

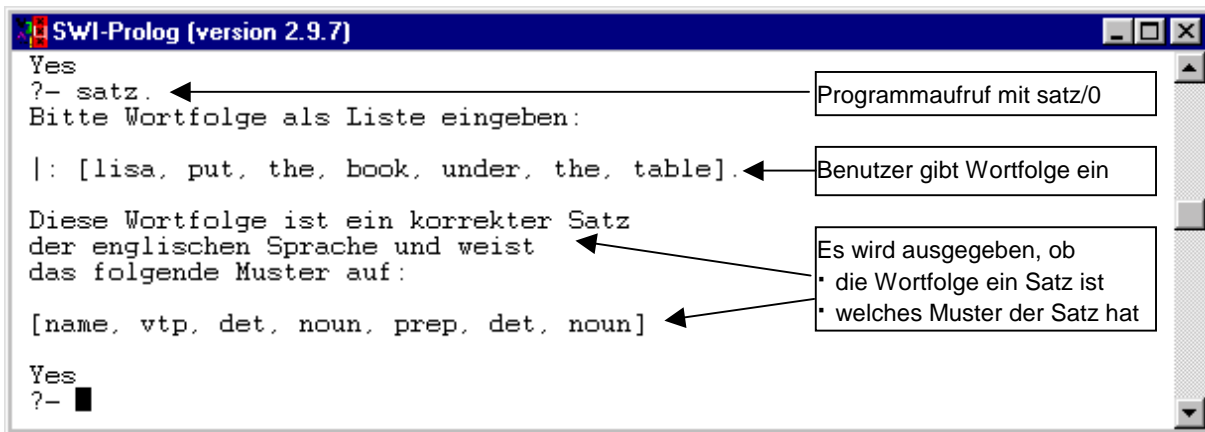
Kapitel 6.

Satzanalyse mit Prolog

In diesem Kapitel geht es um die Verarbeitung von natürlichsprachigen Sätzen mit Prolog. Wir wollen ein Programm entwickeln, welches in der Lage ist, für eine beliebige englische Wortfolge anzugeben, ob diese ein Satz der englischen Sprache ist oder nicht, und wenn ja, diesem Satz eine oder (bei struktureller Ambiguität) mehrere Strukturbeschreibungen zuordnet. Dieses Programm wird in kleinen Schritten aufgebaut, so daß die endgültige Form das Resultat von jeweils nachvollziehbaren Modifikationen ist. Auf diese Weise soll erreicht werden, daß am Ende solch einer sukzessiven Programmentwicklung nicht nur ein lauffähiges Programm steht, sondern auch ein Grundverständnis für die spezifischen Probleme und Methoden geschaffen wird, die für diesen Bereich der Computerlinguistik relevant sind.

6.1. Teil 1: Akzeptor I: Satzanalyse mit Mustervergleich (*Pattern-Matching*)

In diesem Abschnitt geht es darum, eine Reihe von Satz-Strukturmustern in die Prolog-Datenbank zu schreiben, mit denen eine beliebige Wortkette 'abgeglichen' wird. Die erste Version ist ein Programm, welches folgendes zu leisten vermag:



Das Programm wird mit `satz/0` aufgerufen. Die Benutzer können daraufhin eine Wortkette englischer Wörter in Form einer Liste eingeben. Wenn diese Liste mit den Datenbanksatzstrukturmusterlisten (Hottentottenstottertrottelmutter) übereinstimmt, wird die Wortfolge als englischer Satz akzeptiert, und darüberhinaus wird das Satzstrukturmuster ausgegeben. Einen so gearteten Parser nennt man ERKENNER oder AKZEPTOR.

6.1.1. Grundlagen

Bevor wir diese Problemstellung in einem Programm umsetzen, wollen wir die Sache zunächst unabhängig von Prolog angehen. Letztendlich geht es um nichts anderes, als eine (Mini-)Grammatik zu formulieren. 'Grammatik' soll hier im engen Sinne verstanden werden als eine Theorie über eine Einzelsprache, welche festlegt, welche Ketten von Wörtern dieser Sprache grammatische Ketten (z.B. Sätze) dieser Sprache sind.²⁵ Eine wichtige Eigenschaft von Sätzen ist natürlich die lineare Anordnung der einzelnen Wörter – *Milben leben in Kopfkissen ist o.k., *in leben Milben Kopfkissen* dagegen nicht. Wenn man von konkreten Instanzen der lexikalischen Kategorien abstrahiert und also sich nicht auf konkrete Wörter, sondern auf Wortarten bezieht, funktioniert eine einfache Grammatik schlichtweg so, daß eine Reihe von Anordnungsmustern für lexikalische Kategorien vorgegeben wird. Eine Wortkette muß dann also in eine entsprechende Anordnung lexikalischer Kategorien übersetzt werden, und diese muß mit den Mustern verglichen werden. Nur solche Anordnungen, die in den Mustern erfaßt sind, werden als korrekte Sätze identifiziert.

