ProVisor

Eine PROLOG-Visualisierung

Programmhandbuch

Version 1.0

TH Darmstadt
6. Mai 1993

Auftraggeber: FG Praktische Informatik
StR G. Röhner

Auftragnehmer: Software-Entwicklungsgruppe
Schneewittchen und die sieben Zwerge

Betreuer: Prof. Dr. W. Henhapl

Schneewittchen und die sieben Zwerge sind: Colette Elcacho
Konrad Berg
Thomas Goetze
Ferenc Kurucz
Michael Metzger
Jürgen Schirmer
Marc Volz
Jan Zurek

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einführung in das Programmhandbuch

Das Programmhandbuch stellt sowohl eine Ergänzung wie auch eine Erweiterung des Benutzerhandbuchs für den interessierten Leser dar:

- Das Benutzerhandbuch ist als Einführung in ProVisor gedacht und kann daher nur einen Überblick über die Funktionalität von ProVisor schaffen. Eine vollständige Beschreibung und damit eine Ergänzung zum Benutzerhandbuch wird Ihnen in diesem Handbuch gegeben.
- Einige Teile, wie das Breakpointsystem oder die graphische Information, erhalten nur eine knappe Beschreibung. Für diese Teile, wie auch für den Sprachumfang und die Syntax des integrierten PROLOG-Interpreters, stellt das Programmhandbuch eine Erweiterung des Benutzerhandbuchs dar.

Die Beschreibung hat einen formalen Charakter. Dennoch wird auf erläuternde Beispiele dort, wo sie angebracht erscheinen, nicht verzichtet.

1.1 Zielgruppe

An erster Stelle ist das Visualisierungsmodell für die Lehrkräfte gedacht, die am HIBS[1] und HILF[2] Lehrerinnen und Lehrer in PROLOG einweisen. Während dieser Fortbildung sollen die unterrichteten Lehrkräfte die Möglichkeit haben, mit diesem Produkt zu arbeiten. Schließlich ist vorgesehen, das Programm den Schülerinnen und Schülern an (vorerst) hessischen Schulen zugänglich zu machen, die an einem PROLOG-Kurs teilnehmen.

Dieses Buch richtet sich an alle interessierte Benutzer von ProVisor, die die Visualisierungsumgebung in ihrer vollständigen Funktionalität kennen lernen und nutzen möchten. Wenn Sie mit der Programmiersprache PROLOG vertraut sind, werden Sie in diesem Buch Möglichkeiten zur Programmablaufsteuerung und Erweiterbarkeit von PROLOG um Systemprädikate finden. Sind Sie dagegen mit dem Programmieren in PROLOG noch nicht so vertraut, empfehlen wir Ihnen zunächst das *Benutzerhandbuch*.

[1] Hessisches Institut für Bildung und Schulwesen
[2] Hessisches Institut für Lehrerfortbildung

1.2 Zum Inhalt des Programmhandbuchs

Das *Programmhandbuch* enthält die detaillierte Beschreibung der Teile von ProVisor, die zur Visualisierung von PROLOG notwendig sind. Eine genaue Beschreibung der Oberfläche und deren Bedienung finden Sie im *Benutzerhandbuch* (→, Kap. 3, S. 15 ff.).

- in Kapitel 2 wird die Funktionalität aller Menüpunkte beschrieben.
- In Kapitel 3 erhalten Sie den genauen und vollständigen Sprachumfang des integrierten PROLOG-Interpreters.

! → Wir weisen Sie nochmals darauf hin, daß es sich bei dem implementierten Sprachumfang um ein Subset von PROLOG handelt. Dieses Subset lehnt sich an den Edingburgh-Standard an, erfüllt ihn aber nicht vollständig.

  Weiterhin wird in diesem Kapitel die genaue Syntax des Subsets beschrieben.

- Kapitel 4 behandelt ausführlich die Visualisierung. Sie erhalten hier alle Informationen, die aus der graphischen Darstellung gewonnen werden können.
- Mit dem Breakpointsystem befaßt sich Kapitel 5. Hier wird die Funkionalität des Breakpointsystems beschrieben und durch Anwendungsmöglichkeiten ergänzt. Zwei kleinere Beispiele runden das Kapitel ab.
- Anhang A enthält ein kleines Glossar.
- Im Literaturverzeichnis finden Sie eine kleine Auswahl von Lehrmaterial zu PROLOG. Diese Auswahl ist natürlich beschränkt und soll keine Bewertung der vorhandenen Literatur zu PROLOG darstellen.

1.3 Die Typographie der Handbücher

Die Typographie entspricht der des Benutzerhandbuchs und wird hier der Vollständigkeit halber aufgenommen:

- `Schreibmaschine` – steht für Eingaben des Benutzers, Programmlistings und Ausgaben, die von Beispielprogrammen erzeugt werden;
- **Fettschrift** – kennzeichnet reservierte Wörter von PROLOG in Listings;
- Alt – für die Tasten der Tastatur, wie zum Beispiel “ein Druck auf die Tasten Alt F3 ” ;

- ⓘ, ⊖ – für die Pfeiltasten zur Bewegung des Cursors und andere Spezialtasten.
- ! → – dient zur Markierung von Warnungen, wichtiger Sätze oder Absätze.
- → X – ist ein Querverweis, wobei *X* für die Nummer auf das verwiesene Kapitel steht.

2. Funktionalität der Benutzeroberfläche

In diesem Kapitel werden alle über die Menüleiste erreichbaren Funktionen aufgelistet und jeweils eine kurze Erläuterung zu deren Wirkung angegeben.

2.1 Das Menü ≡

In diesem Menü sind verschiedene Funktionen zusammengefaßt, die alle die Oberfläche von ProVisor betreffen.

Menüpunkt *Info...*

Info gibt eine Copyright-Meldung und den noch verfügbaren Speicher aus. Dieser Menüpunkt kann jederzeit aufgerufen werden und hat keinerlei Einfluß auf den Programmlauf.

Menüpunkt *Erneuern*

Erneuern bewirkt ein Neuzeichnen aller dargestellten Fenster. Dieser Menüpunkt sollte aufgerufen werden, wenn die Bildschirmdarstellung aus irgendeinem Grund gestört wurde. Ansonsten hat der Befehl keinen Einfluß auf den Programmlauf.

Menüpunkt *Zurücksetzen*

Alle Graphikfenster außer dem Hauptfenster müßen vor dem Aufruf dieses Menüpunktes geschlossen werden. *Zurücksetzen* bewirkt ein Zurückschalten von ProVisor in den Anfangszustand. Alle Editorfenster werden geschlossen und eine eventuell laufende Anfrage wird abgebrochen.

! → Dieser Befehl entspricht in etwa dem Beenden und Neustarten von ProVisor und sollte wirklich nur eingesetzt werden, wenn alle vorherigen Einstellungen unwichtig sind. Die Wissensbasis des Interpreters bleibt jedoch erhalten.

2.2 Das Menü *Datei*

Das *Datei*-Menü enthält alle Funktionen, die mit Dateien zu tun haben.

Menüpunkt *Neu*

Neu öffnet ein leeres Editor-Fenster mit dem Titel `UNBENANNT`. Der Text in diesem Fenster kann jederzeit mit *Sichern* abgespeichert werden. ProVisor fragt dann automatisch nach dem Namen, unter dem der Text abgespeichert werden soll.

! → Dieser Menüpunkt kann jederzeit aktiviert werden. Das neu geöffnet Editor-Fenster erscheint im Vordergrund und ist aktiv.

Menüpunkt *Öffnen...*

Öffnen... öffnet das Dialog-Fenster *Datei*, in dem Sie eine Datei aussuchen. Der Text der ausgewählten Datei wird in einem Editor-Fenster dargestellt und kann verändert werden. Mit *Sichern* wird der Text wieder unter demselben Namen abgespeichert.

! → Dieser Menüpunkt kann jederzeit aktiviert werden. Das Editor-Fenster erscheint im Vordergrund und ist aktiv.

Menüpunkt *Sichern*

Sichern speichert den Text aus dem aktiven Editor-Fenster ab. Hat das Editor-Fenster noch keinen Namen (`UNBENANNT`), dann wird nach einem Dateinamen gefragt, unter dem der Text abgespeichert werden soll.

! → Dieser Menüpunkt kann nur aufgerufen werden, wenn das aktive Fenster ein Editor-Fenster ist.

Menüpunkt *Sichern als...*

Sichern als... speichert den Text aus dem aktiven Editor-Fenster unter einem neuen Namen ab. Wie beim *Sichern* eines unbenannten Editor-Fensters wird nach dem Dateinamen gefragt, unter dem der Text angespeichert werden soll. Existiert bereits eine Datei mit demselben Namen, so wird nach einer Bestätigung gefragt, daß die andere Datei überschrieben werden soll.

! → Dieser Menüpunkt kann nur aufgerufen werden, wenn das aktive Fenster ein Editor-Fenster ist.

Menüpunkt *Consult...*

Consult... fragt nach einem Dateinamen. Die angegebene Datei wird konsultiert, d.h. in die Datenbasis des PROLOG-Interpreters aufgenommen.

! → Dieser Menüpunkt kann nur aufgerufen werden, wenn der Interpreter nicht damit beschäftigt ist, eine Anfrage zu erfüllen.

Menüpunkt *Reconsult...*

Reconsult... fragt nach einem Dateinamen. Die angegebene Datei wird rekonsultiert, d.h. in die Datenbasis des PROLOG-Interpreters übernommen. Im Unterschied zu *Consult...* werden die alten Prädikat-Klauseln von den neuen überschrieben.

! → Dieser Menüpunkt kann nur aufgerufen werden, wenn der Interpreter nicht damit beschäftigt ist, eine Anfrage zu erfüllen.

Menüpunkt *Verzeichnis wechseln...*

Verzeichnis wechseln... ermöglicht es Ihnen, das standardmäßig vorgeschlagene Verzeichnis zu ändern.

! → Dieser Menüpunkt kann jederzeit aufgerufen werden und wirkt sich erst bei der nächsten Datei-Auswahl aus.

Menüpunkt *Beenden*

Beenden verläßt ProVisor. Sollten noch Editor-Fenster mit verändertem Text offen sein, dann werden Sie gefragt, ob diese gespeichert werden sollen.

Dieser Menüpunkt kann jederzeit aufgerufen werden.

2.3 Das Menü *Bearbeiten*

Das Menü *Bearbeiten* enthält Editor- und Clipboard-Funktionen. Alle Befehle (außer Clipboard anzeigen) können nur aufgerufen werden, wenn ein Editor-Fenster aktiv ist.

Menüpunkt *Rückgängig*

Rückgängig erlaubt Ihnen, die zuletzt ausgeführte Editor-Aktion zurückzunehmen, wie z.B. das Löschen einer Zeile oder die letzte Eingabe.

! → Dieser Menüpunkt kann nur aufgerufen werden, wenn im aktiven Editor-Fenster etwas eingegeben oder gelöscht wurde.

Menüpunkt *Ausschneiden*

Ausschneiden kopiert den markierten Text in das Clipboard. Im Editor wird der markierte Text gelöscht.

! → Dieser Menüpunkt kann nur aufgerufen werden, wenn im aktiven Editor-Fenster eine Markierung gesetzt wurde (mit der Maus oder mit Shift und den Cursortasten).

Menüpunkt *Kopieren*

Kopieren kopiert den markierten Text in das Clipboard. Im Editor bleibt der markierte Text erhalten.

! → Dieser Menüpunkt kann nur aufgerufen werden, wenn im aktiven Editor-Fenster eine Markierung gesetzt wurde.

Menüpunkt *Einfügen*

Einfügen kopiert den Inhalt des Clipboards an die Cursorposition im aktiven Editor-Fenster.

! → Dieser Menüpunkt kann nur aufgerufen werden, wenn ein Editor-Fenster aktiv und das Clipboard nicht leer ist.

Menüpunkt *Clipboard anzeigen*

Clipboard anzeigen öffnet ein Editor-Fenster mit dem Clipboard-Text.

! → Dieser Menüpunkt kann nur aufgerufen werden, wenn noch kein Clipboard-Fenster geöffnet wurde.

Menüpunkt *Löschen*

Löschen entfernt den markierten Text aus dem aktiven Editor-Fenster. Der Text wird nicht in das Clipboard übernommen.

! → Dieser Menüpunkt kann nur aufgerufen werden, wenn im aktiven Editor-Fenster eine Markierung gesetzt wurde.

2.4 Das Menü *Suchen*

Das Menü *Suchen* enthält Suchfunktionen, die sich auf das aktive Editor-Fenster beziehen. Diese Suchfunktionen sind nur dann verfügbar, wenn ein Editor-Fenster aktiv ist.

Menüpunkt *Suchen nach...*

Es erscheint ein Dialog-Fenster, in dem nach einem Begriff gefragt wird. Anschließend wird das erste Vorkommen dieses Begriffs im aktiven Editor-Fenster gesucht. Als Option kann die Groß-/Kleinschreibung berücksichtigt werden.

Menüpunkt *Ersetzen...*

Es erscheint ein Dialog-Fenster, in dem nach einem zu ersetzenden alten und dem ersetzenden neuen Begriff gefragt wird. Danach wird das erste Vorkommen des alten Begriffs durch den neuen ersetzt. Als Option kann die Groß-/Kleinschreibung berücksichtigt werden.

Menüpunkt *Weitersuchen*

Weitersuchen wiederholt den letzten *Such*-Befehl, der durch *Suchen nach...* oder *Ersetzen* ausgeführt wurde.

2.5 Das Menü *Visualisierung*

Das Menü *Visualisierung* enthält alle Funktionen, mit denen der Interpreter und die Graphiken gesteuert werden können.

Menüpunkte *Detailbaum*, *Unifikationsbaum*, *Übersichtsbaum*

Es wird ein Baum-Graphikfenster in dem ausgewählten Modus (Detail, Unifikation oder Übersicht) geöffnet. Vor der ersten Anfrage erscheinen alle Baum-Graphikfenster leer.

! → Diese Menüpunkte können nur aufgerufen werden, wenn noch nicht mehr als zwei Baum-Graphikfenster (neben dem Hauptfenster) geöffnet wurden.

Menüpunkt *Darstellung ändern...*

Darstellung ändern... ermöglicht das Ändern des Darstellungsmodus des aktiven Baum-Graphikfensters.

! → Dieser Menüpunkt kann nur aufgerufen werden, wenn ein Graphikfenster mit Baumdarstellung aktiv ist.

Menüpunkt *Breakpoint...*

Breakpoint... öffnet ein Dialog-Fenster, in dem Breakpoints angelegt, editiert und gelöscht werden können. Breakpoints erlauben ein gezieltes Anhalten des Interpreters. Trifft der Interpreter auf einen Breakpoint und werden die angegebenen Bedingungen erfüllt, dann stoppt der Interpreter an dieser Stelle und geht in einen Wartezustand über. Es erscheint eine Anzeige des entsprechenden Breakpoints.

Dieser Menüpunkt ist jederzeit aufrufbar.

Menüpunkt *Term...*

Term... öffnet ein Dialogfenster, in dem ein Term zur statischen Visualisierung eingegeben werden kann. Ist der Term syntaktisch korrekt, dann erscheint eine statische Term-Graphik. Diese bleibt erhalten, bis sie geschlossen wird.

Dieser Menüpunkt ist jederzeit aufrufbar.

Menüpunkt *Einzelschritt*

Einzelschritt führt einen Schritt des Interpreters aus und visualisiert diesen.

! → Dieser Menüpunkt kann nur aufgerufen werden, wenn eine Anfrage gestellt wurde, der Interpreter noch nach einer Lösung sucht und sich in einem Wartezustand befindet.

Menüpunkt *Überspringen*

Überspringen führt alle untergeordneten Prädikate aus. Der Interpreter kommt erst wieder in einen Wartezustand, wenn er sich an derselben Stelle im Baum befindet.

! → Dieser Menüpunkt kann nur aufgerufen werden, wenn eine Anfrage gestellt wurde, der Interpreter noch nach einer Lösung sucht und sich in einem Wartezustand befindet.

Menüpunkt *Weiter*

Weiter fährt mit der Programmausführung fort, bis der Interpreter eine Lösung gefunden hat oder keine Lösung finden konnte.

! → Dieser Menüpunkt kann nur aufgerufen werden, wenn eine Anfrage gestellt wurde, der Interpreter noch nach einer Lösung sucht und sich in einem Wartezustand befindet.

Menüpunkt *Stoppen*

Stoppen hält den Interpreterlauf an. Der Interpreter geht über in einen Wartezustand.

! → Dieser Menüpunkt kann nur aufgerufen werden, wenn eine Anfrage gestellt wurde, der Interpreter noch nach einer Lösung sucht und sich **nicht** in einem Wartezustand befindet (also nachdem zuvor *Überspringen* oder *Weiter* aktiviert wurde).

Menüpunkt *Abbrechen*

Abbrechen bricht den Interpreterlauf ab. Die Anfrage wird verworfen und der Interpreter steht für eine neue Anfrage bereit.

! → Dieser Menüpunkt kann nur aufgerufen werden, wenn eine Anfrage gestellt wurde und der Interpreter noch nach einer Lösung sucht.

2.6 Das Menü *Wissensbasis*

Das Menü *Wissensbasis* enthält alle Befehle zur Steuerung der Wissensbasis des Interpreters.

Menüpunkt *Anzeigen*

Anzeigen veranlaßt den Interpreter, seine Wissensbasis im Interpreterfenster auszugeben.

Dieser Menüpunkt kann jederzeit aufgerufen werden.

Menüpunkt *Löschen*

Löschen löscht die Wissensbasis des Interpreters.

! → Dieser Menüpunkt kann nur aufgerufen werden, wenn der Interpreter nicht damit beschäftigt ist, eine Anfrage zu erfüllen.

2.7 Das Menü *Fenster*

Das Menü *Fenster* beinhaltet alle Funktionen, die sich auf Position und Größe der Fenster beziehen. Diese Funktionen sind jederzeit verfügbar.

Menüpunkt *Interpreterfenster an/aus*

Interpreterfenster an/aus blendet das Interpreterfenster aus bzw. wieder ein. Die Ausgaben in das Interpreterfenster gehen auch dann weiter, wenn das Interpreterfenster ausgeblendet ist. Wird es wieder eingeblendet, dann sind auch diese Ausgaben vorhanden.

Das Hauptfenster paßt seine Größe jeweils an, so daß Hauptfenster und Interpreterfenster zusammen den gesamtem Arbeitsbereich ausfüllen.

Menüpunkt *Größe*

Größe bezieht sich jeweils auf das aktive Fenster. Mit den Cursortasten kann das aktive Fenster verschoben werden, mit (Shift) und den Cursortasten wird die Größe verändert. Zum Abschluß muß (Return) gedrückt werden.

! → Dieser Menüpunkt ist weder auf das Hauptfenster noch auf das Interpreterfenster anwendbar.

Menüpunkt *Ganzer Bildschirm*

Ganzer Bildschirm vergrößert das aktive Fenster auf Hauptfenstergröße und verkleinert bei nochmaligem Aufruf das Fenster wieder auf die vorherige Größe.

! → Dieser Menüpunkt ist weder auf das Hauptfenster noch auf das Interpreterfenster anwendbar.

Menüpunkt *Nächstes*

Nächstes wechselt das aktive Fenster zyklisch.

Menüpunkt *Nebeneinander*

Nebeneinander ordnet alle Fenster (außer Haupt- und Interpreterfenster) nebeneinander an, so daß das gesamte Hauptfenster überdeckt wird. Obwohl das Hauptfenster nun nicht mehr sichtbar ist, kann es dennoch aktiv sein. Erkennen können Sie es allerdings nur daran, daß scheinbar kein Fenster aktiv ist. Verschieben, verkleinern oder schließen Sie die anderen Fenster, dann sehen Sie auch das Hauptfenster wieder im Hintergrund.

Menüpunkt *Überlappend*

Überlappend ordnet alle Fenster (außer Haupt- und Interpreterfenster) hintereinander an.

Menüpunkt *Schließen*

Schließen schließt das aktive Fenster. War das aktive Fenster ein Editor-Fenster und haben Sie den Text verändert, dann werden Sie noch gefragt, ob Sie den Text sichern wollen.

! → Dieser Menüpunkt kann weder auf das Hauptfenster noch auf das Interpreterfenster angewendet werden.

2.8 Das Menü *Optionen*

Das Menü *Optionen* enthält Funktionen, die die Konfiguration von ProVisor beeinflussen.

Menüpunkt *Konfiguration laden...*

Konfiguration laden... erlaubt Ihnen, eine abgespeicherte Konfiguration von ProVisor wiederherzustellen. In einem Dialog-Fenster wählen Sie die wiederherzustellende Konfigurationsdatei aus.

! → Der Menüpunkt kann nur aufgerufen werden, wenn keine Graphik-Fenster geöffnet sind (abgesehen vom Hauptfenster).

Menüpunkt *Konfiguration speichern...*

Konfiguration speichern... erlaubt es Ihnen, die momentane Fensteraufteilung zu speichern. In einem Dialog-Fenster geben Sie den Dateinamen an, unter dem die Konfiguration gespeichert werden soll.

! → Der Menüpunkt kann nur aufgerufen werden, wenn keine Graphik-Fenster geöffnet sind.

Menüpunkt *Protokoll an/aus*

Protokoll an/aus schaltet das Protokollieren des Interpreterfensters in die Datei `TRACE.LOG` aus bzw. bei nochmaligem Aufruf wieder ein.

Dieser Menüpunkt kann jederzeit aufgerufen werden.

Menüpunkt *Trace an/aus*

Trace an/aus schaltet sowohl die Ausgabe im Interpreterfenster als auch in die Protokoll-Datei aus.

Dieser Menüpunkt kann jederzeit aufgerufen werden.

3. Der Interpreter

3.1 Kurzbeschreibung der System-Prädikate von ProVisor

! (Cut)

Der Cut verhindert das Backtracking, d.h. die nach dem Aufruf des Prädikates, in dessen Körper der Cut steht, bis zum Cut selbst getroffenen Festlegungen der Variablen und Prädikatregeln werden beibehalten. Werden folgende Ziele der Regel nicht erreicht, schlägt das Prädikat fehl.

Beispiel:

```
?- 5 < 8, !, vater(peter, hans).
```

?-

Das PROLOG-System zeigt damit an, daß es bereit ist, Anfragen und Anweisungen entgegenzunehmen. Eingaben müssen mit einem Punkt abgeschlossen werden.

X :- Y

Dieser Operator wird beim Festlegen von Regeln verwandt und kann als 'if' gelesen werden. Der Regelkopf X ist der Name des Prädikates (Funktion), gegebenenfalls gefolgt von einer in runde Klammern gesetzte Liste seiner Argumente. Der Regelkörper Y legt dann fest, welche Ziele das Prädikat dann zu erfüllen hat, um selbst erfüllt zu sein.

Beispiel:

```
elternteil(E,Kind) :- vater(E,Kind).
```

X, Y

Dieser Operator steht für das logische 'und'. Es können so mehrere Aufgaben verbunden werden, die dann erfüllt ist, wenn alle Einzelziele erreicht werden.

Beispiel:

```
?- read(X), X < 10, 2 < X, write('Die Zahl liegt
   zwischen 3 und 9').
```

.(X,Y)

Dieser sogenannte Listenoperator macht das Argument X zum Kopfteil einer Liste und das Argument Y zu deren Endteil.

Beispiel:

```
?- X = .(a,b)
no

?- X = .(a,[]).
```

```
X = [a]
```

[X|Y]

Der senkrechte Strich trennt innerhalb der eckigen Klammern den Kopfteil einer Liste von deren Endteil.

Beispiel:

```
?- [X|Y] = [a,b,c].
X = a
Y = [b,c]
```

$X < Y$,
$X =< Y$,
$X > Y$,
$X >= Y$,
$X = \ = Y$

Es wird eine Vergleichsoperation durchgeführt, die prüft, ob das erste Argument kleiner, kleiner-gleich, größer, größer-gleich oder ungleich dem zweiten ist. Die Argumente müssen entweder Integerzahlen oder korrekte, berechenbare arithmetische Ausdrücke sein. Variablen müssen instantiiert sein.

Beispiel:

```
?- 5 < 6.
Yes

?- 5 < X.
No

?- X = 3, X < 100.
Yes

?- 3 + 2 < 5 * (7 + 2).
Yes
```

X = Y

Hier wird die Gleichheit zweier Strukturen geprüft. Beim Vergleich wird darüber hinaus nicht instantiierten Variablen der Wert der strukturell entsprechenden instantiierten Variablen zugewiesen. Sind beide Variablen frei, so wird bei zukünftigen Instantiierungen einer dieser Variablen der anderen automatisch derselbe Wert zugewiesen.

Arithmetische Ausdrücke werden nicht berechnet, sondern strukturell verglichen.

Beispiel:

```
?- f(a,b) = f(X,b).
X = a

?- 5 + X = 5 + Y.
```

```
X = Y
Y = Y

?- 2 + 3 = 3 + 2.
no
```

X == Y

Hier wird auf Identität hin verglichen. Variablen werden nur dann für identisch gehalten, wenn sie mit dem gleichen Wert instantiiert wurden oder schon gleichgesetzt sind.

Beispiel:

```
?- X = Y, X == Y.
X = Y
Y = Y

?- X == 5.
No.
```

[]

Diese beiden Sonderzeichen stehen für die leere Liste, die keine Elemente enthält.

Beispiel:

```
?- X = a, Y = [], Z = .(X,Y).
Z = [a]
```

atom(P)

Aufgabe dieser Funktion ist es, festzustellen, ob P ein gültiges PROLOG-Atom ist.

Beispiel:

```
?- atom('Na was jetzt').
yes

?- atom(atom).
yes

?- atom(3 < 5).
no

?- atom(X).
no

?- atom(4).
no

?- X = a, atom(X).
yes
```

atomic(K)

Bei dieser Funktion wird überprüft, ob K ein einfacher (nicht zusam-

mengesetzter) gültiger PROLOG-Ausdruck ist. Dies gilt für Integerzahlen und Atome.

Die Implementierung dieses Prädikates erfolgt durch Einlesen folgenden Prädikates aus `SYSTEM.PRO`:

```
atomic(X) :- atom(X).
atomic(X) :- integer(X).
```

Beispiel:

```
?- atomic(a).
yes

?- atomic(1).
yes

?- atomic(f(1)).
No

?- X = b, atomic(X).
yes

?- atomic(X).
no
```

consult(*FN*) Konsultiert die angegebene Datei *FN*. Dieses Prädikat ist nur eingeschränkt verwendbar (nur in Anfragen). In ProVisor können Prädikate konsultiert werden, indem der entprechende Menüpunkt (Dateiconsult) gewählt wird, oder die Daten direkt aus einem Editor übernommen werden.

Beispiel:

```
?- consult('append.pro').
```

display(X) Dies ist eine Ausgabefunktion. Ihr Argument wird in der Form der internen Repräsentation ausgegeben. Strukturen erscheinen in der Funktorschreibweise (Listen in der Punktschreibweise).

Beispiel:

```
?- display(3 + 2 + 1).
+(3, +(2, 1))
```

fail ist ein Teilziel, welches immer fehlschlägt. Um wiederholtes Backtracking zu provozieren und um in Verbindung mit dem Cut notfalls das übergeordnete Prädikat als nicht erfüllt zu erweisen, wird 'fail' als Programmsteuerfunktion verwendet.

Beispiel:

```
?- append(X,Y,[1,2,3]), fail.
no
```

integer(X)

Diese Funktion prüft, ob X eine Integerzahl ist.

Beispiel:

```
?- integer(3).
yes

?- X = 1, integer(X).
yes

?- integer(a).
no

?- integer(X).
no
```

X is Y

X wird der Wert des berechneten arithmetischen Ausdrucks Y zugewiesen. Ist X keine freie Variable, erfolgt eine Vergleichsoperation. Y kann sich aus Integerzahlen und folgenden Operatoren und Konstanten zusammensetzen:

+	Addition
−	Subtraktion
*	Multiplikation
/	Division
mod	Modulo-Funktion

Beispiel:

```
?- 4 is 2 + 2.
yes

?- X is 2 + 2.
X = 4

?- 3 * 2 is 2 + 4.
yes

?- X * 2 is 2 + 4.
no

?- 2 is X.
no
```

listing, listing(P)

Es werden alle Klauseln des Prädikates P im Interpreterfenster ausgegeben. Für 'listing' ohne Argument erfolgt die Ausgabe aller Prädikate

in der Datenbasis.

! → Eingefrorene Prädikate werden nicht angezeigt.

Beispiel:

```
?- consult('member.pro').
yes

?- listing(member).
member(E,[E|_]).
member(E,[_|Es]) :- member(E,Es).
```

nl — erzeugt einen Zeilenvorschub im Interpreterfenster

Beispiel:

```
?- write(hallo), nl, write(hallo).
hallo

hallo
```

nonvar(X) — Diese Funktion stellt fest, ob X keine freie Variable ist.

Die Implementierung dieses Prädikates erfolgt durch Einlesen folgenden Prädikates aus `SYSTEM.PRO`:

```
nonvar(X) :- var(X), !, fail.
nonvar(_).
```

Beispiel:

```
?- nonvar(X).
no

?- nonvar(1).
yes
```

not(X) — Dieses Prädiktat bewirkt ein 'fail', wenn das durch X angegebene Ziel erreicht wird, andernfalls ein 'true'. Variableninstantiierungen werden in jedem Fall rückgängig gemacht.

! → 'not' wird als System-Prädikat angesehen, so daß keine Informationen über die Abarbeitung von X angezeigt werden.

Beispiel:

```
?- not(p(X)), write(X).
X = X
```

op(P,A,X) — Mit 'op' kann die Präzedenz P, die Assoziativität und Position A sowie der Name X eines Operators definiert oder abgerufen werden.

P ist eine Integerzahl im Bereich von 0 bis 1200. Je kleiner die Zahl ist, desto größer ist die Präzedenz des Operators.

A kann einer der folgenden Ausdrücke sein:

xf, yf, fx, fy, xfx, xfy, yfx, yfy

'f' steht dabei für den Operator und gibt seine Position an. 'x' besagt, daß Operatoren, die sich in dem Argument auf dieser Seite des Operators befinden, eine niedrigere Präzedenz haben müssen, wohingegen für 'y' auch gleichrangige erlaubt sind.

In ProVisor wird das op-Prädikat nur bei Ausgaben von Termen berücksichtigt.

! → 'op' ist in der Datei SYSTEM.PRO definiert. Änderungen an diesen Definitionen sollten mit Sorgfalt vorgenommen werden.

Beispiel:

```
?- op(P,A,is).
P = 700
A = xfx
```

read(W) Diese Funktion liest den nächsten eingegebenen Ausdruck, der mit einem Punkt und <Return> abgeschlossen werden muß, aus dem Interpreterfenster. W wird dann mit diesem Ausdruck gleichgesetzt.

Beispiel:

```
?- read(X), display(X).
Sum is 3 + 7.
is(Sum, +(3,7))

?- X = f(Y), read(X).
f(a).
X = f(a)
Y = a

?- X = f(Y), read(X).
a.
no

?- read(f(X)).
a.
no
```

reconsult(*FN*) Re-Konsultiert die angegebene Datei *FN*. Dieses Prädikat ist nur eingeschränkt verwendbar (nur in Anfragen). Schon vorhandene Prädikate gleichen Namens und Stelligkeit werden hier im Gegensatz zu consult

überschrieben.

In ProVisor können Dateien rekonsultiert werden, indem der entprechende Menüpunkt (Datei-reconsult) gewählt wird oder die Daten direkt aus einem Editor übernommen werden.

Beispiel:

```
?- reconsult('append.pro').
```

true — Dieses Prädikat ist immer erfüllt, kann aber nicht reerfüllt werden.

repeat — Dieses Prädikat verhält sich wie 'true', es kann jedoch reerfüllt werden. 'repeat' dient zum Aufbau von Schleifen. Die Implementierung dieses Prädikates erfolgt durch Einlesen des folgenden Prädikates aus SYSTEM.PRO:

```
repeat.
repeat :- repeat.
```

var(X) — Aufgabe dieser Funktion ist festzustellen, ob X eine freie Variable ist.

Beispiel:

```
?- var(Hallo).
yes

?- X = a, var(X).
no
```

write(X) — 'write' ist eine Ausgabefunktion. Im Gegensatz zu display werden bei der Ausgabe von Strukturen die Operator-Definitionen (vgl. op) berücksichtigt.

3.2 Sonderprädikate (nur als Anfragen verwendbar)

mem — zeigt den noch frei verfügbaren Speicherplatz des Systems an.

freezeDB — friert alle in der Wissensbasis befindlichen Prädikate ein. Eingefrorene Prädikate werden in einem Listing und einem Trace-Lauf (in ihrer Ausführung) nicht angezeigt und sind nicht überschreibbar (z.B. durch reconsult).

freezePred, freezePred(P) — Ohne Argument entspricht dieses Prädikat freezeDB, die zweite Variante friert nur das angebene Prädikat ein.

meltPred, meltPred(P) — Es werden alle bzw. nur das angegebene Prädikat aufgetaut (d.h. der eingefrorene Zustand beendet). Dieses Prädikat ist auch auf die internen Prädikate (die zu Beginn aus der Datei SYSTEM.PRO eingelesen werden) anwendbar. Manipulationen an internen Prädikaten sollten

nicht vorgenommen werden!

clearDB — Alle in der Wissensbasis befindlichen Prädikate (mit Ausnahme der eingefrorenen) werden gelöscht. Dieser Befehl ist über den Menü-Punkt *Wissensbasis-Löschen* ebenfalls realisiert.

3.3 Syntax von *Zwerg-PROLOG*

Die Beschreibung der Syntax erfolgt in Form von Syntaxdiagrammen. Die Namen innerhalb der (abgerundeten) Kästen und Kreise stehen für Elemente, aus denen sich ein Konstrukt zusammensetzt und sind durch Pfeile miteinander verbunden.

Um ein gültiges Konstrukt zu erhalten, muß man links im Diagramm beginnen, den Pfeilen folgen (die teilweise mehrere Möglichkeiten und/oder Wiederholungen zulassen) und die einzelnen Elemente „addieren“ — solange, bis die rechte Seite des Diagramms erreicht ist. Dabei wird ein Element eines *nicht* abgerundeten Kastens durch ein weiteres Syntaxdiagramm oder textuell beschrieben. Kreise und abgerundete Kästen symbolisieren Terminale. Das sind Elemente, die nicht mehr aufgelöst werden müssen. Deren Schrift wird wegen der besseren Lesbarkeit kursiv dargestellt.

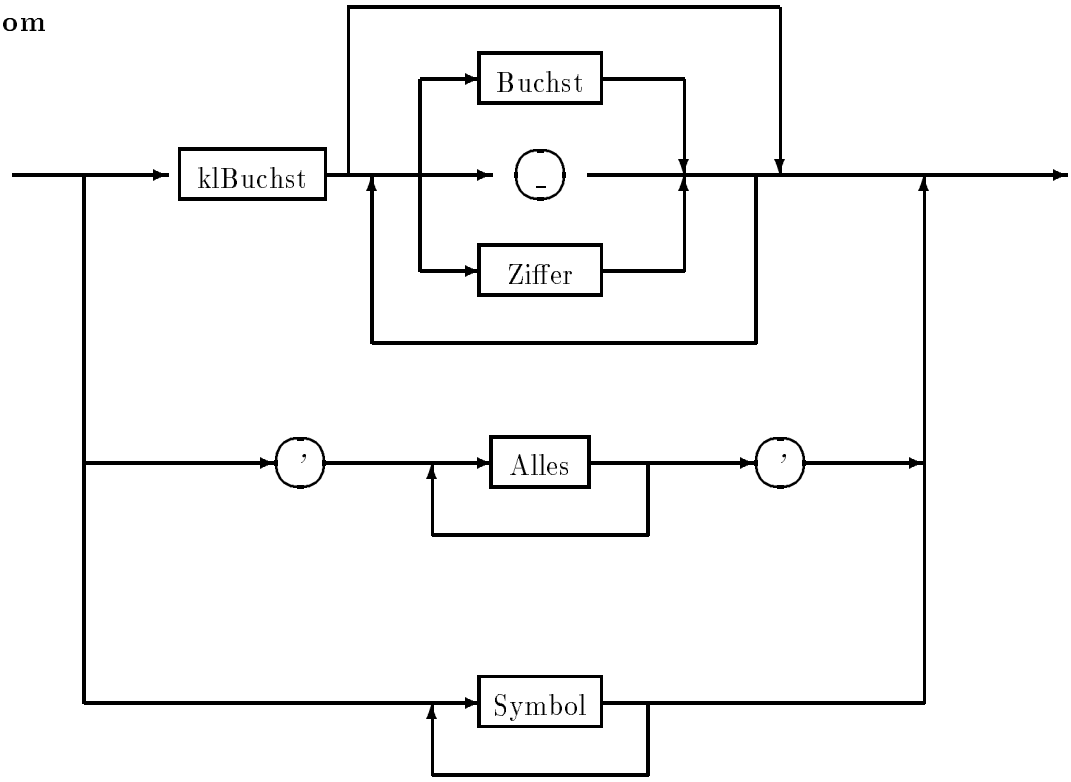
3.3.1 Atome, Zahlen, Variablen, Kommentare

In den Syntaxdiagrammen dieses Abschnitts steht

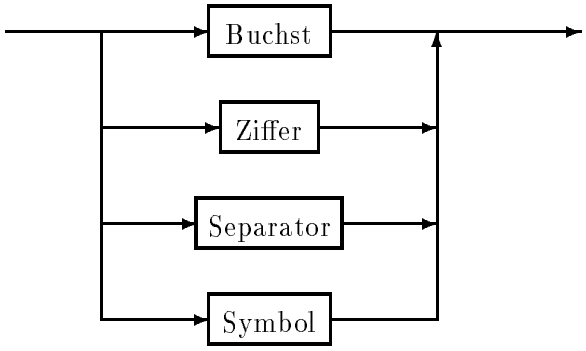
grBuchst	für einen großen Buchstaben „A“ bis „Z“ oder für einen großen deutschen Umlaut („Ä“, „Ö“, „Ü“).
klBuchst	für einen kleinen Buchstaben „a“ bis „z“ oder für einen kleinen deutschen Umlaut („ä“, „ö“, „ü“). Das „ß“ ist nicht erlaubt.
Buchstabe	für grBuchst oder klBuchst.
Ziffer	für eine Dezimalziffer von 0 bis 9.
Separator	für eines der folgenden Zeichen: `.  (  )  [  ]  |  ,  Leerzeichen`

Symbol	für eines der folgenden Zeichen:
	`!  "  %  &  /  =  ?  ‘  ’  {  }  *  +`
	`@  #  \  $  >  ;  :  _  <  -  ^  ~`

Atom



Alles

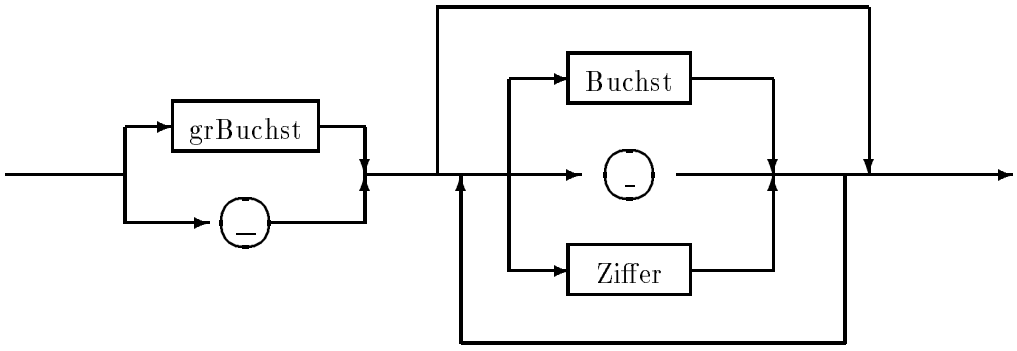


! → Dabei gibt es noch folgende Einschränkungen:

- Ein Atom, das mit einem Hochkomma beginnt, darf außer dem abschließenden Hochkomma kein weiteres enthalten und muß innerhalb einer Zeile beendet werden.

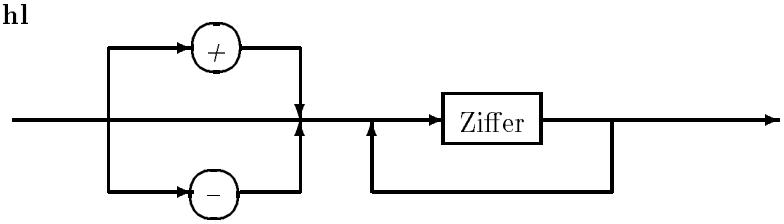
- Ein Atom darf nicht mit „%“, „_“, „+“ oder „−“ beginnen, weil diese Zeichen ein Kommentar, eine Variable bzw. eine Zahl einleiten.
- Ein Atom darf nicht gleich „?−“ oder gleich „:−“ sein, weil dies als Anfragezeichen bzw. als Regelzeichen interpretiert wird.

Variable



! → Die anonyme Variable wird durch ein einfaches Unterstreichungszeichen dargestellt.

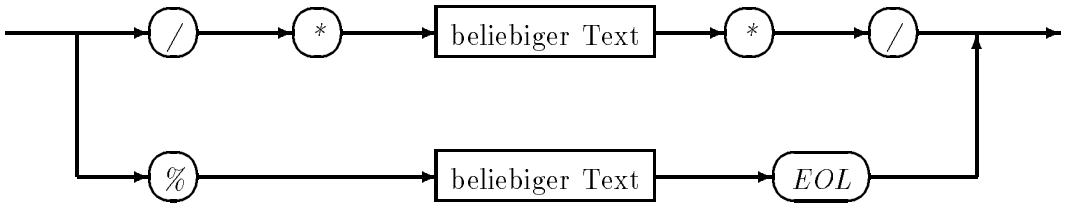
Zahl



! → Einschränkungen:

- Eine Zahl muß innerhalb des Bereichs zwischen −32768 und +32767 liegen.
- Zwischen dem Vorzeichen und der Ziffernfolge darf **kein** Leerzeichen stehen, weil sonst das Vorzeichen als Operationszeichen interpretiert wird.

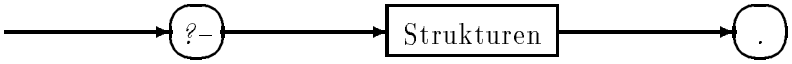
Kommentar



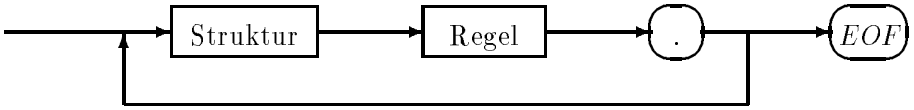
3.3.2 Anfragen und Programme

In den nachfolgenden Syntaxdiagrammen werden Atome, Zahlen und Variablen nur noch als Terminale dargestellt.

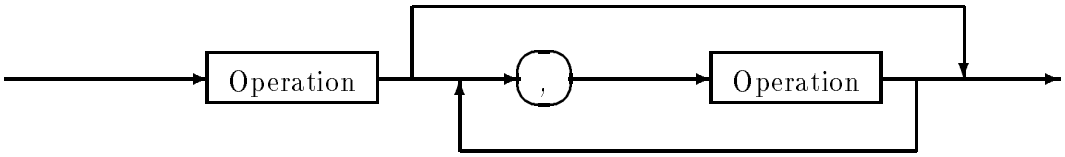
Anfrage



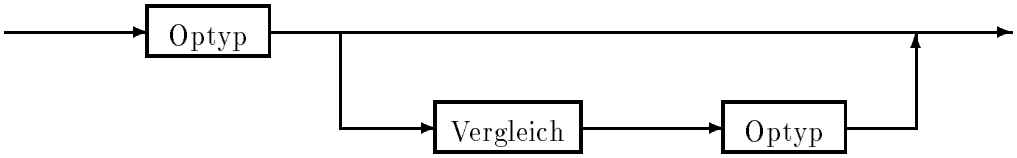
Programm



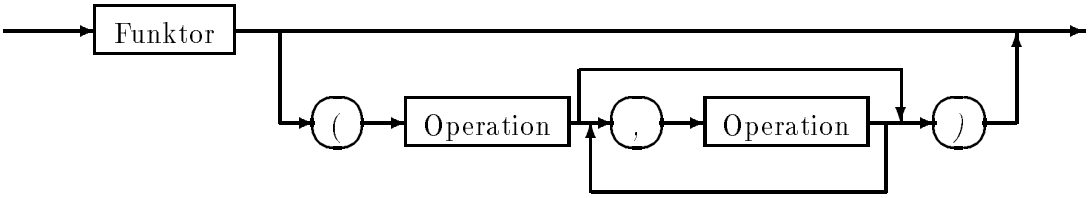
Strukturen



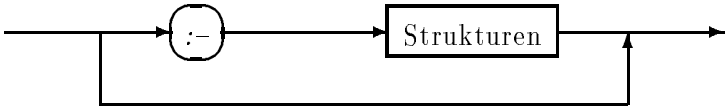
Operation



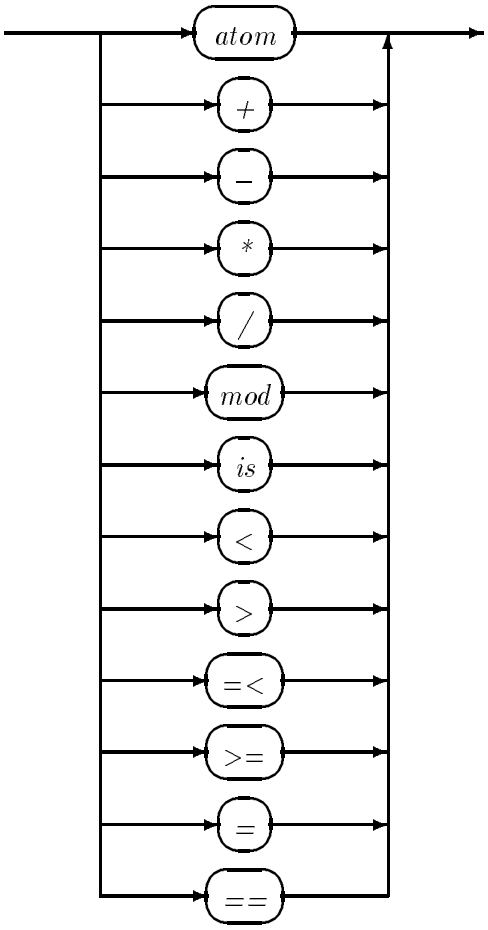
Struktur



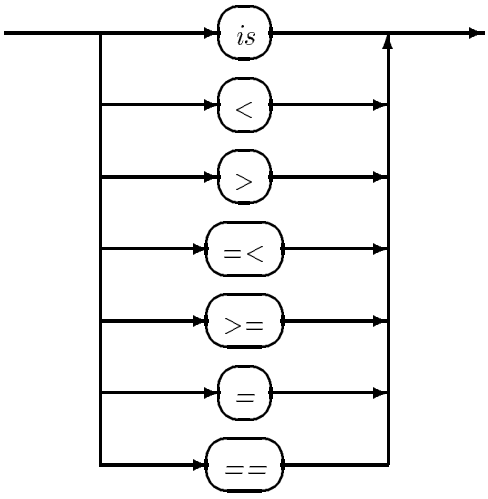
Regel



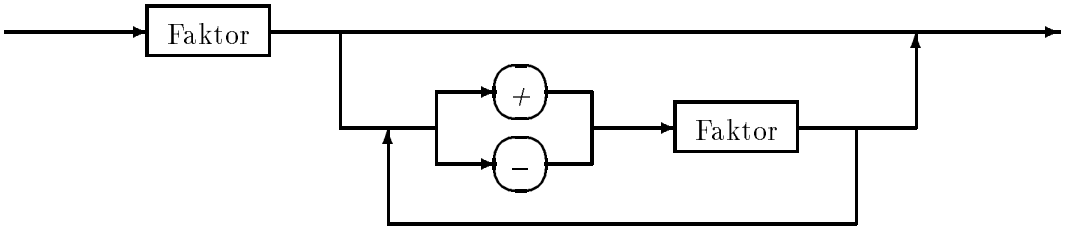
Funktor



Vergleich

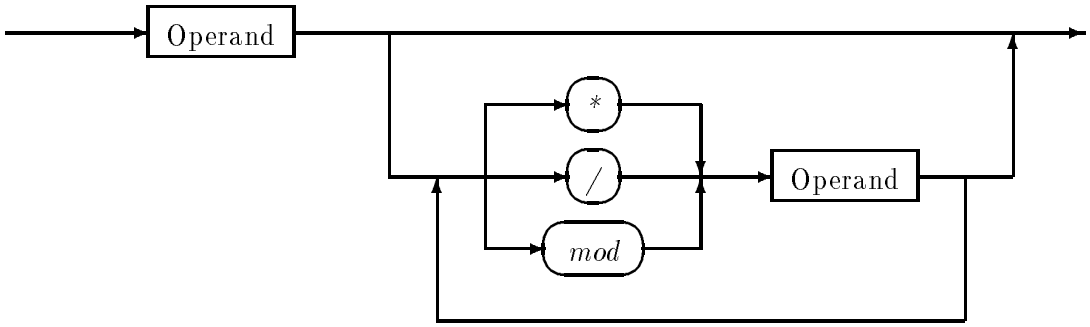


Optyp

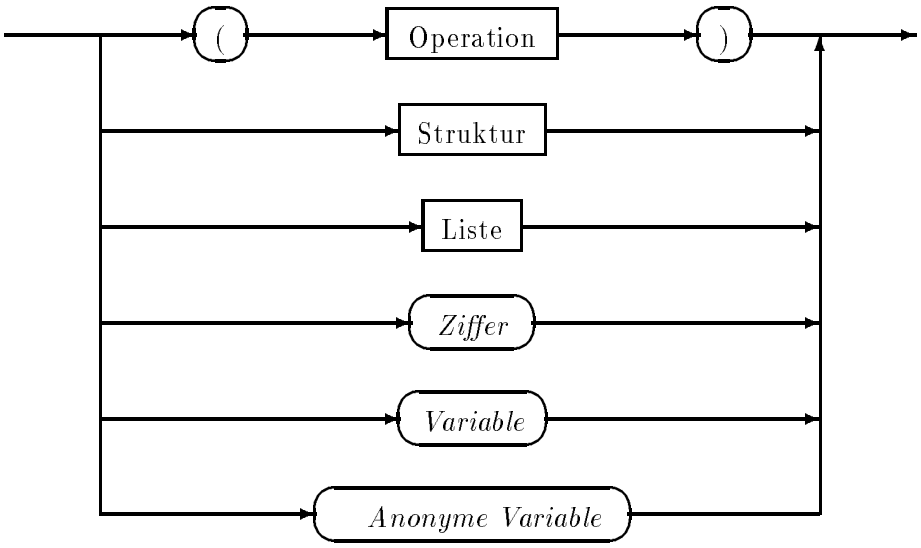


! → Wird hinter dem „+“ bzw. „−“ kein Leerzeichen eingegeben und löst sich das direkt folgende „Faktor“ in eine vorzeichenlose Zahl auf, so wird dies zu einer Zahl mit Vorzeichen zusammengefaßt und für syntaktisch fehlerhaft erklärt. Deshalb muß an dieser Stelle immer nach einem „+“ oder einem „−“ ein Leerzeichen eingegeben werden, wenn darauf eine Zahl folgt!

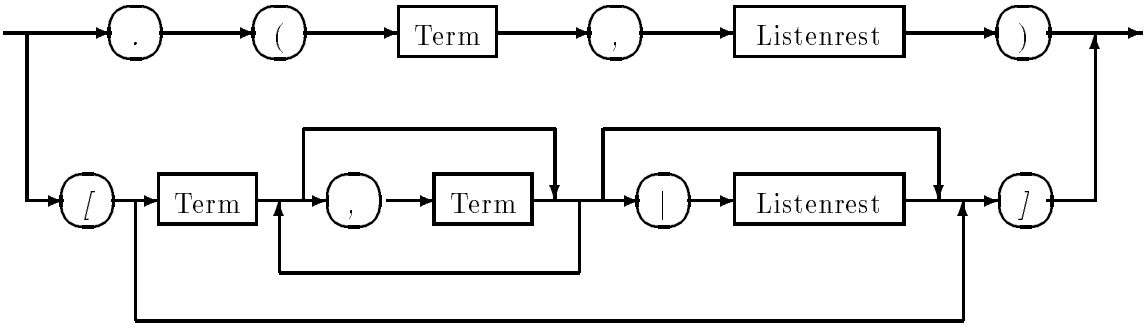
Faktor



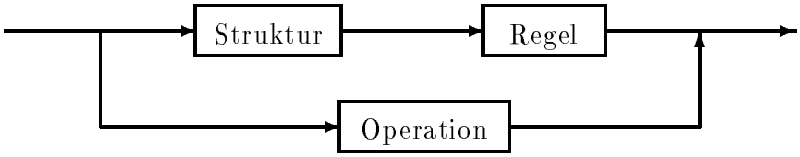
Operand



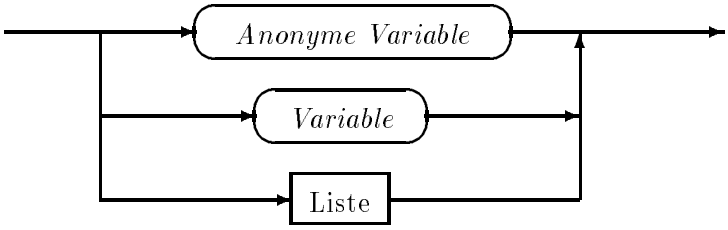
Liste



Term



Listenrest



4. Visualisierung

Wenn in diesem Dokument der Begriff „Visualisierung“ verwendet wird, so meinen wir damit, daß etwas graphisch veranschaulicht wird. In diesem Kapitel wird beschrieben,

- was alles mit ProVisor visualisiert werden kann,
- auf welche Art und Weise etwas visualisiert wird, und
- wie man die verschiedenen Visualisierungsmöglichkeiten aktiviert.

4.1 Übersicht der Visualisierungsmöglichkeiten

Sowohl zum Erlernen von PROLOG als auch zum Debuggen von Programmen sind folgende Fragestellungen sehr wichtig:

- Wie sind PROLOG-Terme aufgebaut?
- Wie arbeitet der Interpreter ein PROLOG-Programm ab?
- Welche Bedeutung hat die Reihenfolge der Klauseln eines Prädikats für den Progammablauf?
- Welche Variablen werden zu welchem Zeitpunkt instantiiert?
- Mit welchem Wert werden sie instantiiert?
- Wie heißt eine Variable, die mit einer Instantiierung überdeckt ist?
- Was heißt „Matchen“ einer Klausel?
- Wie funktioniert Rekursion?
- Was bedeutet Backtracking?
- Wie funktioniert der Cut?

Alle diese Fragestellungen werden durch ProVisor visualisiert und beantwortet. Dazu stellt ProVisor zwei Darstellungen zur Verfügung:

1. **PROLOG-Terme.** Es wird dargestellt, wie PROLOG-Terme intern (also für den Interpreter) aufgebaut sind. Damit wird verdeutlicht, daß sie alle den gleichen Aufbau haben, obwohl sie nach außen hin unterschiedlich aussehen können.

2. **Ablauf eines PROLOG-Programmes.** Dies wird in Form eines Suchbaumes realisiert. Damit ist das dynamische Verhalten eines Programmablaufs nachvollziehbar.

   ProVisor bietet die Möglichkeit, sich den Suchbaum in drei verschiedenen Modi anzeigen zu lassen. Diese unterscheiden sich im Informationsgehalt, der dargestellt wird.

4.2 Termvisualisierung

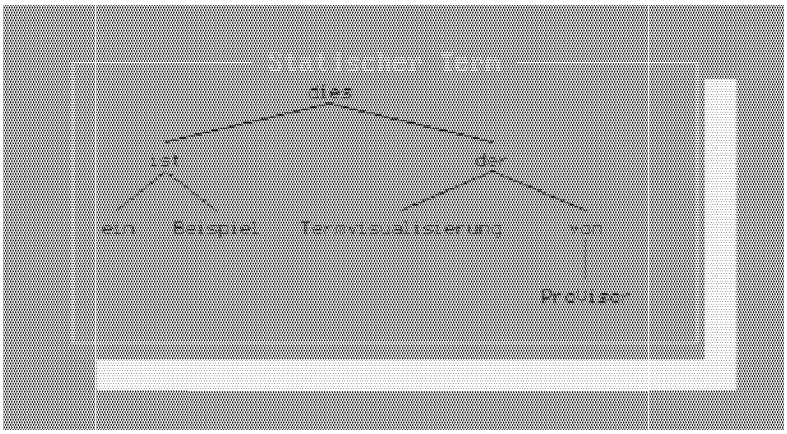
Mit ProVisor können beliebig viele Terme gleichzeitig visualisiert werden, unabhängig davon, ob man die dynamische oder die statische Darstellung wählt. Bevor auf diese beiden Anwendungen eingegangen wird, soll zunächst das Prinzip der Darstellung erläutert werden.

4.2.1 Prinzip der Darstellung

Alle PROLOG-Terme sind entweder Variablen, Zahlen oder Strukturen.

Variablen und **Zahlen** werden textuell ausgegeben.

Strukturen haben einen Funktor und — wenn es sich nicht um ein Atom handelt — auch Argumente, die wiederum beliebige Terme sein können. In der Ausgabe bildet der Funktor die Wurzel und die Argumente die Söhne des Baumes. Ist ein Argument wiederum Funktor eines Teiltermes, so ist dieses Argument Wurzel des Teilbaumes, den dieser Teilterm erzeugt. Das bedeutet, daß der Funktor oberhalb und mittig zu seinen Argumenten plaziert ist und durch eine Linie mit ihnen verbunden ist. Die Abbildung soll dieses Prinzip verdeutlichen:



4.2.2 Dynamische Darstellung

Hiermit lassen sich Terme aus dem Programmlauf visualisieren. Das besondere dabei ist, daß die Visualisierung solcher Terme mit der ersten graphischen Darstellung nicht beendet ist, sondern sie werden während des weiteren Programmablaufs permanent überwacht. Dadurch wird jede Änderung an einem Term erkannt und dieser wird daraufhin neu ausgegeben. Auf diese Weise wird ein ausgewählter Term dynamisch dargestellt. Ist z.B. ein Argument des Terms eine uninstantiierte Variable, so wird zunächst der Variablenname als Argument ausgegeben. Findet im weiteren Programmlauf eine Instantiierung der Variablen statt, so wird daraufhin die Termvisualisierung aktualisiert und somit die Instantiierung sichtbar. Dieses Prinzip ist vergleichbar mit *Watch*-Ausdrücken bei bekannten Debug-Systemen.

4.2.2.1 Auswahl eines Terms

Um einen Term zu visualisieren, muß man ihn aus dem dargestellten Suchbaum auswählen. Dies kann sowohl mit der Tastatur als auch mit der Maus geschehen.

Tastatur

Als erstes muß das Fenster aktiviert werden, aus dem man sich einen Term anzeigen lassen will. Mit den in Tabelle 4.1 aufgelisteten Tastenkombinationen kann jetzt der gewünschte Term markiert werden. Um schließlich den Term zur Visualisierung freizugeben, muß nur noch die RETURN-Taste gedrückt werden. Wenn die Markierung noch nicht sichtbar ist, weil zuvor noch kein Term ausgewählt wurde, so ist immer der aktive Knoten der Ausgangspunkt der ersten Bewegung.

Shift + ↑	gehe zum Vaterknoten
Shift + ←	gehe zum linken Bruder
Shift + →	gehe zum rechten Bruder
Shift + ↓	gehe zum ersten Sohn
Tab	markiere nächstes Argument
Shift + Tab	markiere vorheriges Argument
Return	visualisiere markierten Term

Tabelle 4.1: Tastenkombinationen zur Termauswahl

Nachdem ein Term auf diese Weise ausgewählt wurde, bleibt die Markierung erhalten. Somit können gleich mehrere Terme selektiert werden. Die Markierung verschwindet erst wieder, wenn der Programmlauf fortgesetzt wird.

Maus

Wesentlich einfacher ist die Auswahl eines Terms per Maussteuerung: Klickt man den Funktor des Terms an, so wird der gesamte Term visualisiert. Wenn man sich aber nur ein Argument anzeigen lassen will, so muß das Argument angeklickt werden.

! → Bei der Auswahl sind noch einige Punkte zu beachten:

- Ein Term kann nur dann ausgewählt werden, wenn er schon bzw. wenn er noch existiert. Für den dargestellten Suchbaum bedeutet das, daß nur schon besuchte und nicht fehlgeschlagene Knoten ausgewählt werden können.
- Nach einem Programmlauf lassen sich keine Terme mehr auswählen.
- Es ist nicht möglich, einen Cut-Knoten als Term zu selektieren, weil er eine triviale Struktur besitzt.
- Ist der Übersichtsmodus (siehe Abschnitt 4.3.4) aktiviert, so können mit der Tastatur keine Argumente und mit der Maus überhaupt keine Terme ausgewählt werden.

4.2.3 Statische Darstellung

Von dieser Anwendung wird man wohl weniger Gebrauch machen als von der ersten. Der Unterschied zur obigen liegt darin, daß die Visualisierung eines Terms unabhängig vom Programmlauf stattfindet. Somit verändert sich die Darstellung eines Terms nicht mehr, sobald dieser erst einmal angezeigt ist.

Um die statische Darstellung zu erhalten, muß der Menüpunkt *Visualisierung/Term ...* aktiviert werden. Daraufhin erscheint ein Dialogfenster, in dem der Term dann eingetippt werden kann. Ist der Term syntaktisch korrekt eingegeben (inklusive Punkt am Ende der Eingabe), so wird ein Fenster mit der graphischen Darstellung des Terms geöffnet. Bei einer nicht korrekten Eingabe hingegen wird eine entsprechende Fehlermeldung im Interpreterfenster ausgegeben.

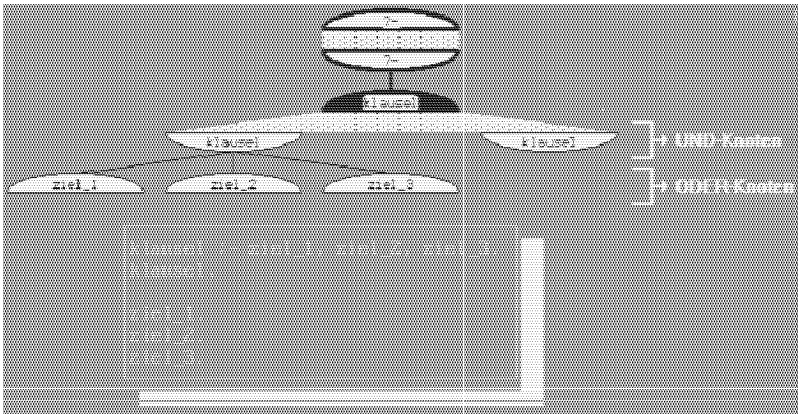
4.3 Visualisierung von Programmabläufen

4.3.1 Die Struktur des Suchbaumes

Das verwendete Visualisierungskonzept interpretiert den Ablauf von Prolog-Programmen als den Aufbau eines Beweisbaumes. Der Beweisbaum, dessen dynamischer Aufbau visualisiert wird, besteht aus UND- und ODER-Knoten. Kennzeichnend für einen UND-Knoten ist, daß er erst dann erfüllt ist, wenn alle seine Nachfolger erfüllt sind. Bei einem ODER-Knoten reicht es, wenn mindestens ein Nachfolger erfüllt ist. Eine solche Darstellung entspricht auch dem Aufbau eines PROLOG-Programms: Klauseln sind disjunktiv und vorhandenene Teilziele einer Klausel konjunktiv miteinander verbunden. Ein UND-Knoten in der graphischen Darstellung „verkörpert" somit ein Klauselkopf und ein ODER-Knoten ein im Klauselrumpf enthaltenes Teilziel.

ODER-Knoten werden durch Kanten an den zugehörigen UND-Knoten angehängt. UND-Knoten werden nebeneinander unterhalb des korrespondierenden ODER-Knotens plaziert. Ihre Zuordnung zum Vaterknoten wird durch eine Schattierung gekennzeichnet.

Ein Effekt der UND-ODER-Baumdarstellung ist, daß bei einem Aufruf einer Klausel stets das ganze aufgerufene Prädikat visualisiert wird.



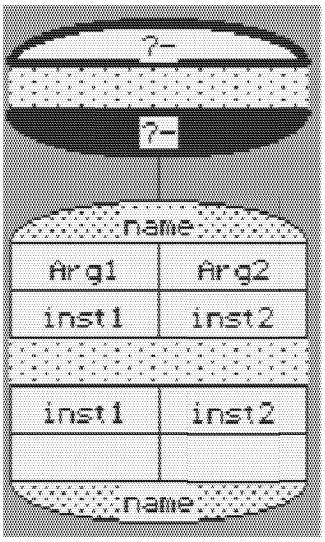
Wie schon zu Beginn des Kapitels erwähnt, existieren mehrere Darstellungsmodi des Suchbaums, die sich im anzeigenden Informationsgehalt voneinander unterscheiden. In den nächsten Abschnitten werden deshalb die einzelnen Modi getrennt voneinander beschrieben.

4.3.2 Der Unifikationsmodus

Alle Knoten sind einheitlich aufgebaut: Aus einem Halboval, das den Namen der Klausel bzw. des Prädikats trägt, und — sofern der Term Argumente hat — aus zwei Parameterleisten. In der oberen Leiste werden die Argumente immer uninstantiiert eingetragen. Enthält ein Argument eine Instantiierung, so wird in der unteren Leiste das Argument noch zusätzlich instantiiert angezeigt.

Die Anfrage

wird durch zwei Halbovale ohne Parameterleisten dargestellt.

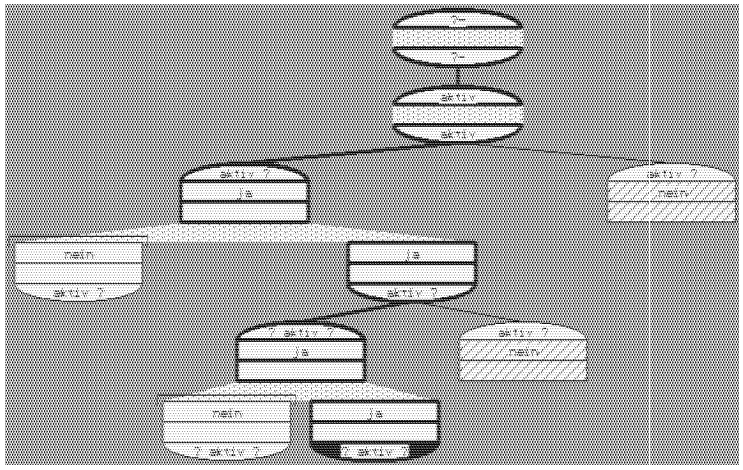


Der aktive Pfad

wird hervorgehoben, indem alle seine Knoten fett umrandet werden. Kanten, die je zwei Knoten des aktiven Pfades berühren, werden ebenfalls fett dargestellt.

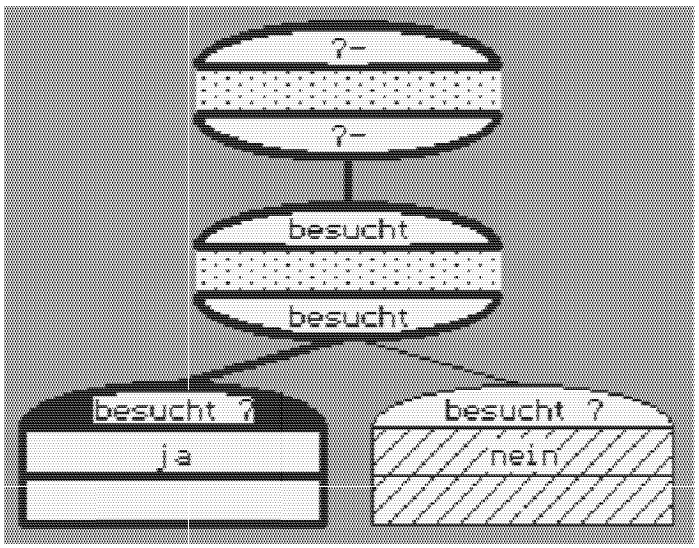
Der aktuelle Knoten

ist der gerade bearbeitete Knoten. Er befindet sich immer am Ende des aktiven Pfades und wird durch inverse Darstellung des Ovals hervorgehoben.



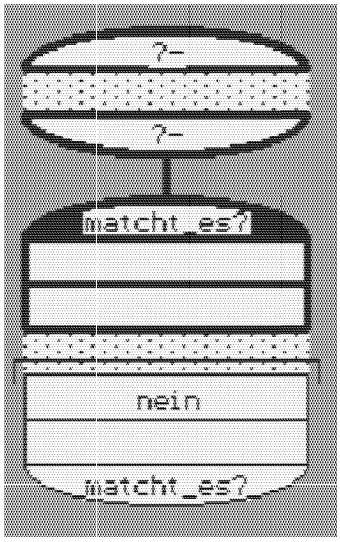
Nicht besuchte Knoten

Bei noch nicht besuchten Knoten oder bei noch ausstehenden Teilzielen wird die Parameterleiste schraffiert dargestellt. Beim Backtracking werden die Knoten, die nochmals zur Berechnung anstehen, ebenfalls so dargestellt. Weil Knoten ohne Argumente keine Parameterleisten haben, wird für sie die Schraffur nicht angezeigt.



Das Nicht-Matchen von Klauseln

wird durch ein horizontales Trennsymbol angedeutet, das sich zwischen das aufrufende Teilziel und die nicht matchende Klausel schiebt.

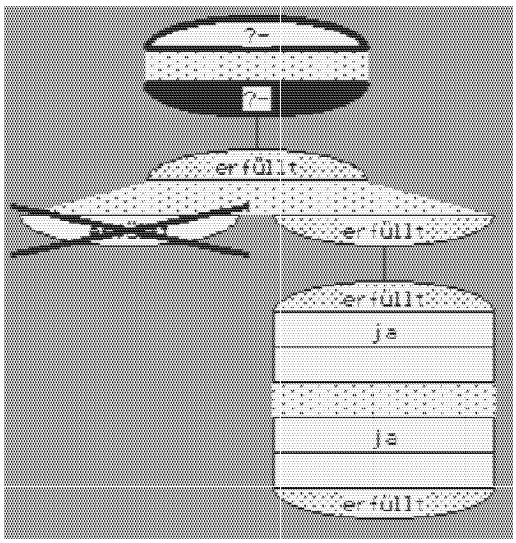


Erfolgreiche Teilziele

Bei Teilzielen, deren Abarbeitung erfolgreich war, werden die Ovale grau unterlegt.

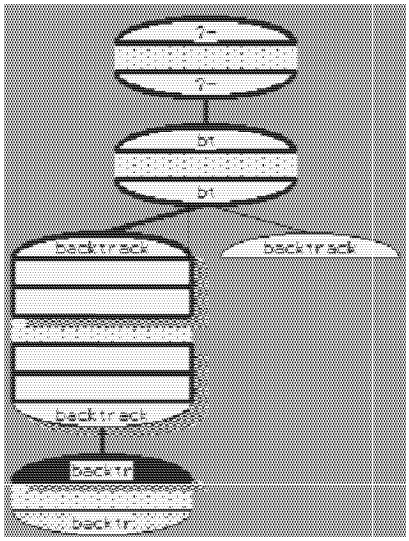
Fehlgeschlagene Teilziele

Die Knoten fehlgeschlagener Teilziele werden zunächst durchgestrichen. Mit Ausnahme des ersten aufgerufenen und fehlgeschlagenen UND-Knotens, der immer durchgestrichen stehen bleibt, werden sie dann beim Rückwärtsgehen sukzessive vom Bildschirm entfernt.



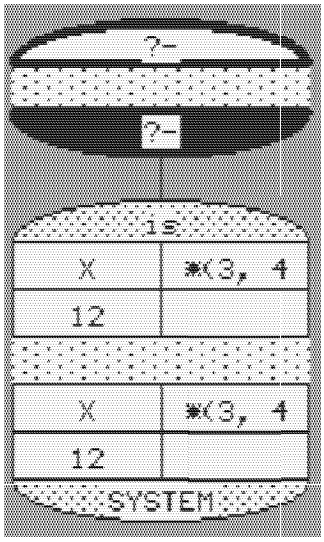
Backtracking

Knoten, die nochmal unifiziert werden müssen (Backtracking), erhalten einen Schatten in der Darstellung.



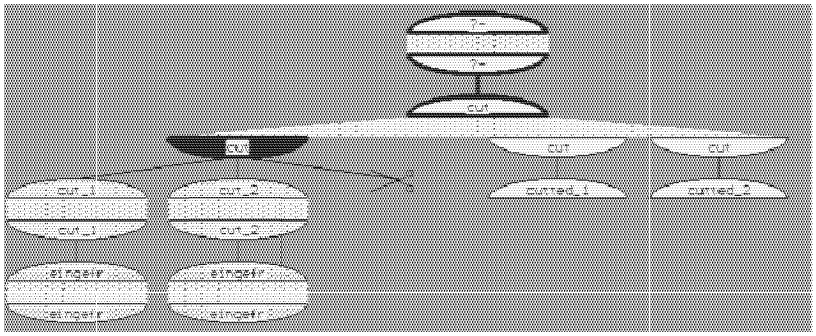
Systemprädikate

werden mit Ausnahme des Cuts und des „:–"-Prädikats einheitlich dargestellt („:–" ist implizit durch die Baumstruktur gegeben). Der korrespondierende UND-Knoten trägt bei den Systemprädikaten die Inschrift „SYSTEM" und hat keine Söhne.

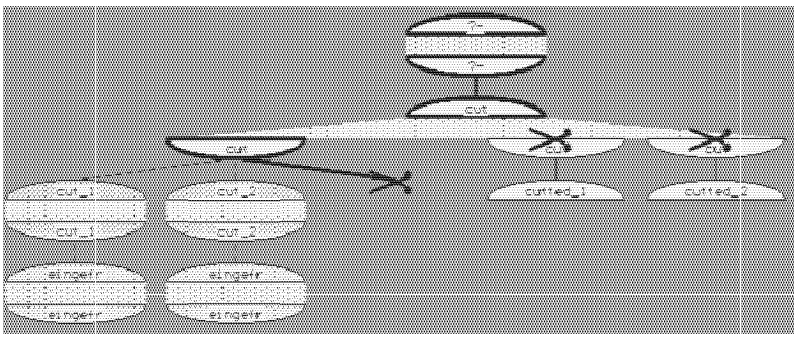


Der Cut

wird durch eine Schere symbolisiert, die statt eines ODER-Knotens in den Baum eingebunden wird. Solange Teilziele links bzw. vor dem Cut berechnet werden, ändert sich an dieser Darstellung nichts.

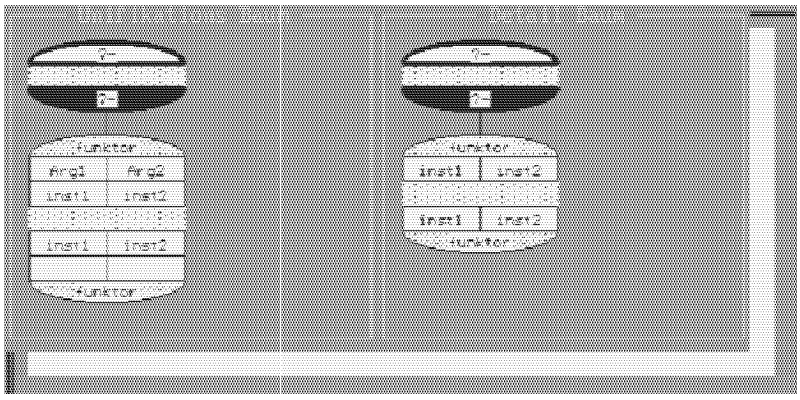


Beim Überschreiten des Cuts von links nach rechts werden alle links liegenden Teilbäume „eingefroren“. Dies wird durch gestrichelte Kanten gekennzeichnet. Die rechtsliegenden Geschwisterknoten des Knotens, in dem der Cut auftrat, werden mit dem Cut-Symbol versehen.



4.3.3 Der Detailmodus

Der einzige Unterschied zum Unifikationsmodus ist, daß die Knoten nur noch eine Parameterleiste haben. Dort werden die Argumente instantiiert eingetragen. Das bedeutet, man kann nicht mehr sehen, ob ein Argument instantiiert ist oder nicht.



4.3.4 Der Übersichtsmodus

Der Übersichtsmodus stellt einen möglichst großen Ausschnitt des Beweisbaumes dar. Bei hinreichend kleinen Beispielen kann der gesamte Baum angezeigt werden. Im Gegensatz zu den beiden vorherigen Modi werden hier keine Knoteninformationen (Funktorname und Argumente) angezeigt. Die Symbolik der Darstellung bleibt allerdings weitgehend erhalten. Deshalb werden hier nur die Abweichungen beschrieben.

Nicht Besuchte Knoten

werden auch in diesem Modus nur dann kenntlich gemacht, wenn sie mindestens ein Argument haben, weil sonst keine Parameterleiste gezeichnet wird. Wurde ein Knoten noch nicht besucht, so wird die Parameterleiste geschlossen dargestellt. Bei besuchten Knoten hingegen ist sie nach oben oder nach unten offen.

Das Nicht-Matchen

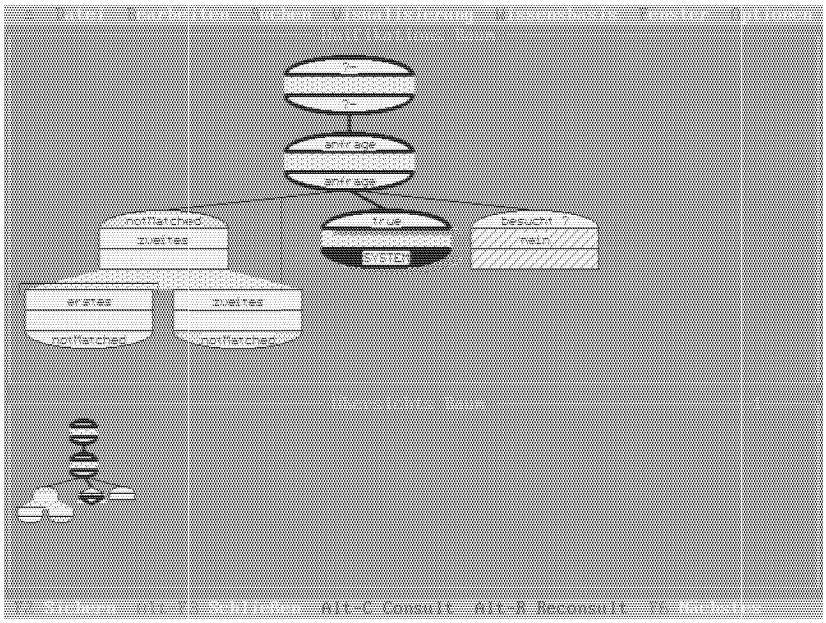
wird durch einen einfachen Strich über dem entsprechenden UND-Knoten angedeutet.

Backtracking

Knoten, über die eine Reerfüllung (Backtracking) stattfindet, werden als Dreiecke dargestellt.

Systemprädikate

werden mit Ausnahme des Cuts nicht mehr besonders dargestellt. Der Cut wird analog zu den beiden anderen Modi realisiert.



4.3.5 Aktivierung

Im Menü *Visualisierung* sind alle drei Darstellungsmodi aufgelistet. Damit lassen sich zu bereits bestehenden Darstellungen des Suchbaums neue hinzufügen. Maximal lassen sich aber nur drei Darstellungen (weil es ja auch nur drei Modi gibt) aktivieren. In einem solchen Fall werden dann die Menüpunkte zu den drei Modis gesperrt, so daß kein weiteres Fenster mehr geöffnet werden kann.

Mit ProVisor ist es aber auch möglich, sich einen Darstellungsmodus in einen anderen umwandeln zu lassen. Dazu muß zuerst das Fenster mit dem nicht mehr gewünschten Modus aktiviert werden. Wählt man jetzt den Menüpunkt *Visualisierung/Darstellung ändern* an, so braucht man nur noch den neuen Darstellungsmodus auszuwählen. Im aktivierten Fenster erscheint daraufhin die neue Darstellung des Suchbaumes. Wenn das aktive Fenster keine Darstellung des Suchbaums enthält (z.B. das Interpreterfenster ist aktiv), so wird dieser Menüpunkt gesperrt.

5. Steuerung des PROLOG-Programmlaufs

ProVisor bietet zwei Möglichkeiten der Beeinflussung des Programmlaufs: die interaktive Steuerung durch den Suchbaum mittels Einzelschritt, Überspringen, Weiter und Stop; sowie ein umfangreiches Breakpoint-System, mit dem bestimmte Programmstellen schnell angesteuert werden können. Diese beiden Möglichkeiten werden in den folgenden beiden Abschnitten beschrieben.

5.1 Schrittweise Steuerung

Nachdem eine Anfrage gestellt und visualisiert wurde, bleibt ProVisor stehen. Es gibt folgende Möglichkeiten, fortzufahren:

Einzelschritt

Der Befehl Einzelschritt (F7 oder Menü *Visualisierung-Einzelschritt*) bewirkt, daß der PROLOG-Interpreter einen einzigen Schritt ausführt und sofort wieder stehenbleibt. Bei schrittweiser Ausführung wird die Visualisierung detailliert angezeigt, wofür Zwischenschritte (in denen keine Ausgaben ins Interpreterfenster erfolgen) verwendet werden.

Überspringen

Die Wirkung von Überspringen (F8 oder Menü *Visualisierung-Überspringen*) ist abhängig vom aktuellen Zustand des Interpreters. Wird im Interpreterfenster als letzter Schritt CALL oder REDO angezeigt, wird der Programmlauf erst wieder gestoppt, wenn ein EXIT oder FAIL auf das angezeigte Ziel erfolgt. Es ist also möglich, bekannte oder nicht interessierende Teilberechnungen schnell zu überspringen. Der TRACE-Lauf wird nach wie vor im Interpreterfenster protokolliert, Graphiken werden allerdings nicht ständig dem neuesten Stand angepaßt (dies geschieht erst, wenn der Überspringen-Befehl beendet wird). Bei Erfüllung der Anfrage wird auf jeden Fall der PROLOG-Lauf gestoppt.

Wird als letzter Schritt EXIT oder FAIL angezeigt, entspricht die Wirkung von Überspringen der Wirkung von Einzelschritt.

Stop

Der Befehl Stop hält den PROLOG-Interpreter an. Eine weitere Steuerung erfolgt dann wieder über Einzelschritt, Überspringen oder Weiter.

Weiter

Weiter bewirkt, daß der Interpreterlauf ohne weitere Unterbrechung weiterläuft, bis

- der Befehl Stop aktiviert wird,

- die Anfrage erfüllt wird oder fehlschlägt,
- ein Breakpoint erreicht wird.

Wie bei Überspringen wird der PROLOG-Trace im Interpreterfenster angezeigt, die Visualisierung aber nicht ständig nachgezogen.

5.2 Breakpoints

Breakpoints werden verwendet, um schnell an interessierende Suchbaum-Stellen zu gelangen. Sie sind unabhängig von einem PROLOG-Lauf oder der Wissensbasis. Bei jedem der Ports CALL, EXIT, REDO und FAIL schaut der Interpreter nach, ob der jeweilige Zielterm mit einem der gesetzten Breakpoints übereinstimmt. Ist dies der Fall, wird der PROLOG-Lauf gestoppt und der aktuelle Zustand des Interpreters in den Grafikfenstern angezeigt. Breakpoints in ProVisor unterscheiden sich etwas von Breakpoints, die in imperativen Programmiersprachen (wie z.B. Pascal) zum Debuggen verwendet werden. Dort können Breakpoints auf einzelne Befehle gesetzt werden; das Programm stoppt, wenn in der sequentiellen Ausführung des Programms diese Ziele erreicht werden. In ProVisor werden Breakpoints auf Klauseln gesetzt, wobei Argumente und Durchlauf-Anzahl zur näheren Spezifikation eines gewünschten Stopps angegeben werden können.

Zur Verwaltung der Breakpoints gelangt man über das Menü *Visualisierung-Breakpoints*. Hier können Breakpoints jederzeit gesetzt, editiert und gelöscht werden:

Jeder gesetzte Breakpoint wird in einer Zeile dargestellt. Die Dialogbox in der Abbildung 5.1 ist eine reine Anzeige der gesetzten Breakpoints. In der ersten Spalte befinden sich die PROLOG-Terme als Hauptbestandteil eines Breakpoints. In der *When*-Spalte können folgende Werte stehen:

CALL	Beim Aufruf des Ziels wird gestoppt.
EXIT	Beim Verlassen nach erfolgreicher Erfüllung des Ziels wird gestoppt.
REDO	Bei einem Reerfüllungsversuch wird gestoppt.
FAIL	Bei einem Fehlschlag wird gestoppt.

Soll ein Breakpoint bei mehreren Ports aktiv sein, müssen mehrere Breakpoints (die sich nur in der Port-Information unterscheiden), erzeugt werden. Die Unify-Spalte enthält ein Flag. Ist Unify angegeben, können im Breakpoint-Term enthaltene Variablen beim Test, ob ein Ziel im PROLOG-Lauf einen Breakpoint erfüllt, mit nicht-freien Termen unifiziert werden. Ist Unify nicht angegeben, müssen die Breakpoint-Variablen bei dem Test frei bleiben (dies wird im Beispiel deutlicher). Die Namen der Breakpointvaria-

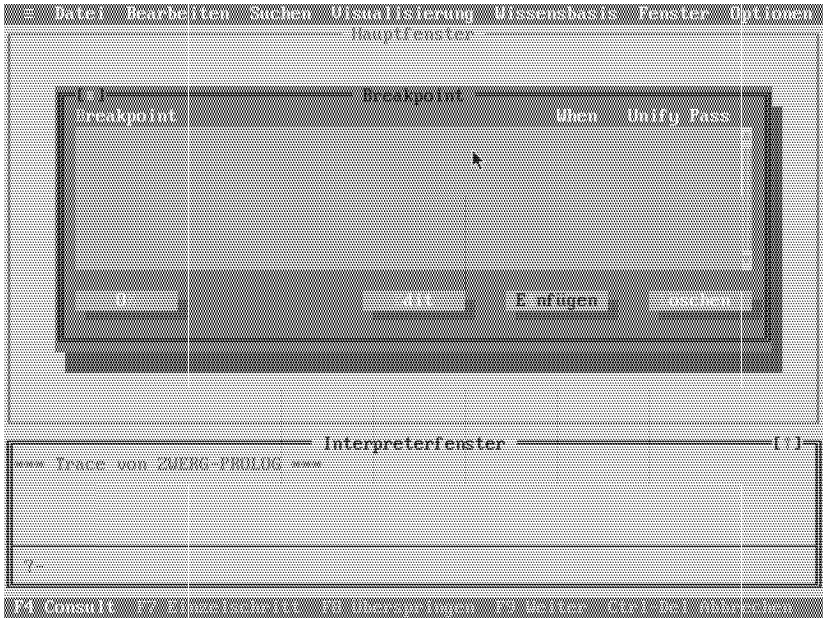


Abbildung 5.1: Dialog-Fenster
Breakpoints (Menü *Visualisierung-Breakpoints*)

blen ist beliebig. In der letzten Spalte (Pass) wird angegeben, beim wievielten Durchlauf durch den Breakpoint angehalten werden soll. Wird 0 (Null) angegeben, dann stoppt der Interpreter bei jeder Breakpoint-Erfüllung, bei einer 1 nur das erste Mal (und dann im gleichen PROLOG-Lauf nicht wieder), bei einer 2 nur das zweite Mal usw. Durch die Verwendung von Pass kann jede beliebige Stelle eines PROLOG-Traces eindeutig angesteuert werden.

Die möglichen Aktionen sind:

Markieren eines Breakpoints Alt-B + Cursortasten bzw. Anklicken mit der Maus.

OK — Verlassen des Breakpoint-Fensters mit Übernahme von Änderungen

Edit — Markierten Breakpoint verändern (dazu wird ein weiteres Fenster geöffnet, siehe Bild 5.1)

Einfügen — Einen neuen Breakpoint einfügen (in einem weiteren Fenster, siehe Bild 5.2)

Löschen — Markierten Breakpoint löschen.

Beispiel 1: Wir wollen nun das Breakpoint-System an zwei Beispielen näher ausführen:

- Laden Sie die Datei FAMILY.PRO (z. B. über den Befehl reconsult).

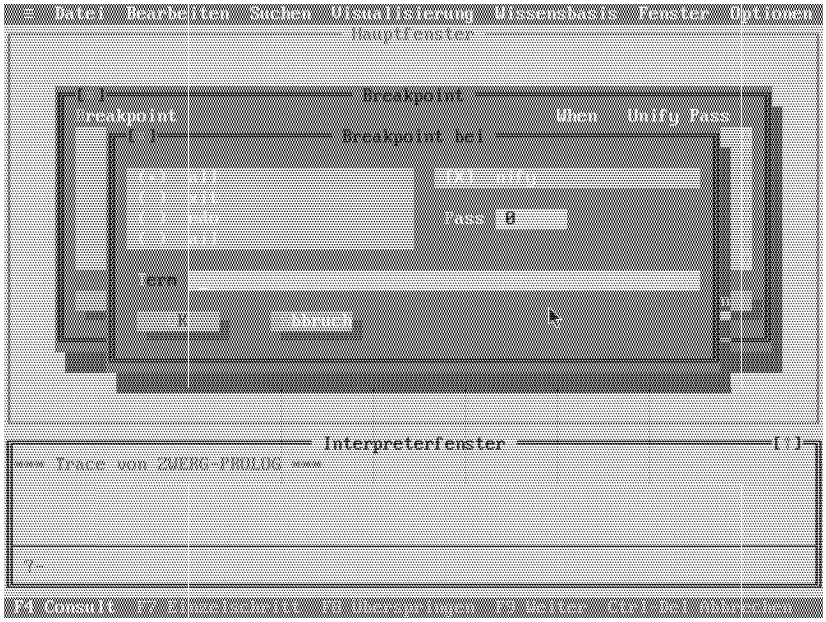


Abbildung 5.2: Dialog-Fenster *Breakpoints-Editieren/Einfügen*

Uns interessieren jetzt nur die Programmstellen, an denen das Prädikat "ist_Mutter_von" aufgerufen wird.

- Wählen Sie den Menü-Punkt *Visualisierung/Breakpoints*. Es erscheint ein Dialogfenster, in dem alle im Moment gesetzten (also keine) Breakpoints aufgelistet sind.
- Wählen Sie *Einfügen*, um einen neuen Breakpoint zu setzen. Es erscheint ein weiteres Fenster.
- Geben Sie

  ```
  ist_Mutter_von(X,Y)
  ```

  als Breakpoint-Term an (die Variablennamen können beliebig gewählt werden, die beiden Namen müssen sich nur unterscheiden).

  Man kann sich vorstellen, daß der Interpreter stoppt, wenn der Breakpoint-Term als Ziel an einem der vier Ports CALL, EXIT, REDO oder FAIL auftaucht.

  Es sind weitere Einschränkungen der Stop-Bedingung vorgesehen:

  – Als Port ist in der Voreinstellung CALL angegeben, das unserer Absicht entspricht. Der Interpreter stoppt jetzt nur noch bei einem CALL auf "ist_Mutter_von".

  - Die Option *Unify* ist angegeben; ProVisor stoppt also (weil im Term Variablen verwendet wurden) bei jedem CALL-Aufruf von "ist_Mutter_von".
  - *Pass* ist auf 0 gesetzt, welches keine weitere Einschränkung bewirkt.

- Mit *OK* verlassen Sie das Fenster *Breakpoint bei*. War der eingegebene Breakpoint-Term syntaktisch korrekt, erscheint nun Ihr Breakpoint in der ersten Zeile. Wählen Sie nochmal *OK*, um auch dieses Fenster zu schließen.

- Aktivieren Sie das Interpreterfenster und stellen Sie die Anfrage "ist_Kind_von(theresa,X)". Der Anfrageknoten wird in den geöffneten Grafikfenstern dargestellt.

  Gewöhnlich würde jetzt ein *Einzelschritt* oder *Überspringen* gewählt, um den Suchbaum zu traversieren. Da uns aber nur die Aufrufe eines bestimmten Prädikates interessieren und wir einen Breakpoint gesetzt haben, kann direkt *Weiter* (F9) gewählt werden.

  Wenn das Trace-Protokoll nicht ausgeschaltet ist, können Sie den Trace im Interpreterfenster verfolgen.

- ProVisor stoppt gleich nach dem ersten CALL und zeigt auf dem Bildschirm ein Info-Fenster mit einer Meldung (siehe Abbildung 5.3).

  Inhalt der Meldung ist der Breakpoint-Term, den Sie vorhin eingetippt haben. Dieses Fenster können Sie verschieben, um die Situation **vor** dem Aufruf des Terms zu betrachten. Wenn Sie *OK* wählen, wird der Schritt

  ```
  CALL: ist_Mutter_von(X, theresa)
  ```

  ausgeführt und der Interpreter stoppt.

- Wählen Sie *Weiter* (F9), um den PROLOG-Lauf fortzusetzen.

  Der Interpreter findet die erste Lösung

  ```
  X = ulrich.
  ```

  Lassen Sie eine weitere Lösung suchen, indem Sie Return drücken oder mit der Maus auf *OK* klicken. Der Interpreter stoppt immer nach jeder Lösung; wählen Sie einfach *Weiter* (F9), um den PROLOG-Lauf fortzusetzen.

- Weitere Unterbrechungen, die durch den Breakpoint verursacht werden, sind die Aufrufe

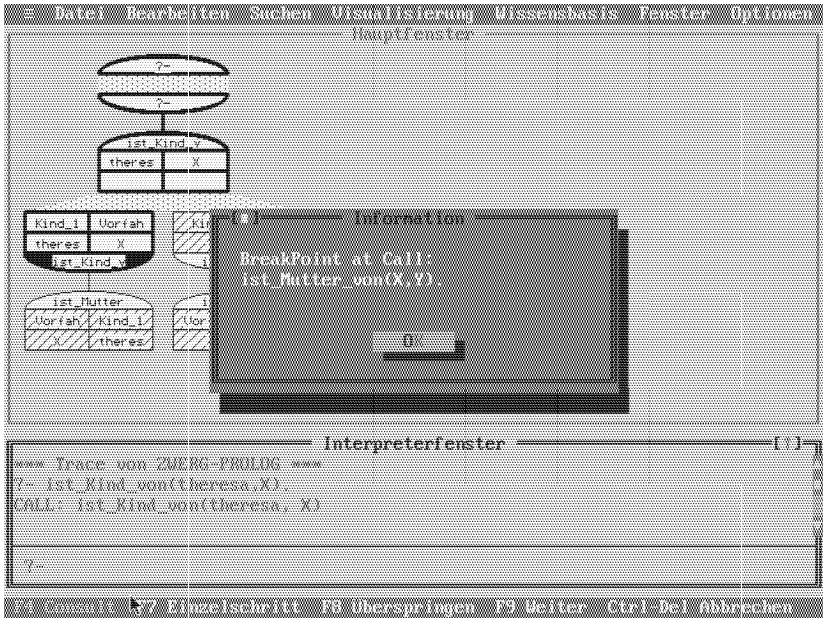


Abbildung 5.3: Breakpoint aktiviert

```
CALL: ist_Mutter_von(kathi, theresa)
CALL: ist_Mutter_von(conny, theresa)
CALL: ist_Mutter_von(lotte, theresa)
```

Beispiel 2: Im zweiten Beispiel interessiert uns nur der Aufruf

```
CALL: ist_Mutter_von(conny, theresa).
```

- Rufen Sie dazu das Menu Visualisierung/Breakpoint auf. Markieren Sie den Breakpoint, und wählen Sie *Edit*, um diesen Breakpoint zu verändern. Es gibt nun mehrere Möglichkeiten, das Ziel zu erreichen:
  - Sie können den Pass auf 3 setzen (da uns nur der dritte Aufruf interessiert)
  - Sie können den Breakpoint-Term in

    ```
    ist_Mutter_von(conny, theresa)
    ```

    ändern. Die Angabe *Unify* ist hierbei irrelevant, wenn im Breakpoint-Term keine Variablen vorhanden sind.
- Wählen Sie "OK", um die Änderung zu übernehmen. Der Interpreter sollte jetzt das gewünschte Verhalten zeigen.

Anhang A. Glossar

aktiver Pfad	Der aktive Pfad besteht aus allen Knoten und Kanten, die zwischen der Wurzel des Baums (der Anfrage) und dem aktuellen Knoten (s.d.) liegen.
aktiver, aktueller Knoten	Der Knoten, der laut PROLOG-Fahrplan als nächstes eine Änderung erfährt (z.B. Expandierung).
Backtracking	Zurückgehen zu einem Ziel im Beweisbaum, um dieses neu zu erfüllen. Tritt ein, wenn ein Ziel fehlgeschlagen ist.
Breakpoint	Ein Breakpoint ist eine Situation, in der der PROLOG-Interpreter stoppt, wenn eine angegebene Bedingung erfüllt ist. Bei PROLOG-Interpretern werden Breakpoints durch PROLOG-Terme angegeben; es wird gestoppt, wenn dieser Term als Ziel auftaucht.
Disjunktion	Ist eine Bedingung erfüllt, wenn mindestens eine Teilbedingung erfüllt ist, spricht man von Disjunktion. Diese entsprechen im Visualisierungskonzept den ODER-Knoten.
Expandieren eines Knotens	Wenn ein ODER-Knoten expandiert wird, wird das gesamte Prädikat auf einmal angezeigt (im Gegensatz zu konventionellen Trace-Protokollen, die klauselorientiert arbeiten). UND-Knoten können in diesem Sinn nicht expandiert werden.
Fakt, Faktum	Ein Eintrag in die PROLOG-Wissensbasis, der kein ": –" enthält. Matcht ein Faktum, ist das aktuell bearbeitete Teilziel sofort erfüllt.
Instantiierung	Während der Durchsuchung der Wissensbasis durch den PROLOG-Interpreter werden Variablen durch Terme ersetzt. Dieser Prozeß wird Instantiierung genannt.
Klausel	Klausel ist der Oberbegriff für Fakten und Regeln. Die Einträge in die Wissensbasis sind Klauseln.
Klauselkopf	Bezeichnet das erste Argument des Systemprädikats ": –".
Klauselrumpf/–körper	Bezeichnet das zweite Argument des Systemprädikats ": –".
Konjunktion	Eine Konjunktion ist erfüllt, wenn alle Teilbedingungen erfüllt sind. Diese entspricht einem UND-Knoten.
Matchen	Ein PROLOG-Ziel "matched" mit einer Klausel, wenn der Klauselkopf mit dem Ziel unifizierbar ist.

ODER-Knoten	Im Zusammenhang mit der Visualisierung: eine unten flache Halbellipse. (vgl. Disjunktion)
Prädikat	Ein Prädikat besteht aus der Menge der Klauseln mit gleichem Funktor und gleicher Stelligkeit.
Regel	Ein Ziel, das mit einer Regel matcht, ist nur erfüllt, wenn alle Teilziele erfüllbar sind, die im Rumpf der Regel aufgeführt sind.
Struktur	Eine PROLOG-Struktur besteht aus einem Funktor und Argumenten. Ist die Stelligkeit null, hat die Struktur keine Argumente und ist somit ein Atom.
Term	Oberbegriff von Struktur, Variable und Integer.
UND-Knoten	Im Zusammenhang mit der Visualisierung: eine oben flache Halbellipse. (vgl. Konjunktion)
Unifikation	Prozeß der "Gleichmachung" von Termen, u.U. durch Instantiierung von Variablen.
Wissensbasis	Mit diesen Begriffen wird die interne Daten-/Programm-Repräsentation des Interpreters bezeichnet, die aus Klauseln besteht, auf die während des Inferenzmechanismus zur Ableitung der Anfrage zugegriffen wird.

Literaturverzeichnis

[BRA87] Bratko, I .: PROLOG, Addison-Wesley, 1. Aufl. 1987

[CLO87] Clocksin, W. J. und Mellish, C. S. : Programming in PROLOG, Springer, 3. Aufl 1987

[KLE86] Kleine, Büning, Schmitgen: PROLOG, Teubner 1986

[NIL81] Nillson, N. J. : Principles of Artificial Intelligence, Tioga 1981

[ROE92] Röhner, G. : PROLOG, Lehrerweiterbildung Informatik, HIBS, HILF

[SHA88] Shapiro, E. und Sterling, L. : The Art of PROLOG, MIT-Press