

Erste Schritte in SWI-Prolog

Die nachfolgenden Abschnitte sollen ein erster Einstieg in den Umgang mit Prolog sein, der sich explizit auf die in unserer Veranstaltung 'Prolog für Linguisten' verwendete Programmversion (SWI-Prolog Vers. 3.3.4) bezieht: es geht um den ersten konkreten Umgang mit dem Programm.

Damit wir mit Prolog überhaupt etwas machen können, müssen zuerst kurz ein paar Termini eingeführt werden, Termini, die in diesem Text nicht näher erläutert werden können – dafür wird auf andere Kapitel verwiesen. Wenn also bestimmte Begriffe in diesem Text nicht so ganz klar werden, ist das überhaupt nicht tragisch, hier geht es nur darum, raschen Zugang zum praktischen Umgang mit Prolog zu gewinnen.

Prolog kann sehr grob in zwei Komponenten eingeteilt werden: den Programmkern, also derjenige Programmteil, in welchem mit der Inferenzmaschine 'gerechnet' wird, und die Wissensbasis. In der Wissensbasis sind die Daten enthalten, mit denen der Programmkern arbeitet. Diese Daten können in zwei Gruppen gegliedert werden: bestimmte eingebaute Daten, die vorgefertigt und quasi ab Programmstart verfügbar sind, und Daten, die erst in die Prolog Wissensbasis eingelesen werden müssen. Die letzte Gruppe ist hier von Interesse, denn hierbei handelt es sich genau um Daten, die von Benutzern (= Euch) geschrieben wurden. Die Daten müssen in einer bestimmten Form eingegeben werden, damit ein Programm damit etwas anfangen kann, und die allgemeine Form von Daten in Prolog ist die der **KLAUSEL**.

Eine Klausel ist in Form einer Funktor/Argumentstruktur notiert, d.h. sie besteht aus einem Funktor und ggf. einer Reihe von Argumenten. Die Notationsweise einer Klausel sieht so aus:

funktor(argument₁, argument₂, ..., argument_n).

Zur Linken steht der Funktor, dahinter, in runden Klammern, die Argumente (falls vorhanden). Die Menge aller Klauseln, die denselben Prädikatsnamen und dieselbe Zahl von Argumenten haben, definieren ein PRÄDIKAT (für eine Definition dazu siehe Kapitel 3).

Beispiele (auf den Inhalt kommt es hier nicht an):

- 1) verb(walk).
 verb(run).
 verb(kiss).
- 2) parent(john,fred).
 parent(mary,fred).
- 3) append(L1, L2, L3).

usw. Hierbei unbedingt die Groß/Kleinschreibung beachten und am Ende den Punkt nicht vergessen!

Ein wichtiger Begriff im Zusammenhang mit Prädikaten ist der Begriff der **STELLIGKEIT**, der sich auf die mögliche Zahl der Argumente eines Prädikates bezieht. In den oa. Beispielen liegen die folgenden Stelligkeiten vor:

- 1) verb hat die Stelligkeit 1, denn jede Klausel, die als Prädikatsnamen verb hat, hat 1 Argument. Schreibweise des Prädikates: verb/1.
- 2) parent hat die Stelligkeit 2, denn jede Klausel, die als Prädikatsnamen parent hat, hat 2 Argumente. Schreibweise des Prädikates: parent/2
- 3) append hat die Stelligkeit 3, denn es hat 3 Argumente. Schreibweise des Prädikates: append/3 usw.

Man kann also folgendes aussagen: die unter 1) geführten Klauseln definieren das Prädikat verb/1; die unter 2) geführten Klauseln definieren das Prädikat parent/2. Es gibt übrigens auch Prädikate, die kein Argument haben, es handelt sich dabei um nullstellige Prädikate. Die zentrale Frage ist: wie werden solche Klauseln bzw. Mengen von Klauseln eingegeben und wie gelangen diese Daten in die Prolog-Wissensbasis? Die Antwort auf diese Frage ist der Gegenstand des vorliegenden Textes.