²⁵ Ein wohlbekanntes Beispiel für eine solche Art von Grammatik ist z.B. die in der Syntaxeinführung vorgestellte Phrasenstrukturgrammatik.

Betrachten wir dazu eine konkrete kleine Grammatik, die völlig frei ist von linguistischen Feinheiten und also entsprechend leicht zu falsifizieren, da hier zunächst einmal das Prinzip deutlich werden soll. Es gelten die folgenden Abkürzungen (an die wir uns am besten gleich hier gewöhnen, da wir sie auch in dem Programm benutzen wollen):

<i>Wortart</i>	<i>Bezeichnung</i>	<i>Beispiel</i>
Determinatoren	det	<i>the</i>
Eigennamen	name	<i>Bart</i>
Nomen	noun	<i>dog</i>
intransitives Verb	vi	<i>slept</i>
transitives Verb	vt	<i>kissed</i>
bitransitives Verb mit PP-Ergänzung	vtp	<i>put</i>
Präposition	prep	<i>on</i>
Adjektiv	adj	<i>ugly</i>
Adverb	adv	<i>very</i>

'Wortart' soll hier im Sinne von Lexemklasse oder lexikalischer Kategorie verstanden werden. Angaben darüber, über welche Elemente eine bestimmte Lexemklasse realisiert werden kann, finden sich im Lexikon. Wir gehen zunächst von dem folgenden kleinen Lexikon aus:

det → {the, a, these}
 name → {bart, lisa, john, mary}
 noun → {girl, girls, boy, book, books, dog, table, garden}
 vi → {slept, cried, died}
 vt → {loved, kicked, kissed, painted}
 vtp → {gave, put, sold }
 prep → {to, in, on, under, into}
 adj → {ugly, beautiful}
 adv → {very, terribly}

Als potentielle Satzstrukturmuster für das Englische kommen z.B. die folgenden in Frage:

- 1) [det, noun, vi]
- 2) [name, vi]
- 3) [det, noun, vt, det, noun]
- 4) [det, noun, vt, name]
- 5) [name, vtp, det, noun, prep, det, noun]
- 6) [det, noun, vtp, det, noun, prep, name]
- 7) [det, adv, adj, noun, vt, det, noun]

