Informationswissenschaft, Universität Regensburg, Sommer 2004:

Seminar "Informationslinguistik"

Quenya Syntax I: Verbal-Phrasen

Hubert Feyrer <hubert@feyrer.de>

7. August 2004

Zusammenfassung

Der vorliegende Text beschreibt einen Teil der Ergebnisse der Syntaxgruppe des Seminars "Informationslinguistik". Grundlagen zur Parser-Implementierung in Prolog und zum Schreiben von Quenyanisch in LaTeX folgen die Dokumentation der Projektstruktur sowie das Vorgehen, aus einer Reihe von Beispielsätzen eine Grammatik zu entwickeln sowie diese anschließend in einem Prolog-Parser zu realisieren. Das Durchführen manueller und automatisierter Texts wird kurz erläutert sowie die Ergebnisse bewertet.

Inhaltsverzeichnis

1 Einleitung

Durch die filmische Umsetzung von J. R. R. Tolkien's Trilogie "Der Herr der Ringe"[1] durch Peter Jacksom[2] ist diese kürzlich ins Rampenlicht gestellt und damit einem breiten Publikum zugänglich gemacht worden. Ursprünglich nur in Romanform vorhanden, wurde so eine komplette Fantasy-Welt mit eigener Landschaft, einer Bevölkerung aus verschiedensten Rassen sowie ihren dazugehörigen Sprachen umgesetzt. Daß Tolkien's ursprüngliche Motivation weniger die Erschaffung einer Fantasy-Welt mit all ihren Eigenheiten war ist dabei eher unbekannt. Vielmehr beschäftigte er sich – durch seinen Beruf als Sprachwissenschaftler motiviert – mit der Erschaffung neuer Sprachen, um die herum er seine Romane erschuf[3].

Insofern ist es auch nich verwunderlich, wenn sich Sprachwissenschaftler ernsthaft mit den von Tolkien erschaffenen Sprachen wie Quenya oder Sindarin auseinandersetzen, und auch im Seminar "Informationslinguistik" wurde die von Tolkien geschaffene Sprache "Quenya" dazu benutzt, die Eigenschaften und Verarbeitung einer Sprache zu betrachten.

Ziel des Seminars war, ein sprachverarbeitendes System mit allen Bereichen wie Phonologie, Morphologie, Syntax und Semantik zu entwerfen, das die quenyanischen Sprache mittels eines in Prolog[4] realisierten Parsers umsetzte. Dieser Text beschreibt die Umsetzung eines Teilprojekts mit dem Fokus auf die Syntax der quenyanischen Sprache, der Umsetzung und Realisierung eines Parsers sowie – begrenzt – der Evaluation des Parseres.

Seitenangaben der Form "less-X/Y"in [Helge Kåre Fauskanger, 2004] beziehen sich auf die PDF-Version der von RTF konvertierten, im Web verfügbaren Texte. Die original-RTF Dateien enthalten jedoch keine Seitennummern und sind dadurch kaum einzeln referenzierbar sind, weshalb sie mittels OpenOffice nach PDF konvertiert wurden. Bei den Referenzen ist "X" dabei die Datei ("less-a" bis "less-d"), "Y" die Seitennummer innerhalb der Datei.

2 Grundlagen

Nachdem der Syntax-Teil des Seminar von zwei Gruppen mit insgesamt fünf Teilnehmern bestritten wurde soll hier nur ein kleiner Teil der Grundlagen diskutiert werden – zum einen die Implementierung von Parsern in Prolog, zum anderen wie man die Quenyanische Schrift im Schriftsatzsystem LaTeX benutzt.

[1] [Tolkien, 2000]
[2] [IMDB, 2004] "Lord of the Rings: The Fellowship of the Ring (2001)"
[3] [Tolkien, 2000] S. 1197ff
[4] [Clocksin and Mellish, 2003]

Weitere Grundlagen zu Grammatik allgemein und im Quenianischen im Speziellen, Wortstellung sowie Kongruenzbedingungen werden von den weiteren Mitgliedern der Syntaxgruppen I und II bearbeitet. Eine Liste der Gruppenteilnehmer ist in Abschnitt 3.1 zu finden.

2.1 Parser-Implementierung in Prolog

Im Folgenden werden Kontextfreie Grammatiken (Context Free Grammars, CFGs) sowie zwei mögliche Realisierungen von Parsern in Prolog diskutiert, jeweils mit ihren eigenen Nachteilen. Als Alternative Lösung existieren Direct Clause Grammatiken (DCGs), die die besten Eigenschaften der beiden Alternativen vereinigen, wobei jedoch auf die sachgemässe Anwendung Wert gelegt werden muss. Durch zusätzliche Argumente auf der linken und rechten Seite können die Regeln noch wesentlich mächtiger gestaltet werden um die Darstellung von Grammatiken zu vereinfachen.

2.1.1 Context Free Grammars (CFGs)

Kontextfreie Grammatiken sind eine endliche Sammlung an Regeln, die besagen dass manche Sätze grammatikalisch (d.h. syntaktisch) korrekt sind, und wie die grammatikalische Struktur aufgebaut ist. Sie besteht aus terminalen Symbolen (Alphabet) und non-terminalen Symbolen (Regeln). Anhand der Regeln kann die Struktur des Satzes z.B. in Form eines Parser-Baum aufgestellt und die Richtigkeit eines Satzes verifiziert werden. Ein kontextfreier Parser ist ein Programm das entscheidet, ob eine Zeichenkette (Wort oder Satz) zu der Sprache gehört, die durch die Kontextfreie Grammatik definiert wird, und wie die zugehörige Struktur ist. Eine kontextfreie Sprache umfasst alle Sätze, die mit Hilfe einer kontextfreien Grammatik erschaffen werden können.

2.1.2 CFG Beispiel

“a woman shoots a man”:

```
s -> np vp              sentence
np -> det n             noun phrase
vp -> v np              verb phrase
vp -> v                 verb phrase
det -> a                ?
det -> the              ?
n -> woman              noun
n -> man                noun
v -> shoots             verb
```

2.1.3 CFG-Implementierung mit append()

Die einzelnen Bestandteile werden als Listen betrachtet, die in einer bestimmten Reihenfolge aufgebaut sein müssen:

```
[a,woman,shoots,a,man]
```

Mit der Regel "`s -> np vp`" d.h. ein Satz besteht aus einer Nominal-Phrase gefolgt von einer Verb-Phrase die aneinandergereiht werden, und damit ergibt sich in Prolog:

```
s(Z) :- np(X), vp(Y), append(X, Y, Z).
```

sowie weiterhin:

```
np(Z) :- det(X), n(Y), append(X,Y,Z).
vp(Z) :-  v(X), np(Y), append(X,Y,Z).
vp(Z) :-  v(Z).
det([the]).
det([a]).
n([woman]).
n([man]).
v([shoots]).
```

Problem dabei ist, dass für die Verifikation komplexer Sätze viele durch Prolog mittels Durchprobieren aller Möglichkeiten (via Backtracking) generierte (instantiierte) Sätze mit dem eingegebenen Satz verglichen werden, was sehr aufwendig ist, da max. einer (1) der generierten Sätze passt, aber viele korrekte Sätze erst generiert und verglichen werden müssen - im Worst Case alle korrekten Sätze einer Sprache was *sehr* aufwendig und zeitintensiv sein kann.