Die Eingabe von Daten in Prolog

Möglichkeit (A): Eingabe direkt über die Prolog-Oberfläche

Wenn SWI-Prolog gestartet wird, erscheint das folgende Fenster auf dem Monitor (möglicherweise in einer anderen Größe):



```

SWI-Prolog [version 3.3.4]
Welcome to SWI-Prolog (Version 3.3.4)
Copyright (c) 1990-2000 University of Amsterdam.
Copy policy: GPL-2 (see www.gnu.org)

For help, use ?- help(Topic). or ?- apropos(Word).

?- █

```

Abb. 1.1. die Prolog-Oberfläche nach dem Programmstart.

Wichtig ist in diesem Fenster die Zeile, die die Eingabeaufforderung `?-` und die (blinkende) Schreibmarke enthält, denn hinter diese werden die Befehle eingegeben.

Um eine Klauseln direkt auf der Prolog-Oberfläche einzugeben, wird das eingebaute Prädikat `assert/1` verwendet. Das engl. *assert* bedeutet soviel wie 'feststellen', 'behaupten'. Das Argument von `assert/1` ist das jeweilige Prädikat, welches in die Wissensbasis geschrieben werden soll.

Beispiel: Durch die Eingabe

`?- assert(verb(walk)).`

wird das Argument von `assert/1`, also die Klausel `verb(walk)` in die Wissensbasis geschrieben. Daß dieser Vorgang stattgefunden hat, wird vom Programm durch die Angabe **Yes** bestätigt:



```

SWI-Prolog [version 3.3.4]
Welcome to SWI-Prolog (Version 3.3.4)
Copyright (c) 1990-2000 University of Amsterdam.
Copy policy: GPL-2 (see www.gnu.org)

For help, use ?- help(Topic). or ?- apropos(Word).

?- assert(verb(walk)).
Yes
?-

```

Abb. 1.2. Eingabe einer Klausel auf der Prolog-Oberfläche



Wenn Ihr das Skript am PC durcharbeitet, könnt Ihr jetzt die **Aufgabe 1** am Kapitelende angehen

Mit `assert/1` können, wie gesehen, Klauseln zeilenweise in die Wissensbasis geschrieben werden. (Es gibt auch noch andere Möglichkeiten für die Eingabe von Daten auf der Prolog-Oberfläche.)

Das Überprüfen der Wissensbasis

Wenn die Prolog-Wissensbasis um die entsprechenden Klauseln aufgestockt ist, können Anfragen an das Programm gestellt werden, die sich auf diese Klauseln beziehen. So können beispielsweise durch die Verwendung einer Variablen (die in Prolog durch Zeichenfolgen, die mit einem Grossbuchstaben beginnen, notiert werden) alle Argumente eines Prädikates 'erfragt' werden. Auf unser Beispiel bezogen: die Eingabe `verb(X)` liefert als Ergebnis die Bindung der Variablen `X` an das bzw. die entsprechenden Argumente:



```

SWI-Prolog [version 3.3.4]
Welcome to SWI-Prolog (Version 3.3.4)
Copyright (c) 1990-2000 University of Amsterdam.
Copy policy: GPL-2 (see www.gnu.org)

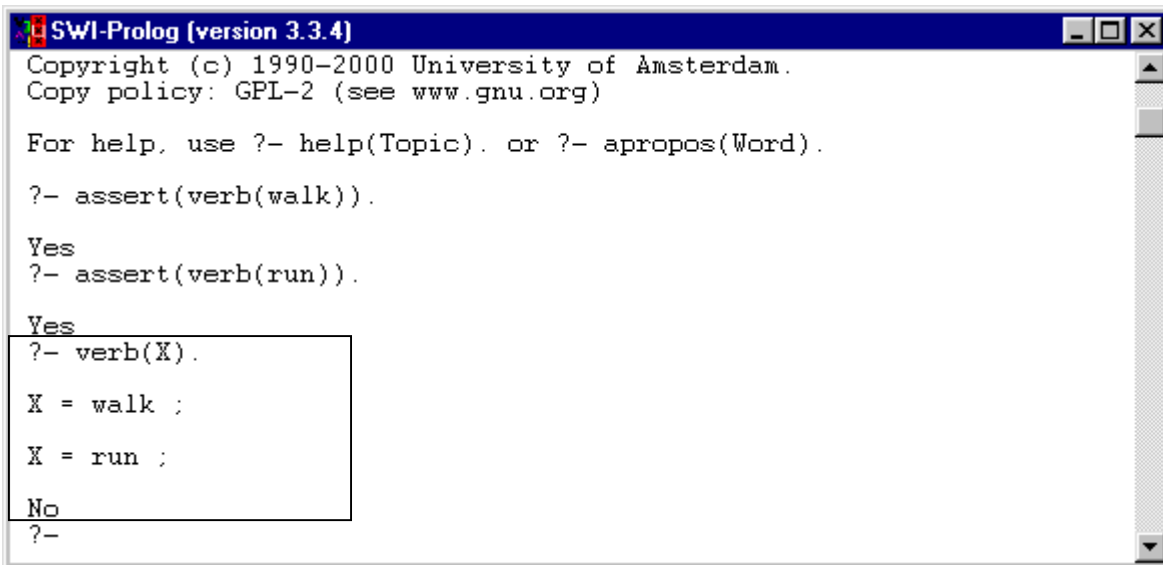
For help, use ?- help(Topic). or ?- apropos(Word).

?- assert(verb(walk)).

Yes
?- verb(X).
X = walk

```

Wird an dieser Stelle das Semikolon eingegeben, so wird nach weiteren Antworten gesucht, bis schließlich keine mehr gefunden werden (NO), der Vorgang ist dann beendet:



```

SWI-Prolog [version 3.3.4]
Copyright (c) 1990-2000 University of Amsterdam.
Copy policy: GPL-2 (see www.gnu.org)

For help, use ?- help(Topic). or ?- apropos(Word).

?- assert(verb(walk)).

Yes
?- assert(verb(run)).

Yes
?- verb(X).

X = walk ;
X = run ;
No
?-

```

Eine direktere Methode, um Prädikate in der Wissensbasis zu überprüfen, ist die Nutzung des eingebauten Prädikates `listing/1`. Das Argument von `listing/1` ist das Prädikat, das überprüft werden soll. Auf unser Beispiel bezogen: die Anfrage `listing(verb/1)` liefert das folgende Ergebnis:



```

SWI-Prolog [version 3.3.4]
X = walk ;
X = run ;

No
?- listing(verb/1).

verb(walk).
verb(run).

Yes
?-

```



Wenn Ihr das Skript am PC durcharbeitet, könnt Ihr jetzt die **Aufgabe 2** am Kapitelende angehen

Auf diese Art und Weise können Klauseln problemlos direkt auf der Oberfläche eingegeben werden. Die Sache hat aber einen gewaltigen Haken: wird Prolog beendet, so verschwinden diese Klauseln wieder aus der Wissensbasis – sie sind flüchtig, denn sie werden nicht dauerhaft gespeichert. Diese Art der Dateneingabe ist exklusiv für solche Fälle gedacht, in denen man 'mal schnell' eine Klausel oder ein Prädikat überprüfen möchte. Für größere Prolog-Programme, die dauerhaften Status haben, verwendet man dagegen die Möglichkeit (B) der Dateneingabe:

Möglichkeit (B): Prolog-Programme im Editor erstellen

Bei dieser Möglichkeit werden Klauseln oder Mengen von Klauseln und Prädikate nicht direkt auf der Prolog-Oberfläche eingegeben, sondern als eigenständige Dateien in einem eigenen Anwenderprogramm, dem Windows-Editor *Notepad* (theoretisch gingen auch Word, Wordpad usw.). Der Editor ist ein kleines Textverarbeitungsprogramm, in welchem dem Benutzer bestimmte Zeichen- und Textmanipulationsmöglichkeiten zur Verfügung stehen (z.B. 'kopieren' und 'einfügen'), die auch bei der Erstellung von Prolog-Programmen nützlich sein können. Die mit dem Editor geschriebenen Prolog-Programme können, wie bei anderen Textverarbeitungsprogrammen auch, unter einem Dateinamen abgespeichert werden.

Zwei Fragen sind nun von Bedeutung:

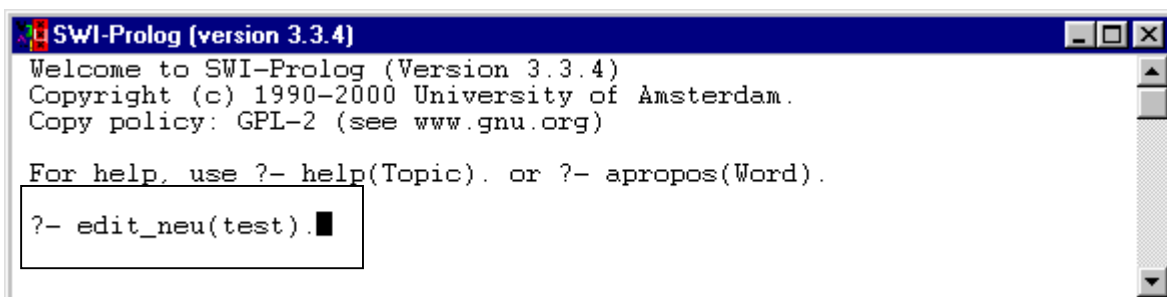
1. Wie startet man von Prolog aus den Editor?
2. Wie kommen die Editor-Daten in die Prolog Wissensbasis?

Zunächst zur ersten Frage. Normalerweise startet man den Editor *Notepad* von der Windows-Oberfläche. Wir aber wollen das Programm von Prolog aus starten, und zwar deshalb, weil auf diese Art gewährleistet wird, daß die Dateien, die wir mit dem Editor schreiben, automatisch in dem Verzeichnis gespeichert werden, auf welches Prolog direkten Zugriff hat.

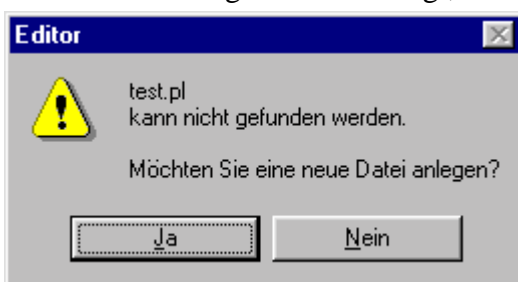
Dabei sind zwei Fälle strikt voneinander zu unterscheiden: nämlich **Fall 1)**: eine Datei wird neu angelegt und **Fall 2)**: eine bereits existierende Datei soll geöffnet werden.

Fall 1): Eine neue Datei anlegen

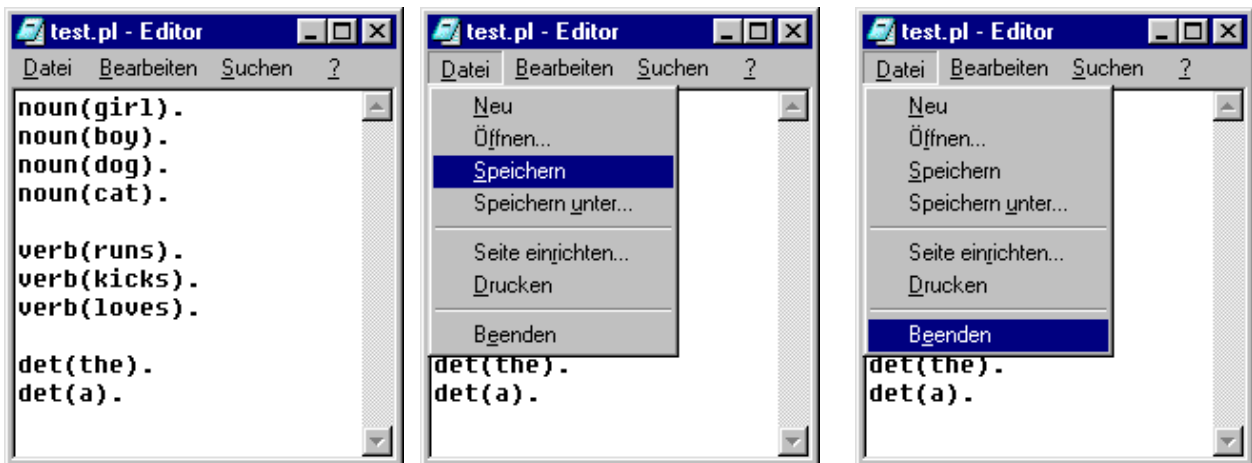
Um eine neue Datei anzulegen, verwenden wir das Prädikat `edit_neu/1`. Das Argument dieses Prädikates ist der Dateiname der neu anzulegenden Datei (den Ihr Euch natürlich vorher überlegen müßt). An einem Beispiel wollen wir den gesamten Vorgang einmal nachvollziehen. Wir legen eine neue Datei namens `test.pl` an. Dazu verwenden wir den Befehl `edit_neu(test)` (wichtig: Kleinbuchstaben!):



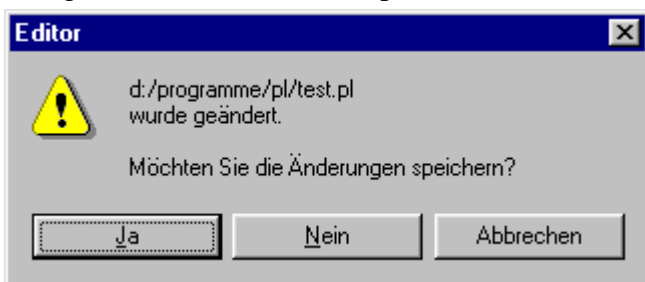
Wird nun die Eingabetaste betätigt, erscheint die folgende Frage:



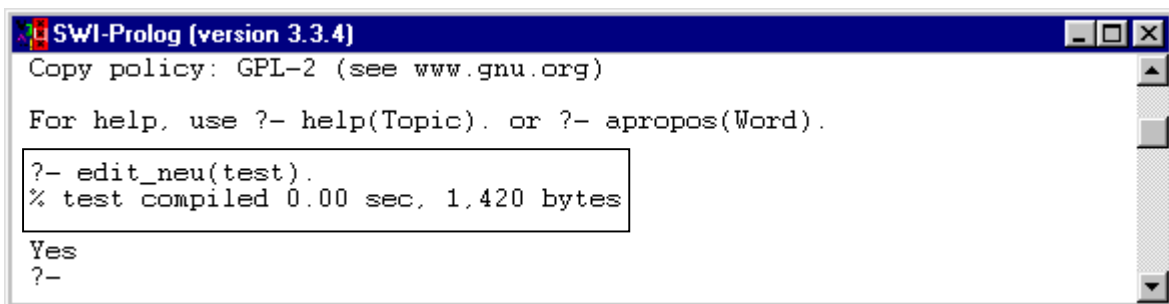
Ihr drückt die Taste `j` (für *ja*), Notepad wird gestartet und Ihr könnt mit dem Schreiben der Datei beginnen. Unsere Beispieldatei soll eine Reihe von Klauseln über die Zugehörigkeit von Wörtern zu lexikalischen Kategorien enthalten, wie im linken der nachfolgenden Screen-Shots dargestellt ist. Wenn Ihr die Datei fertig geschrieben habt, müßt Ihr sie natürlich sichern. Dazu geht Ihr mit der Maus (oder der Tastenkombination `alt+d`) in das Menü *Datei* und wählt den entsprechenden Befehl 'Speichern'. Auf die gleiche Art beendet Ihr Notepad; nur wählt Ihr dazu den Befehl 'Beenden':



Übrigens: wenn Ihr ohne zu speichern den Befehl 'Beenden' wählt, werdet Ihr folgendes gefragt:



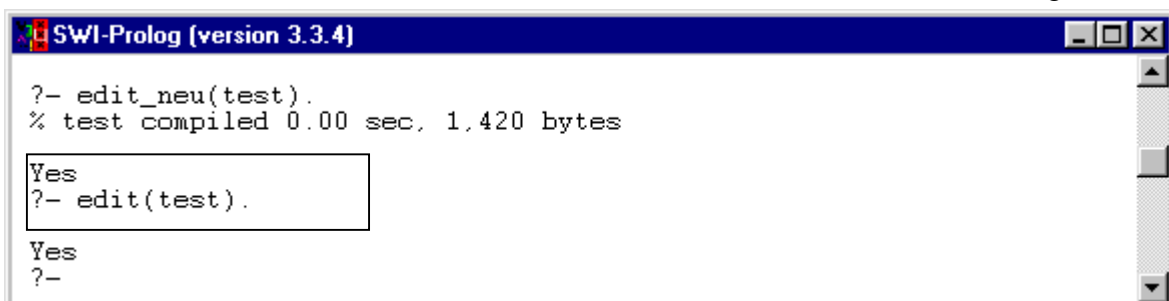
Wenn Ihr hier auf 'Ja' klickt, wird die Datei gespeichert und Ihr kehrt automatisch auf die Prolog-Oberfläche zurück. Dort erscheint nun die folgende Meldung:



Die Zeile `% test compiled...` besagt, dass Datei kompiliert und ihre Inhalte also in die Wissensbasis geschrieben sind.

Fall (2): eine bereits existierende Prolog Datei im Editor bearbeiten.

Dieses ist der einfache Fall: hier benutzen wir `edit/1`; mit dem Dateinamen als Argument:



Daraufhin wird die entsprechende Datei unverzüglich im Editor geöffnet und kann bearbeitet werden. Speichern der Datei und Beenden des Editors verläuft genauso, wie oben angegeben.

Wie wir gesehen haben, liest Prolog die im Notepad **neu** erstellten Dateien automatisch ein. Wird aber eine bereits existierende Datei bearbeitet, läuft das anders, insofern das Einlesen – nach dem ersten Aufruf – hier vom Benutzer ausgelöst werden muss.

Auch hierfür verwenden wir ein eingebautes Prolog-Prädikat, nämlich `consult/1`. Das Argument von `consult/1` ist der Name der Datei, deren Inhalt in die Wissensbasis eingelesen werden soll

```

SWI-Prolog (version 3.3.4)
Copy policy: GPL-2 (see www.gnu.org)

For help, use ?- help(Topic). or ?- apropos(Word).

?- edit(test).

Yes
?- consult(test).
% test compiled 0.00 sec, 560 bytes

Yes
?-

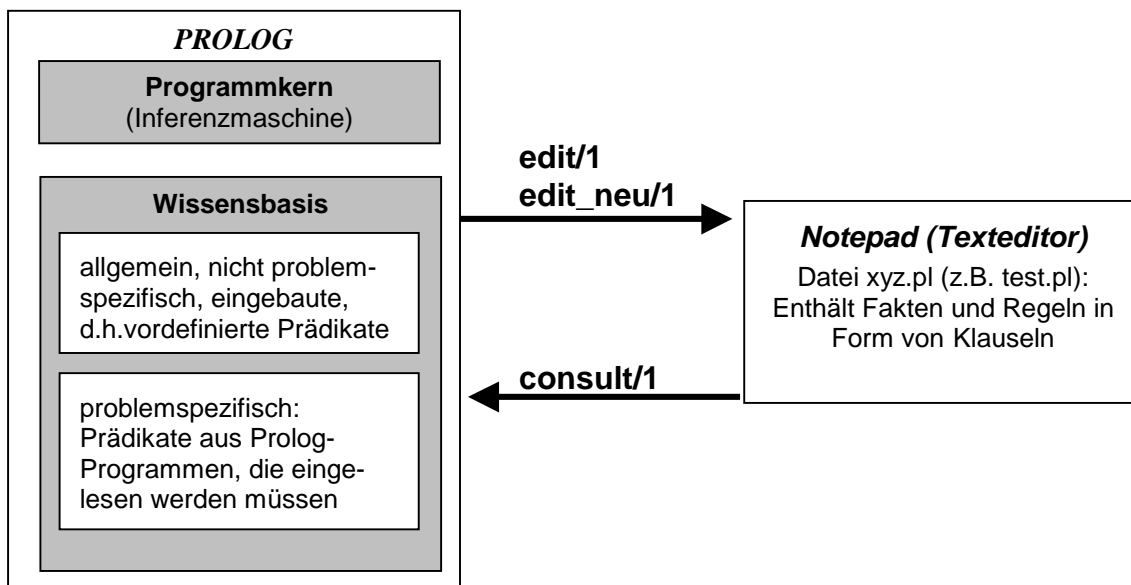
```

Wir erhalten die Meldung, daß die Datei `test` kompiliert wurde (außerdem Angaben über die Dateigröße bzw. die Dauer der Kompilierung). Jetzt sind die Klauseln aus der Datei `test.pl` in der Prolog-Wissensbasis, was durch `listing/1`, z.B. `listing(noun/1)`, überprüft werden kann:



Wenn Ihr das Skript am PC durcharbeitet, könnt Ihr jetzt die **Aufgabe 3** am Kapitelende angehen

Der Datenaustausch zwischen Prolog und dem Editor kann abschließend wie folgt graphisch dargestellt werden, detaillierte Erläuterungen folgen in späteren Kapiteln:



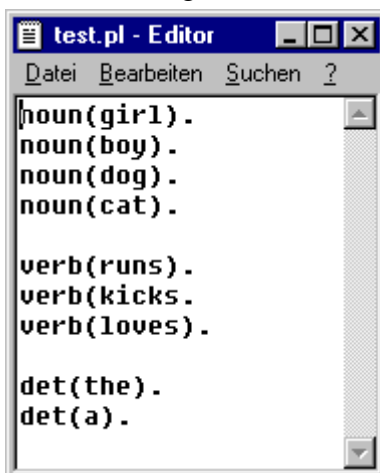
Zusammenfassung:

- Um eine **neue** Prolog-Programmdatei anzulegen, rufen wir mithilfe von `edit_neu/1` den Editor auf. Das Argument von `edit_neu/1` ist der Dateiname der neuen Datei. Beispiel: `edit_neu(test)`.
- Um eine **bereits existierende** Datei zu bearbeiten, wird `edit/1` verwendet. Das Argument ist der Dateiname. Beispiel: `edit(test)`.
- Um die in Dateien kodierte Information in die Wissensbasis einzulesen, wird `consult/1` verwendet. Das Argument von der Dateiname. Beispiel: `consult(test)`. Dieses ist aber nur in dem Fall erforderlich, in dem eine bereits existierende Datei aufgerufen wird, und auch da nur beim ersten Mal. (klingt ja grauhaft – Ihr werdet in der Praxis aber sofort kapieren, was gemeint ist)
- Um zu überprüfen, ob ein Prädikat auch tatsächlich in der Wissensbasis enthalten ist, verwenden wir das eingebaute `listing/1`, dessen Argument das zu überprüfende Prädikat ist.

Tips & Tricks

SWI-Prolog ist kein kommerzielles Windows-Anwenderprogramm – nicht zuletzt deshalb weist es eine gewisse Benutzerunfreundlichkeit auf. Es gibt keine bequemen Menüs, und, wie gesehen, muß bei dem Erstellen einer neuen Datei mit dem Editor höllisch auf Kleinigkeiten geachtet werden, damit das Ganze funktioniert. Die nachfolgenden Punkte enthalten einige Tips, deren Beachtung das reibungslose Arbeiten mit SWI-Prolog fördern.

- Immer alle Prolog-Klauseln mit einem Punkt abschließen
- Prädikatsnamen stets klein schreiben.
- Keine Leerzeichen einsetzen.
- Keine Umlaute, 'ß' oder Sonderzeichen verwenden
- Bei allen mit dem Editor erstellten Dateien nach der letzten Befehlszeile in eine neue Zeile gehen, also die Eingabetaste betätigen!! (Dieser Punkt ist ganz besonders blöd, aber immens wichtig!!)
- Den Editor stets von Prolog aus aufrufen, andernfalls erhalten die damit erstellten Dateien die Kennung .txt und werden per Default in einem Verzeichnis gespeichert, auf welches Prolog keinen Zugriff hat.
- Auf die Fehlermeldungen von Prolog achten. Wenn Dateien kompiliert werden, findet eine kleine Syntaxprüfung statt; falls Prolog Fehler findet, wird dieses gemeldet. Beispiel: In diesem Programm steckt ein Fehler:



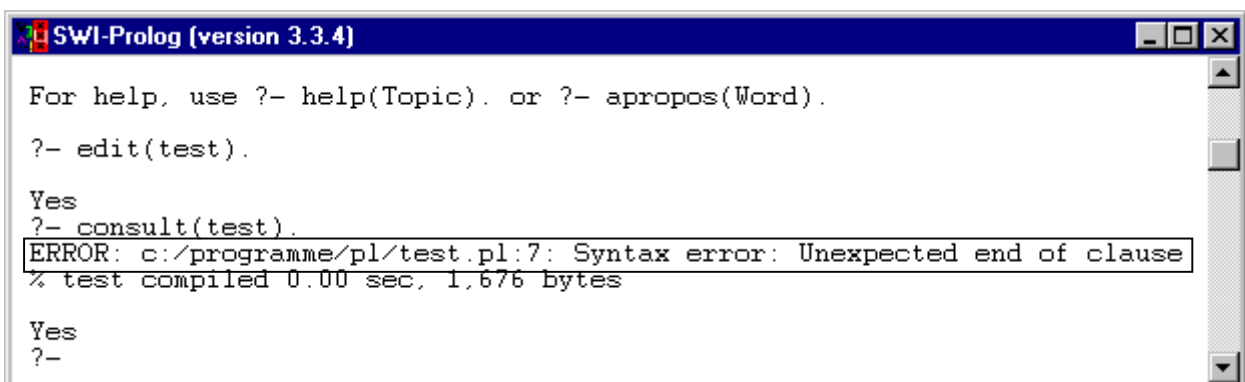
```

test.pl - Editor
Datei Bearbeiten Suchen ?
noun(girl) .
noun(boy) .
noun(dog) .
noun(cat) .

verb(runs) .
verb(kicks .
verb(loves) .

det(the) .
det(a) .
  
```

Wenn diese Datei konsultiert wird, kommt die folgende Meldung:



```

SWI-Prolog (version 3.3.4)
For help, use ?- help(Topic). or ?- apropos(Word).
?- edit(test).
Yes
?- consult(test).
ERROR: c:/programme/pl/test.pl:7: Syntax error: Unexpected end of clause
% test compiled 0.00 sec, 1,676 bytes
Yes
?-
  
```

Hier wird angegeben, dass in Zeile 7 ('pl:7') ein Syntax-Fehler vorliegt – und genau das ist der Fall: dort fehlt nämlich eine Klammer.

Erste Schritte in SWI-Prolog: Übungen

Aufgabe 1:

Gebt mithilfe des Prädikats `assert/1` die folgenden Klauseln ein. Achtet darauf, dass das Argument von `assert/1` in Klammern aufgeführt wird. Achtet weiter darauf, dass ihr Kleinbuchstaben verwendet und jede Eingabe durch einen Punkt abschließt.

```
verb(walk).  
verb(run).  
verb(kick).  
parent(john,mary).  
parent(john,bill).
```

Aufgabe 2:

- Gebt mithilfe des Prädikats `assert/1` die folgenden Klauseln ein.

```
verb(love).  
verb(eat).  
parent(jane,mary).  
parent(jane,bill).
```
- Stellt mithilfe von Variablen (Großbuchstaben) Anfragen an Prolog, die das Prädikat `verb/1` überprüfen.
- Überprüft mit `listing/1` das Prädikat `parent/2`

Aufgabe 3

- Legt mit `edit_neu/1` im Editor eine Datei names `test.pl` an und tragt die folgenden Klauseln über die Zugehörigkeit von Wörtern zu lexikalischen Kategorien in diese Datei ein:

```
noun(book).  
noun(boy).  
noun(girl).  
adj(stupid).  
adj(boring).  
det(the).  
det(a).
```

Speichert diese Datei und beendet den Editor. Überprüft anschließend mit `listing/1` die Prädikate `noun/1`, `adj/1` und `det/1`
- Ruft mit `edit/1` die Datei `test.pl` erneut auf und ergänzt diese um die folgenden Klauseln:

```
adv(very).  
adv(rather).  
prep(in).  
prep(to).  
prep(with).
```

Speichert die Datei und beendet den Editor. Überprüft anschließend mit `listing/1` die Prädikate `adv/1` und `prep/1`