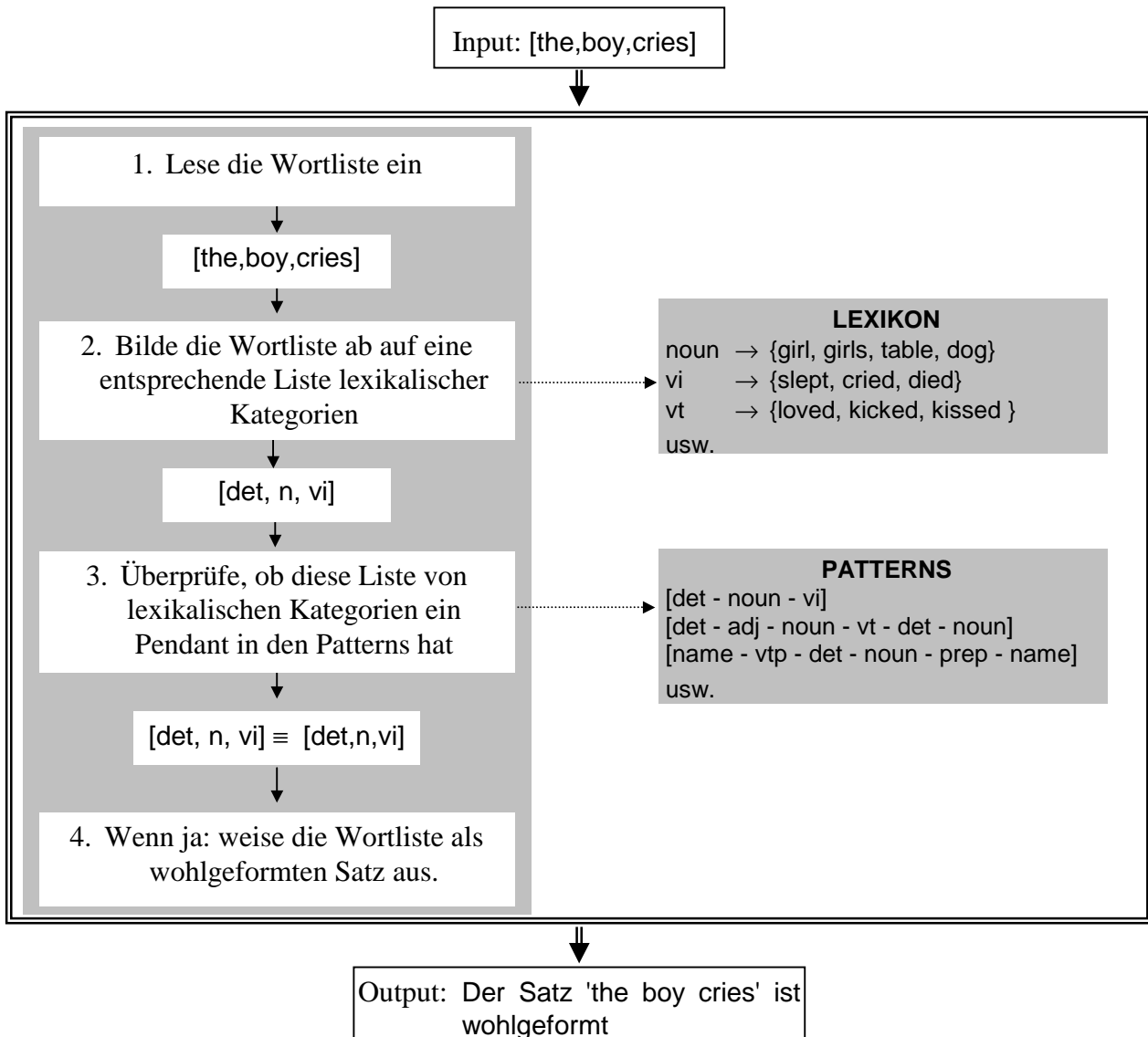
Über diese einfache Grammatik können nun unter anderem die folgenden Wortketten als grammatische Folgen, sprich Sätze des Englischen abgeleitet werden:

Lisa slept, John cried (Muster 2), *The girl loved the dog, A boy kicked the table, These girls kissed Bart* (Muster 3), *Mary gave the dog to the girls, Bart put the book under the table* (Muster 5), *The girls sold the books to John* (Muster 6) *a terribly ugly boy kicked the dog* (Muster 7) und so weiter. Natürlich würde zunächst auch die Kette **these boy died* als korrekt erkannt – dazu später noch mehr.

Die Umsetzung dieser Grammatik in Prolog ist einfach. Der Mini-Pattern-Matcher soll zunächst die folgenden Komponenten umfassen:

- Angaben darüber, welcher Wortart ein Wort angehört (Lexikon)
- Angaben darüber, welche Satzmuster wohlgeformt sind (Patterns)
- Eine Funktion, die eine Eingabeliste von Wörtern auf eine entsprechende Liste lexikalischer Kategorien abbildet (Bezug: das Lexikon)
- Ein Funktion, die eine Liste lexikalischer Kategorien mit den (in Listenform notierten) Patterns vergleicht. (Bezug: die Patterns)

Ausgehend von der Überlegung, dass die Benutzer die zu analysierende Wortkette in Form einer Liste eingeben, kann Programmaufbau und -ablauf wie folgt charakterisiert werden:



6.1.2. Die Realisierung des Mini-Pattern-Matchers in Prolog

Was die Umsetzung betrifft, sollen zuerst einmal die folgenden Maßgaben gelten:

- das Lexikon soll in einer eigenen Datei namens **lexikon.pl** erstellt werden.
- die Patterns sollen in einer eigenen Datei namens **patterns.pl** erstellt werden.
- der 'Steuermechanismus', sprich diejenigen Komponenten, die in der Graphik in den Punkten 1.-4. notiert sind, soll in einer eigenen Datei namens **grammatik1.pl** erstellt werden.

Bei diesem kleinen Programm mag man sich fragen, wieso die Komponenten nicht zusammen in einer einzigen Datei erfaßt werden. Die Antwort darauf lautet, dass wir beispielsweise das Lexikon auch für weitere Satzanalyseprogramme verwenden werden, und es sich also nicht nur wegen der besseren Transparenz des Programmes anbietet, die einzelnen Komponenten voneinander zu trennen, sondern dieses auch ganz praktische Gründe hat.

6.1.2.1. Das Lexikon

Die Zugehörigkeit einzelner Wörter zu bestimmten Lexemklassen wird in Form von lex/2-Fakten notiert:

lex(Wort,Kat), also z.B.

lex(the,det).

lex(boy,noun).

usw.

6.1.2.2. Die Patterns

Die Satzmuster werden in Form von muster/1-Fakten notiert; das Argument von muster/1 ist eine Liste: muster(Liste), also z.B.

muster([det,n,vi]).

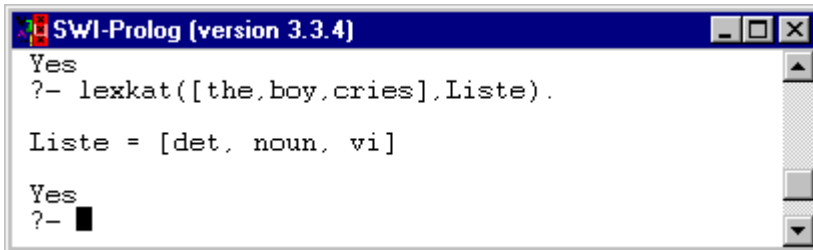
muster([name,vt,det,noun]).

usw.

6.1.2.3. Der Pattern-Matcher

Für die Steuerdatei benötigen wir zunächst das Hilfsprädikat lexkat/2.

lexkat/2 (lexkat(Liste1,Liste2)) bildet eine Liste von Wörtern ab auf eine entsprechende Liste lexikalischer Kategorien:



```

SWI-Prolog [version 3.3.4]
Yes
?- lexkat([the,boy,cries],Liste).

Liste = [det, noun, vi]

Yes
?-

```

Mithilfe von lexkat/2 kann der Pattern-Matcher über ein Prädikat satz/0 wie folgt realisiert werden:

1. fordere den Benutzer auf, eine Wortkette in Form einer Liste einzugeben (write...)
2. lese die Liste ein (read...)
3. bilde die Liste ab auf eine Liste lexikalischer Kategorien (lexkat...)
4. überprüfe, ob diese Liste einem Satzmuster entspricht (muster...)
5. wenn ja, weise den Satz als wohlgeformt aus und zeige das Satzmuster an. (write...)



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

Damit hätten wir ein kleines Programm für einen Satzakzeptor erstellt. Wie aber sehr leicht nachzuweisen ist, ist dieses Programm wirklich nur eine erste Annäherung an komplexere Programme. Die Mängel sind unterschiedlicher Natur; können wie folgt beschrieben und sollen der Reihe nach revidiert werden:

- Das Programm erkennt auch ungrammatische und anomale Formen als korrekt - beispielsweise die Ketten **a girls kissed these boy* oder *?the book cried*.
- Das Programm rekuriert – wie alle Grammatikprogramme – stark auf das Lexikon. Falls nun in einem zu analysierenden Satz ein Wort auftritt, welches nicht im Lexikon erfaßt ist, kann die Analyse nicht vollzogen werden. Wieviel angenehmer wäre es doch, wenn in einem solchen Fall durch eine entsprechende Benutzerschnittstelle das Programm in der Lage wäre, das Lexikon um genau das fehlende Wort zu ergänzen, und die Analyse des Satzes doch durchgeführt würde.
- Die Satzstrukturmuster sind allesamt 'per Hand' eingegeben, nicht aber aus einer vernünftigeren Grammatik abgeleitet. Dieser Punkt ist ein ganz zentrales Manko des Programmes. Was in den Mustern erfaßt wird, ist die reine lineare Anordnung der lexikalischen Kategorien. Dass Sätze aber hierarchische Strukturen sind, kann in der Grammatik, die überhaupt keinen Bezug nimmt zu syntaktischen Kategorien (also NP, VP etc.) überhaupt nicht zum Ausdruck gebracht werden. Damit würde beispielsweise die Mehrdeutigkeit des Satzes *Lisa watched the boy with the telescope* über die Zuordnung zweier unterschiedlicher Strukturbeschreibungen nicht erfaßt, ebensowenig wie die Rekursivität von Sprache(n) beispielsweise was die Verschachtelungstiefe von Konstruktionen angeht. Um diesen Punkt zu ändern, müssen wir einen völlig anderen Aufbau der Grammatik verwenden.