Abhilfe: es werden nicht blind Sätze generiert, sondern der Eingabestring (Liste) aufgesplittet und gegen die Regeln verglichen:

```
s(Z) :- append(X,Y,Z), np(X), vp(Y).
np(Z) :- append(X,Y,Z), det(X), n(Y).
vp(Z) :-  append(X,Y,Z), v(X), np(Y).
```

Aber: Dabei wird durch append() der Eingabestring in die verschiedenen Möglichkeiten

```
X=[the],Y=[women,shoots,a,man],
X=[the,woman],Y=[shoots,a,man],
...
```

aufgesplittet und gegen die Regeln verglichen, was nicht viel weniger aufwendig und zeitintensiv ist, v.a. für umfangreichere Texte!

2.1.4 CFG-Erkennung mit Differenzlisten

Eine Alternative Implementierung von CFGs besteht nun mit Hilfe von Differenzlisten. Die Idee hier ist, nicht verschiedene mögliche Sätze zu bilden und diese ggf. gegen die Eingabe zu vergleichen, sondern die Eingabe von links nach rechts durchzulesen, und das jeweils am weitesten links stehende Eingabe-Token (ggf. wieder durch rekursiven Abstieg) zu klassifizieren, und zu überprüfen ob die Komponenten innerhalb der Grammatik erlaubt sind:

```
s(X,Z)  :- np(X,Y), vp(Y,Z).
np(X,Z) :- det(X,Y), n(Y,Z).
vp(X,Z) :- v(X,Y), np(Y,Z).
vp(X,Z) :- v(X,Z).
det([the|W],W).
det([a|W],W).
n([woman|W],W).
n([man|W],W).
v([shoots|W],W).
```

Damit wird der Satz "the women shoots a man" wie folgt analysiert:

```
the:    s -> np -> det -> [the],       Rest: [woman,shoots,a,man]
B(backtracking det -> np, 2. Teil)
woman:  n -> [woman],                        Rest: [shoots,a,man]
(backtracking n -> np -> s, 2. Teil)
shoots: s -> vp -> v -> [shoots],                   Rest: [a,man]
(backtracking v -> vp, 2. Teil)
a:      np -> det -> [a],                             Rest: [man]
(backtracking n -> np, 2. Teil)
man:    n -> [man],                                      Rest: []
```

Vorteil des Ansatzes mit Differenzlisten ist, dass er wesentlich effizienter ist da kein unnötiges Backtracking ("probieren") geschieht, der Parser jedoch auch schwerer zu lesen (und verstehen!) ist.

Abhilfe bietet die Definite Clause Grammar, die die Lesbarkeit des ersten Ansatzes und die Effizienz des zweiten Ansatzes vereint.

2.1.5 Definite Clause Grammars (DCG)

Definite Clause Grammatiken (DCG) bieten eine Schreibweise für Grammatiken an, die die unterliegenden Differenzlisten versteckt:

```
s   --> np,vp.
np  --> det,n.
vp  --> v,np.
vp  --> v.
det --> [the].
det --> [a].
n   --> [woman].
n   --> [man].
v   --> [shoots].
```

Diese der ursprünglichen Form doch sehr ähnliche Schreibweise kann wie die Differenzliste-Variante abgefragt werden:

```
s([the,woman,shoots,a,man],[]).
```

Die Schreibweise mit "`-->`" ist hierbei nur syntaktischer Zucker: die komplexe Schreibweise der Differenzlisten-Variante wird versteckt und durch eine einfache Schreibweise ersetzt, die jedoch intern effizient umgesetzt wird!

Wovor DCG-Regeln jedoch nicht schützen sind Fehler die man in jedem Prolog-Programm machen kann, v.a. wenn man rekursive Grammatiken definiert. Hierbei kann nur eine sorgfältige Spezifikation der Grammatik sorgen.

Beispiel: um zwei Sätze zu verbinden könnte die bestehende Grammatik um die folgenden Regeln erweitert werden:

```
s    --> s,conj,s.
conj --> [and].
conj --> [or].
conj --> [but].
```

Dies führt jedoch zu einer sofortigen Endlos-Rekursion ($s \rightarrow s \rightarrow s \rightarrow \ldots$), eine saubere Abhilfe ist hier nur durch eine korrekte Spezifikation der Grammatik zu erreichen:

```
s        --> simple_s.
s        --> simple_s conj s.
simple_s --> np vp.
...
```

2.1.6 Erweiterung von DCGs durch zusätzliche Argumente

Mit Hilfe von zusätzlichen Argumenten können DCGs erweitert werden, um z.B. Einzuschränken, an welcher Stelle ein bestimmtes grammatikalisches Objekt stehen darf und wo nicht, zu realisieren.

Beispiel: Es sollen Pronomen “he”, “she”, “him”, “her” eingeführt werden. Die Nomen-Regel wird zusätzlich entsprechend erweitert:

```
np  --> pro.
pro --> [he].
pro --> [she].
pro --> [him].
pro --> [her].
```

Problem hierbei ist, dass auch Sätze wie “a woman shoot she” etc. als gültig angesehen werden, da nicht unterschieden wird, ob das Pronom in der Subjekt- oder Objkekt-Position steht.

Abhilfe ist durch zusätzliche Non-Terminale möglich:

```
s --> np_subject,vp.

np_subject --> det,n.
np_object  --> det,n.
np_subject --> pro_subject.
np_object  --> pro_object.
vp --> v,np_object.
vp --> v.
pro_subject --> [he].
pro_subject --> [she].
pro_object --> [him].
pro_object --> [her].
...
```

Aus der übersichtlichen Grammatik ist damit etwas relativ unhandliches geworden. Wenn man nun weitere Attribute wie Singular/Plural mit berücht-sichtigen will wird die Situation nur noch um vieles Schlimmer.

Eine bessere Lösung ist hier, das Nomen in der Grammatik syntaktisch zu kennzeichnen, je nachdem ob es im Subjekt oder im Objekt stehen soll:

```
s --> np(subject),vp.
np(_) --> det,n.
np(X) --> pro(X).
vp --> v,np(object).
vp --> v.
pro(subject) --> [he].
pro(subject) --> [she].
pro(object) --> [him].
pro(object) --> [her].
...
```

Hiermit kann realisiert werden, dass ein Pronomen vor einem Subjekt eine andere Form als vor einem Objekt besitzt, ohne jedoch zusätzlich weitere Grammatik-Regeln einzuführen. Dies ist entscheidend, da die Grammatik weiterhin einfach und lesbar bleibt.

Mithilfe der zusätzlichen Argumente für DCG-Klauseln können weiterhin semantische Belange von Texten erfasst werden, die erst z.B. durch die Bildung von Parser-Bäumen aufgedeckt werden, vgl. [Blackburn et al., 2004][5].

2.1.7 Erweiterung von DCGs durch zusätzliche Ziele

Nachdem – wie bereits festgestellt – DCGs nur eine andere Schreibweise für Prolog-Klauseln sind, kann man natürlich auch auf der rechten Seite weitere Prädikate angeben. Diese Prädikate können dann ebenfalls weitere überprüfungen und Einschränkungen machen, wie z.B. zum Trennen zwischen Lekikon und Grammatik-Regeln benutzt werden:

```
lex(the,det).
lex(a,det).
lex(woman,n).
lex(man,n).
lex(shoots,v).

det --> [Word], { lex(Word,det) }.
n   --> [Word], { lex(Word,n) }.
v   --> [Word], { lex(Word,v) }.
```

Damit werden im Lexikon Attribute von Worten definiert, und der Zugriff erfolgt getrennt über Regeln, die bestimmte Attribute benutzen. Anstatt über Prädikate mit fester Ordinalität können die Attribute auch in Listen von (Attribut-Typ, Attribut) gehalten werden, z.B.

```
ne([[cat, noun], [stem, andond], [num, singular],
    [casus, alt([nominative, possessive, allative, instrumental])],
    [germ, [tor]]]) -->         "andon".
```

2.1.8 Zusammenfassung

Zusammenfassend kann gesagt werden, dass Kontextfreie Grammatiken (CFGs) einfach zu lesen sind, die Implementierung mittels append() jedoch ineffizient und die Implementierung mittels Differenzlisten ist unleserlich ist. Als Abhilfe bieten Direct Clause Grammatiken (DCGs) eine

[5] [Blackburn et al., 2004] 8.1.2 Building parse trees, ganz unten

Schreibweise, um das beste aus beiden Arten zu erhalten: lesbar und effizient. Auch beim Einsatz von DCGs muss auf korrekte Spezifikation der Grammatik geachtet werden. Durch zusätzliche Argumente können DCGs erweitert werden, ohne die Grammatik zu verändern, und durch zusätzliche Argumente auf der rechten Seite können den Grammatik-Regeln von DCGs weitere Auflagen gemacht werden.

Im Rahmen des vorliegenden Seminars wurden die Punkte "Erstellung von Syntaxbäumen" sowie die Behandlung von Kongruenzbedingungen der Syntaxgruppe II überantwortet.

2.2 Quenianisch schreiben mit LaTeX

Bei der Umsetzug von Quenianisch am Computer stellt sich neben Syntax, Grammatik und Wortschatz auch die Frage nach der Schrift, da Tolkien neben den verschiedenen Sprachen auch entsprechende Schriften ersann[6]. Bei Verwendung von LaTeX als Schriftsatzsystem können Quenyanisch und weitere elbische Schriften bequem mit Hilfe des TgTeX-Pakets (sprich: "Teng-Tech") benutzt werden[7]. Abgeleitet von der dem Quenyanischen übergeordneten Sprache "Tengwar" umfaßt TgTeX nicht nur den quenyanischen Dialekt, sondern weiterhin auch Sindarin und Beleriandisch.

Das TgTeX-Paket erhältlich, es enthält zum einen Zeichensätze, zum anderen LaTeX-Macros, um diese Schriften einfach zu benutzen.

Bei der Benutzung muß darauf geachtet werden daß die Schriften und Macros entweder in dem TeX-System bekannten Verzeichnissen installiert werden, oder aber es wird explizit angegeben wo sie zu suchen sind, z.B. wenn die Schriften im Verzeichnis ../mf und die Macros im Verzeichnis ../tex liegen:

```
env MFINPUTS=../mf: TEXINPUTS=../tex:  latex tengdoc.tex
```

Zur Benutzung muß im LaTeX-Eingabetext dann zuerst der Tengwar-Dialekt ausgewählt werden. Hierzu stehen die drei Macros `\quenya`, `\sindarin` und `\beleriand` zur Verfügung. Eines dieser Macros muß vor dem Text in einem der elbischen Schriften benutzt werden, da kein Default existiert.

Nach der Auswahl des Dialogs kann innerhalb der "elvish"-Umgebung Text in der lateinischen Umschrift gesetzt werden, wobei die lange Vokale durch Wiederholung des jeweiligen Zeichens geschieht. Z.B. wird der Text[8]

Ai! laurië lantar lassi sūrinen,

[6] [Tolkien, 2000] S. 1181ff
[7] [Derzhanski, 2000]
[8] [Namárië, 2004]

yēni ūnōtimë ve rāmar aldaron!

wie folgt mit Hilfe von LaTeX und TgTeX gesetzt:

```
\quenya
\begin{elvish}
ai! laurie lantar lassi suurinen,
yeeni uunootime ve raamar aldaron!
\end{elvish}
```

was dann zu folgendem Ergebnis führt:

ai! laurie lantar lassi suurinen,
yeeni uunootime ve raamar aldaron!

3 Umsetzung

Zur Umsetzung des Projekts wurde die Projektstruktur inkl. Aufgabenbereiche erörtert. Anschliessend wurde die Umsetzung des Parsers durch die Analyse von Beispielsätzen und Erstellen einer Grammatik bestimmt. Die Projektstruktur sowie das Vorgehen zur Parser-Erstellung sowie die Realisierung des Parsers werden im Folgenden beschrieben.

3.1 Projektstruktur

Innerhalb des Seminars wurden mehrere Gruppen mit verschiedenen Aufgaben betreut. Die Zusammenhänge zwischen den einzelnen Gruppen ist in Abbildung 1 aufgezeigt, die Arbeitsaufteilung innerhalb der Syntaxgruppe ist in Abbildung 2 zu sehen.

Im Folgenden sollen die einzelnen Gruppen kurz inkl. EMail-Adressen zur Kontaktaufnahme aufgelistet werden, damit Fragen möglichst effizient beantwortet werden können. Weiterhin soll die Aufteilung der Aufgaben innerhalb der Syntaxgruppe näher festgelegt werden.

3.1.1 Gruppen und Mitarbeiter

Syntax I:

Claudia Henghuber <henghuber@gmx.de>

Hubert Feyrer <hubert@feyrer.de>

Syntax II:

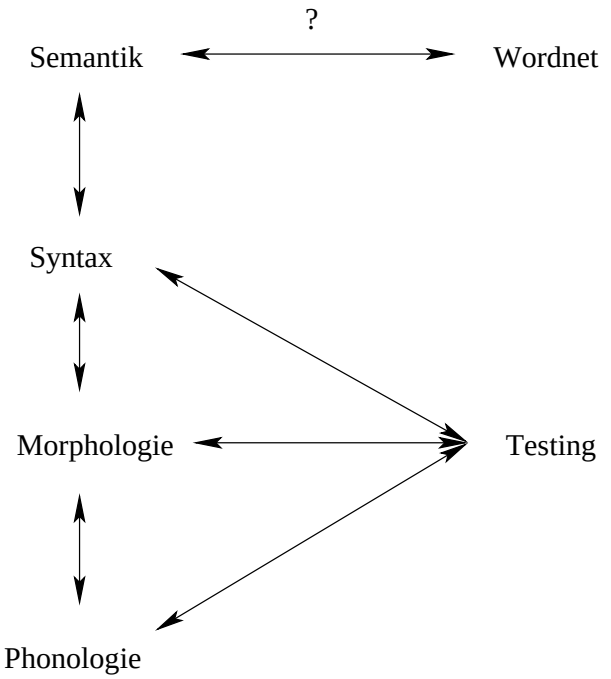

Abbildung 1: Projektstruktur mit allen Gruppen

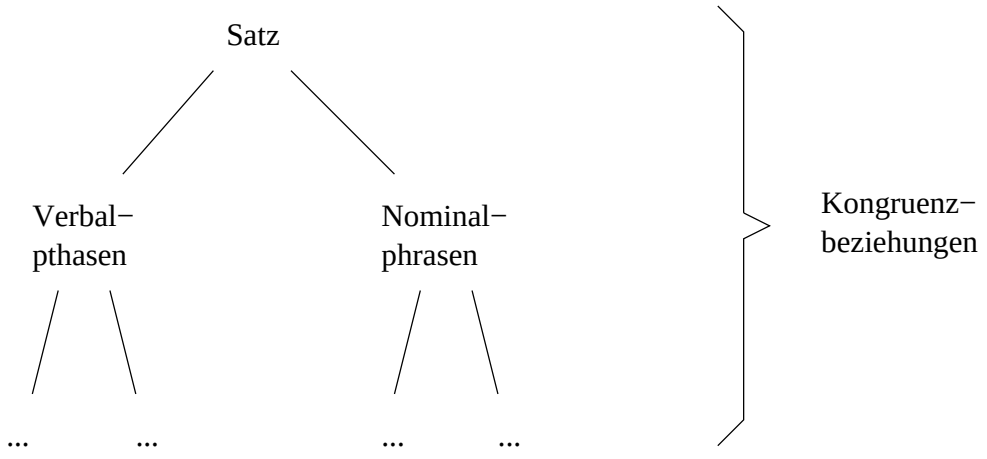

Abbildung 2: Interne Struktur der Syntaxgruppe

Sonja Rachelski <sonja.rachelski@t-online.de>

Anna Schreiber <anna.schreiber@stud.uni-regensburg.de>

Torsten Griebl <malle23@gmx.de>

Testing:

Unbekannt.

Semantik:

Rainer Kohlmann <rainer.kohlmann@web.de>

Morphologie / Phonologie:

Unbekannt, eigene EMail-Adresse wurde jedoch für Fragen weitergegeben

Siehe unten bzgl. Fragen!

Offene Tätigkeiten:

- Lexikongruppe: Bis dato ist unklar, wer das Lexikon pflegt und um neue Einträge erweitert:

  ```
  ?- analyze("massa", X).
  not_in_lexicon(massa)
  ```

- Partikel “na” und “la” um Alternativen “ná”, “naa”, “lá” und “laa” erweitern

3.1.2 Aufgabenverteilung

Die Aufgabenverteilung innerhalb der Syntaxgruppe ergibt sich wie folgt:

Hubert: Implementierung Satz, Projektmanagement, Testing; Ausarbeitung Parser-Implementierung in Prolog, Satz und automatisiertes Testing

Claudia: Implementierung Verbalphrasen (VP); Ausarbeitung Grammatik-Grundlagen und VP

Thorsten: Implementierung Kongruenzbeziehungen; Ausarbeitung: dito (nur Programmierung, hat beschlossen den Schein nicht zu machen).

Sonja, Anna: Implementierung Nominalphrase (NP); Ausarbeitung: NP, ggf. Anbindung an Semantik (Syntaxbaum) oder Kongruenzregeln in Quenya

3.1.3 Randbedingungen

- Interfacing mit Testgruppe: Syntaxgruppe (S, NP, VP) liefert Beispielsätze
- Semantikgruppe: Syntaxgruppe sollte Syntaxbaum liefern
- Morphologie: Interface steht, Fragen direkt an Prof. Dr. Hammwöhner
- Erweiterbarkeit, falls Testing-Gruppe neue Testfälle findet, die die bestehende Grammatik nicht beherrscht/abdeckt

3.2 Vom Beispielsatz zur Grammatik

Für die praktische Realisierung eines Quenya-Parsers standen zwei Wege zur Verfügung. Zum einen einen fertig existierenden Parser um bestimmte Eigenschaften zu erweitern, zum anderen einen relativ einfachen Parser selbst umzusetzen. Die Entscheidung fiel auf den zweiten Ansatz, da innerhalb der Syntax-Gruppen zu wenig Prolog-Wissen vorhanden war um den gesamten existierenden Parser-Code zu lesen und zu verstehen. Stattdessen sollte durch das Erstellen eines eigenen Parsers Verständnis für die Grundfunktionalitäten erworben werden, mit dem Hinblick auf mögliche Erweiterbarkeit.

Ausgehend von einer Anzahl ausgewählter Beispielsätze wurde eine Sammlung von Regeln aufgestellt, die dann weiter zu einer Grammatik für Satz- und Verbalphrasen verfeinert wurden. Dieser Prozess ist im Folgenden dargestellt.

Die Betrachtung und Analyse von Nominalphrasen ist dabei bewusst ausgeklammert, da diese von der Syntaxgruppe II behandelt werden, siehe deren Ausarbeitungen.

3.2.1 Beispielsätze

Die folgenden Beispielsätze wurden [Helge Kåre Fauskanger, 2004] entnommen, die Beschreibung umfaßt dabei die folgenden Komponenten:

Beispiel-Schema:

	𝔮𝔯𝔶̂	
	Quenya	Fauskanger-Referenz
	Deutsch	(Kommentar)
⇒	Grammatik	

Für jeden Beispielsatz werden die original elbische Schreibweise sowie die lateinische Umschrift (“Quenya”) sowie die deutschsprachige Übersetzung

(“Deutsch”) angegeben. Zur Belegung des Beispiels ist die Referenz in [Helge Kåre Fauskanger, 2004] (“Fauskanger-Referenz”) sowie ggf. ein Kommentar angegeben, der die syntaktische bzw. grammatikalische Eigenheit näher bestimmt. Die daraus abgeleitete Grammatik wird zuletzt dargestellt.

Hier nun die Beispielsätze:

Beispiel 1: Einfacher Satz:

	i íʃ̂ m̂îp̂	
	i elda MÁTA	less-a/59
	Der Elf ISST	(a-Verb)

⇒ NP, Verb

Beispiel 2: Satz mit Objekt:

	i íʃ̂ m̂îp̂ m̂ʒ̂	
	i elda MÁTA massa	less-a/59
	Der Elf ISST Brot	(mit Objekt)

⇒ NP, Verb, NP

Beispiel 3: Andere Wortstellung:

	m̂îp̂ i íʃ̂	
	MÁTA i elda	less-a/62
	ES ISST der Elf	

⇒ Verb, NP

Beispiel 4: Adjektiv (Positiv):

	iɔ̇ƨ m̂î q̂ɻ̇m̂	
	isil NÁ calima	less-a/63f
	Der Mond IST hell	(Copula+Adj.)

⇒ NP, Copula, Adjektiv(Positiv)

Beispiel 5: Komparativ:

	iɔ̇ƨ m̂î q̂ɻ̇m̂ ƨ̂î íƶm̂	
	isil ná calima LÁ eleni	less-a/63f
	Der Mond ist hellER ALS die Sterne	

⇒ NP, Copula, Adjektiv(Komparativ), NP

Beispiel 6: Superlativ:

iϡζ ḿî ĵœq̂ζ̇m̂
isil ná ANcalima less-a/63f
Der Mond ist AM hellSTEN

⇒ NP, Copula, Adjektiv(Superlativ)

Beispiel 7: Wörtliche Rede:

íq̃ ĵζ̇ṗmζ: íp ĩ̂ỹẑ íp̃mỹm̃ iṗ̈ζ̇ím:
EQUË Elendil: ... less-c/52
Elendil SPRACH: ... (zeitlos)

⇒ "eque", NP, ":", E

Beispiel 8: Relativsatz:

i λpm̂ ḿî îζp̂
i harma ná alta less-c/76
der Schatz ist groß

λpḿζ̃
hirnelyë
Du hast ihn gefunden

i λpm̂ q̧̂ λpḿζ̃ ḿî îζp̂
i harma YA hirnelyë ná alta
der Schatz, DEN du gefunden hast, ist groß

⇒ NP, "ya", E, VP

Beispiel 9: Imperativ:

λ̃q̂
heca! less-d/10
Geh zur Seite!

⇒ Verb(imperativ)

Beispiel 10: Verb+Subjekt (1.P):

ḿζ̃m
melON less-c/76
ICH liebe (-n/-nye)

⇒ Verb("-nye")

Beispiel 11: Verb+Subjekt und Objekt:

ḿζ̃m m̂ŝ
melON massa less-c/76
ICH liebe Brot (-n/-nye)

⇒ Verb("-nye"), NP

Beispiel 12: Verb+Subjekt (2.P):

hiruvaLYË	less-b/33
DU wirst finden	(-lye)

⇒ Verb(“-lye”)

Beispiel 13: Verb+Subjekt+Objekt

melONLYË	less-b/33
ICH liebe DICH	-SE-OE

⇒ Verb(“-nye”, “-lye”)

Merke:

Subjekt-Endung vor Objekt-Endung[9]!

3.2.2 Resultierende Grammatik, erste Version

Zusammengefaßt ergibt sich aus den vorangegangenen Beispielen folgende Grammatik:

1. S → NP, Verb
2. S → NP, Verb, NP
3. S → Verb, NP
4. S → NP, Copula, Adjektiv(Positiv)
5. S → NP, Copula, Adjektiv(Komparativ), NP
6. S → NP, Copula Adjektiv(Superlativ)
7. S → “eque”, NP, “:”, E
8. S → NP, “ya”, E, VP
9. S → Verb(imperativ)
10. S → Verb(“-nye”)
11. S → Verb(“-nye”), NP
12. S → Verb(“-lye”)
13. S → Verb(“-nye”, “-lye”)

[9] [Helge Kåre Fauskanger, 2004] less-b/36

3.2.3 Quenya-Grammatik: Satz

Mit etwas Optimierung erfolgt daraus folgende Grammatik für einen Satz in Quenya:

1. S → NP, VP
2. S → Verb, NP
3. S → “eque”, NP, “:”, E
4. S → NP, “ya”, E, VP
5. S → Verb(imperativ)
6. S → Verb(“-nye”)
7. S → Verb(“-nye”), NP
8. S → Verb(“-lye”)
9. S → Verb(“-nye”, “-lye”)

3.2.4 Quenya-Grammatik: Verbal-Phrase (VP)

Für die Verbal-Phrase ergibt sich aus den oben genannten Beispielen folgende Grammatik:

1. VP → Verb
2. VP → Verb, NP
3. VP → Copula, Adjektiv(Positiv)
4. VP → Copula, Adjektiv(Komparativ), NP
5. VP → Copula, Adjektiv(Superlativ)

3.2.5 Quenya-Grammatik: Nominal-Phrase (NP)

Die Grammatik für Nominal-Phrasen in Quenya wurden im Seminar von der Syntaxgruppe II bearbeitet und sind in deren Ausarbeitungen näher beschrieben.

3.2.6 Mögliche Erweiterungen und Ergänzungen

Der oben gemachte Ansatz geht von einer Reihe einfacher Sätze aus, die einen gewissen Grundstock der quenyanischen Sprache darstellen, jedoch bei weitem nicht vollständig sind. In Anlehnung an [Helge Kåre Fauskanger, 2004]

kann gesagt werden dass u.a. keine Unterscheidung zwischen starken, schwachen, einfachen und primären Verben[10] sowie negativen Verben[11] gemacht wird. Weiterhin wurden das Gerund[12], Possessivpronomen[13], relative Pronomen[14] sowie Demonstrative[15] und Fragewörter[16] und interrogative Partikel[17] ausgeklammert.

Im Hinblick auf die bekannte Unvollständigkeit der Grammatik wurde bei der Definition und Umsetzung der Grammatikergeln darauf Wert gelegt, dass diese einfach zu erweitern sind.

3.3 Umsetzung der Satz-Grammatik

Hier soll nun die Implementierung des Prolog-Parsers zur Analyse der Satz-Grammatik erläutert werden, weiterhin wird auf die Einbindung in den existierenden Parser besprochen.

3.3.1 Prolog-Implementierung

Die vorliegende Umsetzung des Quenya Syntax-Parsers basiert auf einer fertigen Implementierung, die innerhalb des Seminars als Grundlage zur Verfügung gestelt wurde. Die Aufgabe der Syntax-Gruppe bestand darin, die Syntax-Behandlung mit Hilfe einer neu erstellten Grammatik zu realisieren (siehe Abschnitt 3.2), und dabei den Vorhandenen Code der Bereiche Phonologie, Morphologie und Semantik zu verwenden.

Anhang A listet den Prolog-Code für die Satz-Behandlung auf, der sich in die bestehende Parser-Struktur einfügt, und dabei bestehende Routinen der Morphologie benutzt sowie den erstellten Syntaxbaum zur weiteren Verarbeitung durch die Semantik-Gruppen bereitstellt.

Die Umsetzung der einzelnen in Abschnitt 3.2 ermittelten Grammatikregeln erfolgte in drei Schritten:

1. Formulieren der Grammatik-Regeln unter Verwendung existierender oder selbst definierter Prädikate:

   ```
   s --> np, verb
   ```

   Für die Umsetzung der Satz-Grammatik wurden neben den existierenden morphologischen Prädikaten (`verb`, ...) minimale Implemen-

[10] [Helge Kåre Fauskanger, 2004] less-b/1f
[11] [Helge Kåre Fauskanger, 2004] less-b/48
[12] [Helge Kåre Fauskanger, 2004] less-c/44
[13] [Helge Kåre Fauskanger, 2004] less-c/55
[14] [Helge Kåre Fauskanger, 2004] less-c/76
[15] [Helge Kåre Fauskanger, 2004] less-d/15
[16] [Helge Kåre Fauskanger, 2004] less-d/55
[17] [Helge Kåre Fauskanger, 2004] less-d/60

tierungen von Nominal- und Verbal-Phrasen verwendet, die tiefergehenden Ausführungen dazu sind in den Ausarbeitungen der einzelnen Mitglieder der Syntax-Gruppe wie in Abschnitt 3.1 beschrieben zu finden.

2. Die von den existierenden Prädikaten gelieferten Merkmalslisten wurden mit Variable eindeutig instantiiert:

   ```
   s --> np(NP), verb(V)
   ```

3. Zur semantischen Weiterverarbeitung werden neue Merkmalslisten definiert, die die ermittelte Satzstruktur sowie die genaue Ausprägung in Form eines als Liste implementierten Syntaxbaumes aufstellten und an das aufrufende Prädikat zurückgeben:

   ```
   s([[np, NP], [verb, V]]) --> np(NP), verb(V)
   ```

Die Umsetzung der Grammatikregeln in einen funktionierenden Parser war so relativ leicht, wobei jedoch noch einige Fragestellungen offen blieben bzw. absichtlich offengelassen wurden, da sie von anderen Mitgliedern des Seminars behandelt werden:

- Die Benennung der genauen Merkmale für Satzgrammatik zur weiteren Verarbeitung sollten geklärt werden, insbes. `verb_imperativ`, `verb_nye`, `verb_lye` und `verb_nye_lye`, sowie die beiden Ergänzungen `e1` und `e2`, aber auch `copula`, `adj`, `prep`, `np` und `vp`. Während bei den verschiedenen Verb-Formen eine allgemeine Bezeichnung als `verb` noch naheliegt sind genauere Bezeichnungen für die Weiterverwendung durch die Semantik-Gruppen am besten von diesen konkret vorzuschlagen bzw. die momentan realisierten Bezeichnungen von ihnen zu übernehmen.
- Die Überprüfung von Kongruenz-Bedingungen wurden in der vorliegenden Implementierung bestenfalls durch Kommentare angedeutet, ansonsten bewusst weggelassen da diese durch Mitglieder der Syntaxgruppe II behandelt werden.
- Die beiden Ergänzungen `ergaenzung1` bzw. `ergaenzung2` sollten grammatikalisch näher bestimmt und weiter ausgearbeitet werden.

3.3.2 Einbinden und Funktionstest

Die folgenden Schritte beschreiben, wie die erstellte Satz-Grammatik in der Datei `hf-grammar.pl` in den bestehenden Parser eingebunden werden kann:

- In `syntax.pl` ist anstatt "`consult('grammar.pl')`" "`consult('hf-grammar')`" zu schreiben:

```
:-consult('hf-grammar.pl').   /*HF*/
/* :-consult('grammar.pl'). *//*HF*/
```

- In `interface.pl` ggf. den Aufruf "`:- start.`" entfernen, da die Eingabe-GUI nicht die gleichen Möglichkeiten des `analyze` Prädikates bietet und umständlich zu bedienen ist:

  ```
  /* :-start. *//*HF*/
  ```

- Aufruf: "`xpce -f interface.pl`" (unter Unix/Linux und NetBSD)
- Testen: `analyze("lassi lantar").`

Umfangreichere Tests werden im folgenden Kapitel beschrieben.

4 Evaluation

4.1 Eigene Tests

Für erste bzw. einzelne Tests sind manuelle Testmethoden wichtig, um den Gesamtumfang des Parsers zu testen wurde jedoch ein Automatismus erstellt, der eine Liste von dokumentierten Testfällen auf ihre Parsbarkeit hin überprüft.

4.1.1 Manuell

Zum manuellen überprüfen einzelner Grammatikregeln zeigte sich das `analyze` Prädikat sehr wertvoll, mit dessen Hilfe einzelne Sätze auf ihre Parsbarkeit hin überprüft werden können. Entsprechend wir am Ende "No" bzw. "Yes" und der zugehörige Syntaxbaum ausgegeben:

```
?- analyze("eldar melar eldar").
noun([[pers, '3'], [casus, nominative], [num, plural], [cat, noun],
     [stem, elda], [germ, [elb]]], [eldar|_G476], _G476)
verb([[pers, '3'], [num, plural], [mod, normal], [temp, present],
     [cat, verb], [stem, mel], [germ, [lieben]]], [melar|_G579], _G579)
noun([[pers, '3'], [casus, nominative], [num, plural], [cat, noun],
     [stem, elda], [germ, [elb]]], [eldar|_G673], _G673)

[[np, [noun, [[pers, '3'], [casus, nominative], [num, plural], [cat,
     noun], [stem, elda], [germ, [elb]]]]], [verb, [[pers, '3'],
     [num, plural], [mod, normal], [temp, present], [cat, verb],
     [stem, mel], [germ, [lieben]]]], [np, [noun, [[pers, '3'],
     [casus, nominative], [num, plural], [cat, noun], [stem, elda],
     [germ, [elb]]]]]]

Yes
```

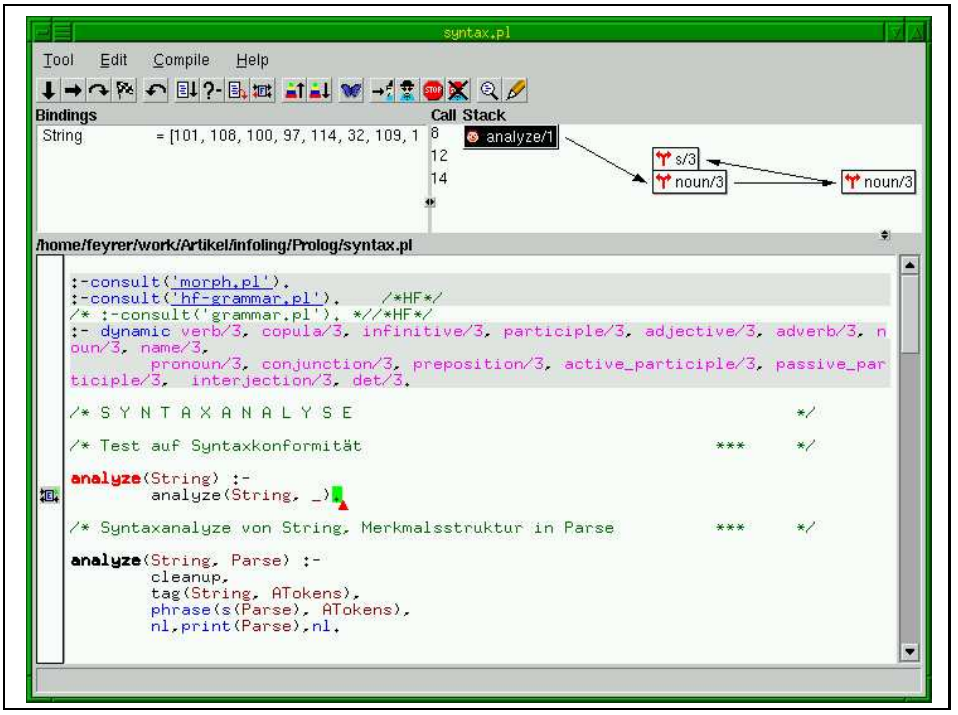

Abbildung 3: Die mit `guitracer` anschaltbare Trace-Oberfläche

Weitere Methoden für sehr tiefes Debuggen (welches im Rahmen dieses Projekts nicht angewandt werden mussten) sind zum einen das `trace` Prädikat, mit dessen Hilfe der Inferenzmechanismus von Prolog im Detail verfolgt werden kann:

```
trace, analyze("eldar melar eldar").
```

Da das Verfolgen des Inferenzmechanismus' rein Textbasierend recht unübersichtlich ist, existiert im SWI-Prolog durch das `guitracer` Pädikat zusätzlich die Möglichkeit, ein grafisches Tracing-Verfahren hinzuzuziehen:

```
guitracer.
trace, analyze("eldar melar eldar").
```

Eine Abbildung der grafischen Tracing-Oberfläche in Aktion ist in Abbildung 3 zu sehen.

4.1.2 Automatisiert

Das `analyze`-Prädikat ist für das Testen einzelner Fälle sehr hilfreich, um einen größeren Umfang an Testfällen automatisch zu testen muß es jedoch in eine Test-Infrastruktur eingebaut werden.

Da der Quellcode des Parsers bereits für jede Grammatik-Regel Testfälle in Form von Kommentaren enthält, liegt es nahe, diese aus dem Prolog-Quellcode zu extrahieren und zum automatischen Test zu nutzen, was im Folgenden beschrieben ist.

Vorbereitung: Jeder automatisch zu testende Fall muß im Prolog-Quellcode (Dateiname: `hf-grammar.pl`, vgl. Anhang A) folgendermaßen als Kommentar formatiert sein:

```
/*
 * Test: "lassi lantar"
 * Test: "i elda maata"
 * Test: "i elda máta"
 */
```

Implementierung: Das automatische Testverfahren ist als Prozedur von Unix-Befehlen realisiert, die die Textfälle aus dem Prolog-Quellcode extrahiert und in deren Mittelpunkt der Prolog-Parser gestartet und mit entsprechenden `analyze`-Befehlen programmiert wird. Je nach Erfolg oder Misserfolg wird das ausgegebene "Yes" bzw. "No" dann für die weitere Verarbeitung in LaTeX-Tabellenform ausgegeben.

Hier die Implementierung des Testverfahrens als Makefile:

```
test:
        cd Prolog ;                                 \
        grep ^...Test: hf-grammar.pl                \
        | sed -e 's/.*Test: //'                     \
        | while read s ;                            \
          do                                        \
                printf "%-60s " "$$s & " ;          \
                echo "analyze($$s)."                \
                | xpce -f interface.pl 2>&1         \
                | egrep 'Yes|No' ;                  \
                echo '\\' ;                         \
          done                                      \
        | sed -e 's/"//g'                           \
        | tee test-output.tex
```

Testen: Die Testprozedur wird mit "`make test`" gestartet. Die Laufzeit beträgt ca. 5 Minuten[18], und die Ausgabe wird parallel auf den Bildschirm und eine Datei `test-output.tex` geschrieben:

```
noon% make test
```

[18]Hardware: Toshiba Portégé 3440CT Laptop mit 500MHz Pentium III, 128MH RAM; Betriebssystem: NetBSD 1.6.2/i386; Software: SWI-Prolog 5.2.13

```
"lassi lantar" &                                Yes
\\
"i elda maata" &                                No
\\
"eldar melar eldar" &                           Yes
\\
...
```

Ergebnis: Die folgende Tabelle mit den Testergebnissen wird aus der mittels "`make test`" erstellten Datei `test-output.tex` erzeugt:

Satz	Geparst
lassi lantar	Yes
i elda maata	No
i elda máta	Yes
eldar melar eldar	Yes
i elda maata massa	No
i elda máta massa	Yes
melar i eldar	Yes
maata i elda	No
máta i elda	Yes
isil na calima	Yes
isil naa calima	No
isil na calima la eleni	Yes
isil naa calima laa eleni	No
isil na ancalima	Yes
isil naa ancalima	No
eque i eldar: eldar massar eldar	No
eque elendil: et earello endorenna utuulien	No
i elen ya hiruvalye naa ancalima	No
i harna ya hirnelye na alta	No
i harna ya hirnelye naa alta	No
heca	No
melon	No
hiruvalnye	No
melon massa	No
hiruvalye	Yes
melonlye	No

4.2 Bewertung der Testergebnisse

Die Testergebnisse aus dem vorherigen Kapitel können wie folgt zusammengefaßt werden:

- “máta” vs. “maata”: Es wäre schön wenn die Umschreibung der Akzente zur Lautverdopplung erkannt werden würden. Dies ist wohl innerhalb der Phonetik oder Morphologie zu lösen. Dies gilt auch für “ná” vs. “naa” und “lá” vs. “laa”.
- Die Erkennung einzelner Worte ist momentan unklar. Dies betrifft insbesondere “eque” und “ya”.
- Es existieren keine (mir bekannten?) Möglichkeiten, Wortendungen zu überprüfen. Dies betrifft v.a. die Endungen “-nye” (bzw. “-n”), “-lye” sowie eine Kombination der beiden. Ein klares Problem für die Morphologie, das in den Prädikaten `verb_nye`, `verb_lye` und `verb_nye_lye` realisiert werden sollte.
- Die Überprüfung ob ein Verb im Imperativ vorliegt ist momentan ebenfalls ungelöst. Ein weiteres Problem für die Morphologie-Gruppe, das im `verb_imperativ` abgeklärt werden sollte.

4.3 Testing-Gruppe

Angeblich existierte im Rahmen des Seminars eine eigene Testing-Gruppe. Da diese jedoch nicht merklich in Erscheinung trat konnten hier keine weiteren Tests besprochen und durchgeführt werden, so daß nur die oben beschriebenen eigenen Tests durchgeführt werden konnten.

5 Zusammenfassung

Die vorliegende Arbeit dokumentiert den Ansatz der beiden Syntax-Gruppen, ausgehend von der Satz-Grammatik diese zu erweitern und einen Parser für einen Satz von bekannten Sätzen und Grammatikregeln zu erstellen sowie automatisierte Tests über die bekannten Testfälle laufen zu lassen.

Fragestellungen an die Gruppen Morphologie und Semantik bleiben leider ebenso offen wie die Ergebnisse umfangreicherer Tests mit Hilfe eines umfangreichen Korpus, da dies nicht Koordiniert wurde.

A Quelltext: hf-grammar.pl

```
/*
 * hf-grammar.pl: Grammatikregeln
 */


/*****************************************************************************
*****************************************************************************
```

```
***
*** SATZ - Hubert
***
*****************************************************************************
*****************************************************************************/

/*****************************************************************************
 * Beispiel 1: Einfacher Satz
 *
 * Test: "lassi lantar"
 * Test: "i elda maata"
 * Test: "i elda máta"
 *
 *****************************************************************************/
s([[np, NP], [verb, V]]) -->
        np(NP),
        verb(V)
        /* Kongruenz */.


/*****************************************************************************
 * Beispiel 2: Satz mit Objekt
 *
 * Test: "eldar melar eldar"
 * Test: "i elda maata massa"
 * Test: "i elda máta massa"
 *
 *****************************************************************************/
s([[np, S], [verb, P], [np, O]]) -->
        np(S),
        verb(P),
        np(O)
        /* Kongruenz */.


/*****************************************************************************
 * Beispiel 3: Andere Wortstellung
 *
 * Test: "melar i eldar"
 * Test: "maata i elda"
 * Test: "máta i elda"
 *
 *****************************************************************************/
s([[verb, V], [np, NP]]) -->
        verb(V),
        np(NP)
        /* Kongruenz */.


/*****************************************************************************
 * Beispiel 4: Adjektiv (Positiv)
```

```
 *
 * Test: "isil na calima"
 * Test: "isil ná calima"
 * Test: "isil naa calima"
 *
 ****************************************************************************/
s([[np, NP], [copula, C], [adj, ADJ]]) -->
        np(NP),
        copula(C),
        adjective(ADJ)
        /* Kongruenz: ADJ=positiv */.


/****************************************************************************
 * Beispiel 5: Komparativ
 *
 * Test: "isil na calima la eleni"
 * Test: "isil ná calima la eleni"
 * Test: "isil naa calima laa eleni"
 *
 ****************************************************************************/
s([[np, S], [copula, C], [adj, ADJ], [prep, P], [np, O]]) -->
        np(S),
        copula(C),
        adjective(ADJ),
        preposition(P),
        np(O)
        /* Kongruenz: ADJ=komparativ, P.stem="la" */.


/****************************************************************************
 * Beispiel 6: Superlativ
 *
 * Test: "isil na ancalima"
 * Test: "isil naa ancalima"
 *
 ****************************************************************************/
s([[np, O], [copula, C], [adj, ADJ]]) -->
        np(O),
        copula(C),
        adjective(ADJ)
        /* Kongruenz: ADJ.form=superlative */.

/****************************************************************************
 * Beispiel 7: Wörtliche Rede
 *
 * Test: "eque i eldar: eldar massar eldar"
 * Test: "eque elendil: et earello endorenna utuulien"
 *
 ****************************************************************************/
/*
```

```
s(F) -->
        ‘‘eque’’,
        np,
        ‘‘:’’,
        ergaenzung1.
*/


/*****************************************************************************
 * Beispiel 8: Relativsatz
 *
 * Test: "i elen ya hiruvalye naa ancalima"
 * Test: "i harna ya hirnelye na alta"
 * Test: "i harna ya hirnelye naa alta"
 *
 *****************************************************************************/
/*
s([[np, NP], [e2, E], [vp, VP]]) -->
        np(NP),
        ‘‘ya’’,
        ergaenzung2(E),
        vp(VP).
*/

/*****************************************************************************
 * Beispiel 9: Imperativ
 *
 * Test: "heca"
 *
 *****************************************************************************/
s([verb_imperativ, V]) -->              /* oder [verb ,V]? -> