logische Eingabegerät
readint: liest eine Integer-Zahl
readln: liest einen String
readreal: liest Prolog-Objekte
removewindow: löscht das aktuelle Fenster
renamefile: benennt eine Datei um
retract: löscht Fakten aus der Datenbasis
right: dreht die Turtle nach rechts
save: schreibt bewiesene Sätze in eine Textdatei
scr_attr: bestimmt Bildschirm-Attribute
scr_char: setzt resp. liest Zeichen direkt in den Bildschirm
scroll: rollt den Inhalt des aktuellen Fensters
shiftwindos: wechselt das Bildschirmfenster bzw. vergibt neue Window-Nummern
sound: generiert Töne
storage: dient zu Veränderungen der Speicheraufteilung
str_char: wandelt Strings in Zeichen
str_int: wandelt Strings in Integer-Zahlen
str_len: ermittelt die Stringlänge
str_real: wandelt Strings in Real-Zahlen um
system: ermöglicht das Absenden von DOS-Kommandos
text: setzt den Bildschirm in den Textmodus zurück
time: setzt resp. liest die Uhrzeit
trace: bestimmt den Trace-Status
upper_lower: konvertiert Klein- und Großschrift
window_attr: liest die Bildschirmfenster-Attribute
window_str: schreibt bzw. liest einen String direkt aus dem aktuellen Bildschirmfenster
write: gibt ein Zeichen aus
writedevice: bestimmt das logische Ausgabegerät
writef: ermöglicht eine formatierte Ausgabe von Konstanten und Variablen

Die Programmierumgebung

Gleich nach dem Start des Systems teilt Turbo Prolog dem Benutzer seine Konfiguration mit *(Bild 1).* An dieser Stelle wird für den Benutzer des Systems ersichtlich, wo sich seine Quelldateien, die Objektdateien, die ausführbaren Turbo-Prolog-Programme befinden, und wo das Verzeichnis des Systems selbst angesiedelt ist. Hiernach, das heißt nach dem Druck einer beliebigen Taste, werden dem Benutzer gleich fünf übersichtlich angeordnete Fenster zugleich präsentiert *(Bild 2).* Im obersten sich über die anderen erstreckenden werden die Optionen *Run, Compile, Options, Files, Setup* und *Quit* dargeboten. Sollte man zum Beispiel mit der zuvor angezeigten Konfiguration nicht zufrieden sein, so wählt man den Setup-Punkt an. Nun er-

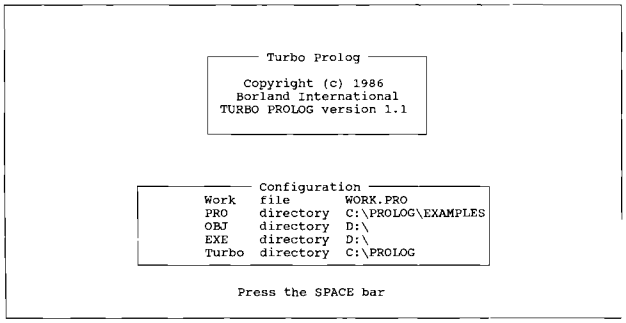

Abb. 1

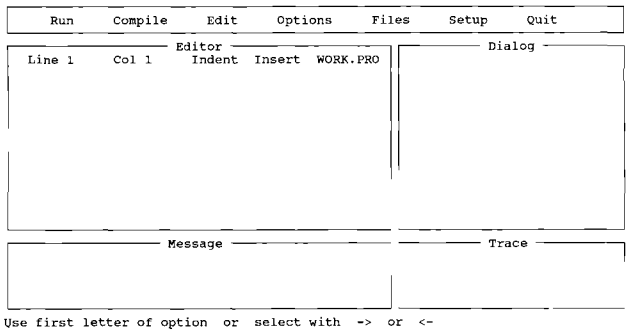

Abb. 2

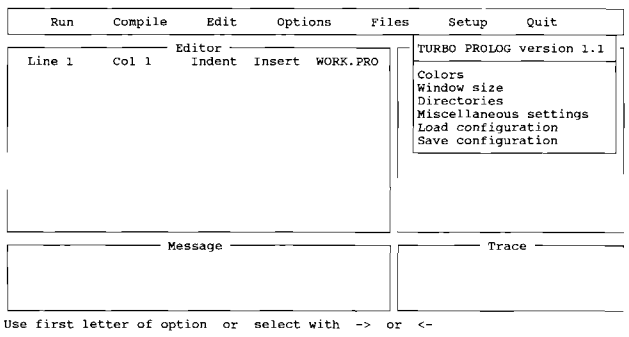

Abb. 3

scheint ein weiteres Fenster, das die anderen teilweise überlappt *(Bild 3)*. Innerhalb dieses Fensters werden wiederum zahlreiche Optionen *(Colors, Window Size, Directories, Miscellaneous settings, Load Konfiguration, Save Konfiguration)* angeboten, unter denen ausgewählt werden kann. Dieses Prinzip gilt für alle anzuwählenden Menüpunkte: Eine unter dem Begriff *Pull-down-Window-Technik* bekannte Benutzerführung, die ohne jede Maus auskommt, schnell erlernt wird und sich durch gute Handhabbarkeit auszeichnet. Zudem steht dem Benutzer hierbei eine situationsbezogene Hilfefunktion zur Verfügung, die nach Aktivierung ebenfalls ein eigenes, die anderen überlappendes Fenster eröffnet.

Zusammenfassung

Eine abschließende Beurteilung Turbo-Prologs wird von zwiespältigen Erfahrungen im Umgang mit diesem System geprägt. Auf der einen Seite steht die außergewöhnlich gute Benutzeroberfläche (ideal für den Anfänger), der schnelle Compiler, der reiche Umfang an Funktionen, auf der anderen Seite jedoch die Feststellung, daß Turbo-Prolog zwar alle diese wunderbaren Attribute zurecht verdient, aber im Grunde kein echtes Prolog mehr ist. Zu gravierend sind die Einschränkungen gegenüber dem Standard von *Clocksin* und *Mellish*. Hier könnte, um als Beispiel das zuvor angeführte Problem der selbstmodifizierenden Programme noch einmal aufzugreifen, die Implementierung eines integrierten Prolog-Interpreters, der eine dynamische Übersetzung ermöglicht, Abhilfe schaffen.

Ein weiterer Kritikpunkt, der hier noch angesprochen werden soll, ist der, daß der Compiler, wohl aus Gründen der Geschwindigkeit, auf einem relativ primitiven Stadium der Analyse verblieben scheint. Dies stellt man vor allem bei der Überprüfung des Determinismus fest, der hier mit Vorliebe einen Nicht-Determinismus signifiziert. Mit kleineren unliebsamen Überraschungen sollte der Benutzer ebenso rechnen, wenn er versucht sein Programm modular zu compilieren.[10]

Dennoch, für den Anfänger ist, abgesehen von der Spracheinschränkung, Turbo-Prolog ein gutes System, um sich mit einer nicht-deterministischen Programmiersprache zu beschäftigen.

Grundlegende Literatur

Dietrich, G.: Kompaktführer Turbo-Prolog. Addison-Wesley-Verlag GmbH, Bonn 1987.
Giannesini, F., Kanoui, H.: Prolog. Addison-Wesley Verlag 1986.
Goldenthal, Nathan: Turbo Prolog. Programmer's Guide. Chesterland/Ohio 1987
Robinson, Phillip, R.: Using Turbo Prolog. Osborne Mac Graw Hill, California 1987
Rogers, Jean B.: A Turbo Prolog Primer. Addison-Wesley Publishing Company Inc. 1987.
Roussel, P.: Prolog Manuel de référenze et d'utilisation. GIA, faculté des sciences de Marseille — Luminy, 1975.
Shafer, Dan: Advanced Turbo Prolog Programming. Howard W. Sans & Company 1987.
Townsend, Carl: Einführung in Turbo Prolog. Düsseldorf 1987.

Weiterführende Literatur zu Prolog und Turbo-Prolog:

Antoniou: Turbo Prolog. Düsseldorf 1987.
Clocksin, W.F. und Mellish, C.S.: Programming in Prolog. New York, 1981.
Colmerauer, A. Kanoui: Prolog, bases théoriques et développements actuels, Technologie et Science Informatiques, Vol. 2, No 4, 1983, faculté des sciences de Marseille — Luminy, 1982.

[10] So wurde ein Programm nach erfolgreicher Compilierung und Ausführung modular gegliedert; mit dem Erfolg, daß dieses Programm nach wie vor ohne Fehlermeldung compiliert wurde, die Ausführung jedoch mit einem Laufzeitfehler, der die freien Variablen betraf, abbrach.