6.1.3. Die Erweiterung der Grammatik um sekundäre grammatische Merkmale²⁶

Das Problem bei der Sache ist, dass weder im Lexikon noch in den Satzmustern berücksichtigt ist, ob die kombinierten Elemente bezüglich solcher Merkmale wie Numerus, Person oder Kasus kongruieren. Das heißt, dass Kombinationen wie

**these book* (Problem bzgl. Numerus)

**I kicks you* (Problem bzgl. Person)

**She loves he* (Problem bzgl. Kasus)

allesamt als o.k. angezeigt würden, ohne es zu sein. Wie kann man diesen Zustand revidieren? Nun, wir müssen sowohl das Lexikon als auch die Satzmuster modifizieren dahingehend, dass in ihnen den jeweils vorliegenden Merkmalen bzgl. Numerus, Person usw. Rechnung getragen wird.

Bevor wir uns der Umsetzung in Prolog widmen, müssen wir aber natürlich Klarheit haben, für welche Lexemklasse überhaupt welche Merkmale für unsere Grammatik relevant sind. Informell sähe das – für das Englische! – z.B. so aus:

DETERMINATOREN: Numerus, Definitheit

NOMINA: Numerus, Person, Kasus

PRONOMINA: Numerus, Person, Kasus, Genus

VERBEN: Numerus, Person, Tempus, Modus

PRÄPOSITIONEN, ADJEKTIVE und ADVERBIEN: Keine.

Wir wollen uns für den Einstieg allerdings exklusiv auf das Numerusmerkmal konzentrieren.

Die Erweiterung des Lexikons um ein solches Numerusmerkmal kann **exemplarisch** wie folgt dargestellt werden:

<i>Wort</i>	<i>Lexemklasse</i>	<i>Merkmal</i>
<i>a</i>	det	singular
<i>these</i>	det	plural
<i>the</i>	det	singular/plural
<i>boy</i>	noun	singular
<i>boys</i>	noun	singular
<i>girl</i>	noun	singular
<i>girls</i>	nomen	plural
<i>kisses</i>	vt	singular
<i>kissed</i>	vt	singular/plural
<i>in</i>	prep	—
<i>ugly</i>	adj	—
<i>very</i>	adv	—

Was die Umsetzung dieser Information im Pattern-Matcher angeht, stellen sich (mindestens) zwei Fragen:

1. Was macht man in den Fällen, in denen ein Wort bezüglich des Merkmals nicht spezifiziert ist, beispielsweise bei *the* und *kissed*?
2. Wie kann man dieses Merkmal überhaupt in das Prolog-Lexikon integrieren?

Die erste Frage ist leicht zu beantworten – in solchen, nicht-festgelegten Fällen verwenden wir einfach die anonyme Variable, sprich den Unterstrich '_'.

²⁶ Anomale Konstruktionen (*?the bed kicked the garden*) werden nicht berücksichtigt. Prinzipiell könnte deren Behandlung aber nach demselben Prinzip ablaufen, wie es in diesem Abschnitt für die ungrammatischen Konstruktionen vorgestellt wird - sprich über Merkmale wie z.B. [±BELEBT]

Bezüglich Frage 2 könnte man auf die Idee kommen, die *lex/2*-Fakten einfach um das relevante Merkmale zu erweitern, d.h. dass wir das Merkmal einfach als zusätzliches Argument mitführen, beispielsweise so:

lex(a, det, sg).

lex(these, det, pl).

lex(the, det, _).

lex(boy, noun, sg).

lex(boys, noun, pl).

lex(kisses, vt, sg).

lex(kissed, vt, _).

lex(in, prep).

lex(ugly, adj).

So aber funktioniert das nicht. Warum? Die Merkmale eines jeden Eintrages sind zwar auf diese Art im Lexikon erfaßt aber – und das ist das Problem – wir haben nun kein *lex/2*-Prädikat mehr: wir hätten nämlich für die Determinatoren, Nomina und Verben ein *lex/3*-Prädikat, für Präpositionen, Adjektive und Adverbien dagegen ein *lex/2*-Prädikat. Diese Situation verschärft sich natürlich dann, wenn es nicht mehr ausschließlich um Numerus, sondern auch noch um andere sekundäre grammatische Kategorien geht.

Wenn wir so vorgehen, würden nämlich weder die Abbildung einer Wortliste auf eine Liste der lexikalischen Kategorien noch der Mustervergleich mehr funktionieren - ganz einfach deshalb, weil wir hierfür Lexikoneinträge brauchen, die **alle** dieselbe Stelligkeit haben. Angesichts der Tatsache, dass die Anzahl der relevanten Merkmale aber für die unterschiedlichen Lexemklassen variiert, scheint das nicht zu funktionieren – es sei denn, dass man die Merkmale nicht als 'eigenständige' Argumente des *lex*-Funktors notiert, sondern sie stattdessen als Argumente der Kategorienbezeichnung aufführt, denn auf diese Art würde sich an der Stelligkeit von *lex/2* nichts ändern. Abstrahiert kann diese Vorgehensweise so notiert werden: *LEX(WORT, KAT(MERKMAL₁, MERKMAL₂, ..., MERKMAL_N))*. Für die einzelnen Kategorien heißt das:

det(Numerus)

noun(Numerus)

verb(Numerus)

prep

adj

adv

Konkret würden die oa. Lexikoneinträge also die folgende Form haben:

lex(a, det(sg)).

lex(these, det(pl)).

lex(boy, noun(sg)).

lex(boys, noun(pl)).

lex(kisses, vt(sg)).

lex(kissed, vt(_)).

lex(in, prep).

lex(ugly, adj).

Man sieht, dass auf diese Weise alle Lexikoneinträge die Stelligkeit 2 haben. Die einzelnen Kategorien, die ja nun in Form von Funktor-Argument-Strukturen notiert sind, variieren dagegen in ihrer Stelligkeit - *det/1*, *noun/1* und *verb/1* gegenüber *prep/0* und *adj/0*. Soviel zunächst zum Lexikon.

Natürlich müssen aber auch die Satzmuster modifiziert werden, denn hier müssen ja nun Angaben gemacht werden, aus denen das Programm entnehmen kann, dass beispielsweise Determinator und Nomen kongruieren müssen mit Bezug auf das Numerusmerkmal.

Diese Modifikation ist aber einfach. Erst einmal muss berücksichtigt werden, dass die Form der Liste lexikalischer Kategorien ja jetzt anders aussieht. Wenn eine Eingabewortliste auf eine Ausgabeliste lexikalischer Kategorien abgebildet würde, hätte das Ergebnis die folgende Form:

Eingabeliste: [a,boy,sleeps].

Ausgabeliste: [det(sg), noun(sg), vi(sg)].

Hieran sieht man vielleicht schon, wie der Hase laufen muss. Offensichtlich ist es so, dass das einzige Argument von `det/1` übereinstimmen muss mit dem einzigen Argument von `noun/1`. Das einzige Argument von `noun/1` muss übereinstimmen mit dem einzigen Argument von `vi/2`.²⁷

Betrachten wir auf diesem Hintergrund das folgende althergebrachte Pattern:

`muster([det, noun, vi]).`

Dieses kann nun einfach modifiziert werden, indem den Kategorienbezeichnungen die entsprechenden Argumente hinzugefügt werden. In einem Prolog-Muster müssten identische Variablen für den kongruierenden Wert verwendet werden:

`muster([det(N), noun(N), vi(N)]).`

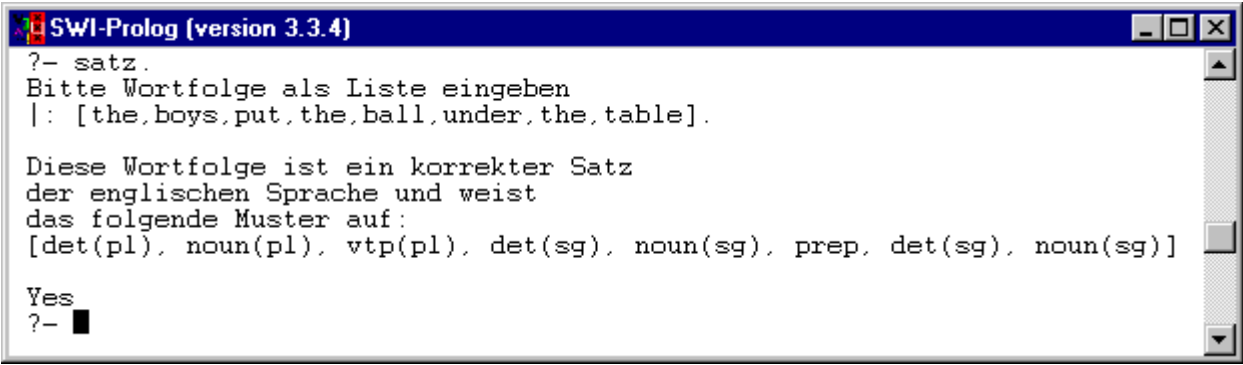
Wenn ein Verb ein Objekt hat, muss dieses bzgl. Numerus natürlich nicht mit dem Verb kongruieren:

*The boy reads **the book** / The boy reads **the books**.*

Ergo muss für den Numeruswert eines Objekts-Nomens (und dessen Determinator) in einer Konstruktion mit einem transitiven Verb eine **andere** Variable verwendet werden:

`muster([det(N), noun(N), vt(N), det(N1), noun(N1)]).`

Wenn diese Modifikationen durchgeführt werden, würde der Pattern-Matcher nunmehr so aussehen:



```

SWI-Prolog [version 3.3.4]
?- satz.
Bitte Wortfolge als Liste eingeben
| : [the,boys,put,the,ball,under,the,table].

Diese Wortfolge ist ein korrekter Satz
der englischen Sprache und weist
das folgende Muster auf:
[det(pl), noun(pl), vtp(pl), det(sg), noun(sg), prep, det(sg), noun(sg)]

Yes
?- █

```

Interessant an dieser Stelle ist die Tatsache, dass für einen Det wie *the*, der im Lexikon bezüglich des Numerusmerkmals un spezifiziert ist, hier doch konkrete Werte (im vorliegenden Fall PLURAL für *the boys*; SINGULAR für *the ball* und *the table*) ausgegeben wird. In der Veranstaltung werden wir im Zusammenhang mit Unifikationsprozessen noch darüber unterhalten.



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

6.1.4. Die Modifikation des Lexikons während der Laufzeit des Programmes

Das zweite der weiter oben angesprochenen Probleme ist die Tatsache, dass natürlich nur diejenigen Wortketten behandelt werden können, für deren Elemente sich entsprechende Einträge im Lexikon finden. Falls der Benutzer eine Kette wie z.B. *the cat purrs* eingibt, die ja ohne Zweifel ok ist, von der aber nicht alle Elemente im Lexikon auftreten, würde auch hier das befürchtete 'No' als Analyseergebnis erscheinen.

Was wir benötigen, ist ein Prädikat, welches ein unbekanntes Wort in der Wort-Eingabeliste erkennt, den Benutzer auffordert, die Kategorie und ggf. das Numerusmerkmal dieses Wortes einzugeben um es dann in der Wissensbasis zur Verfügung zu haben. Es gäbe auch die Möglichkeit, diese neue Information

²⁷ Auf dem 'einzigsten' wird hier ein bißchen rumgeritten deshalb, weil in einer Erweiterung um Person, Tempus etc. natürlich mehr als nur ein Argument in der Struktur auftreten wird.

tatsächlich in die Datei `lexikon.pl` einschreiben zu lassen – angesichts der Tatsache aber, dass viele Benutzer häufig nur eine etwas vage Vorstellung über die Zugehörigkeit eines Elementes zu einer Lexemklasse haben (jajohl: auch Linguistikstudierende), riskieren wir diese Option lieber nicht.

Es war eigentlich so geplant, dass wir dieses Prädikat (passenderweise `nachschlagen/2` genannt) gemeinsam definieren, da es relativ leicht zu durchschauen ist und man dabei etwas über Programmablauf und Datenbankprädikate lernt. Durch die Integration des Numerusmerkmals für bestimmte Kategorien aber hat sich dieses Prädikat verkompliziert, also wird es hier einfach in seinen Grundzügen beschrieben, anschließend komplett in Prolog wiedergegeben und kann dann in das Lexikon eingebaut werden.

Wird `nachschlagen(Wort,Katmerk)` mit einem `Wort` aufgerufen, welches **nicht** im Lexikon steht, so

1. informiere den Benutzer über diese Tatsache und bitte ihn, `Kat` einzugeben,
2. lese `Kat`
3. Falls `Kat` *Nomen*, *Verb* oder *Det* ist:
Erfrage das Numerusmerkmal `Num` mit `hole_merkmal/2`.
Erzeuge aus der Kategorie und dem Numerusmerkmal einen Eintrag `Katmerk` in Funktor-Argumentstruktur
Falls die Kategorie *Präposition*, *Adjektiv* oder *Adverb* ist:
Setze `Katmerk` mit `Kat` gleich
4. Setze die Variable `Eintrag` mit einem Fakt `lex(Wort, Katmerk)` gleich,
5. Frage, ob `Eintrag` in die Datenbank übernommen werden soll (ja oder nein),
6. wenn die Antwort 'ja' lautet: 'asserte' `Eintrag`,
7. wenn die Antwort 'nein' lautet, dann `true` (der Grund dafür wird später klarer (vielleicht)).

`hole_merkmal(_Kat,Merkmal):-`

```
write('Bitte das Numerusmerkmal
eingeben (sg/pl/ _)'),
nl,
read(Merkmal).
```

`nachschlagen(Wort,Kat):-`

```
lex(Wort,Kat),!
```

`nachschlagen(Wort,Katmerk):-`

```
write('Das Wort '),
write(Wort),
write(' steht nicht im Lexikon'),
nl,
write('Bitte die Kategorie angeben: '),
nl,
read(Kat),
(
(member(Kat,[noun,verb,det]),
hole_merkmal(Kat,Merkmal),
Katmerk =.. [Kat,Merkmal],!)
;
Katmerk = Kat
),
Eintrag = lex(Wort,Katmerk),
write('Soll der folgende Eintrag
in die Datenbank übernommen werden? (ja/nein)'),
nl,
write(Eintrag),
nl,
read(Antwort),
nl,
(Antwort = ja, assert(Eintrag),!);
Antwort = nein, true, !).
```

Wenn nun die rekursive Regel des Prädikates `lexkat/2` entsprechend modifiziert wird:

`lexkat([Wort|Wortliste],[Kat|Katliste):-`

```
nachschlagen(Wort,Kat),
lexkat(Wortliste,Katliste).
```

werden während der Laufzeit des Programmes neue `lex/2` Fakten in die Datenbank übernommen.



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

Aufgabe 1:

1. Legt eine Datei `lexikon.pl` an, in der Lexikon von Seite 58 in der Form von `lex/2`-Fakten realisiert ist.
2. Legt eine Datei `patterns.pl` an, in der die Satzmuster von Seite 58 in der Form von `muster/1`-Fakten realisiert sind.
3. Legt eine Datei `grammatik1.pl` an
 - a) Lasst darin die Dateien `lexikon.pl` und `patterns.pl` per `consult/1` aufrufen.

Anmerkung:

Die Datei mit der Grammatik benötigt sowohl die Information aus dem Lexikon als auch die Satzmuster. In diesem Fall bietet es sich an, die entsprechenden Dateien, also `lexikon` und `patterns` aus der anderen Datei heraus aufzurufen. Dazu verwendet man – wie auf der Prolog-Oberfläche – `consult/1`, und zwar als Befehl in der Datei `grammatik1.pl`. Dabei ist die Eingabe zu beachten: ein solcher Befehl wird durch das 'Falls'-Zeichen ausgelöst, also `:- consult(lexikon)`, wie in dem nachstehenden Screen-Shot:



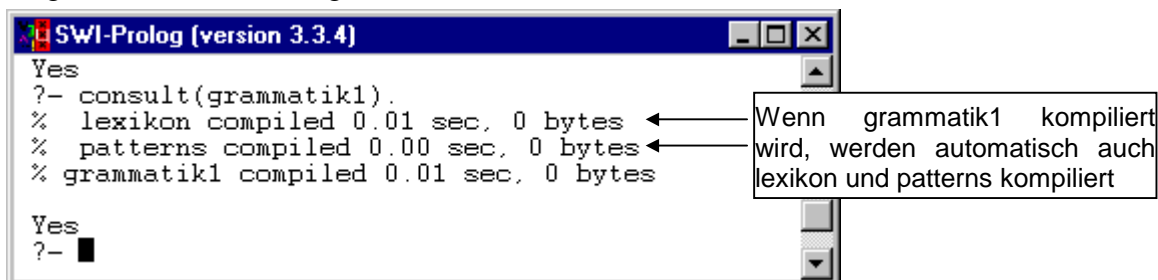
```

:- consult(lexikon).
:- consult(patterns).

satz:-

```

Wenn eine solche Datei eingelesen wird, werden auch automatisch die aus der Datei heraus aufgerufenen Dateien eingelesen:



```

Yes
?- consult(grammatik1).
% lexikon compiled 0.01 sec, 0 bytes
% patterns compiled 0.00 sec, 0 bytes
% grammatik1 compiled 0.01 sec, 0 bytes

Yes
?- █

```

- b) Definiert anschließend das Prädikat `satz/0`, welches die in Abschnitt 6.1.2.3 vorgestellten Eigenschaften hat

Aufgabe 2:

1. Erweitert die Datei `lexikon.pl` (an den relevanten Stellen) um das Numerusmerkmal.
2. Erweitert die Satzmuster in der Datei `patterns.pl` um das Numerusmerkmal.

Aufgabe 3:

1. Erweitert die Datei `lexikon.pl` um die Prädikate `nachschlagen/2` und `hole_merkmal/2`
2. Erweitert das Prädikat `lexkat/2` in der Datei `grammatik1.pl` so, dass bei der Abbildung einer Wortliste auf eine Kategorienliste zunächst überprüft wird, ob das Wort im Lexikon steht (setzt hier also `nachschlagen/2` ein).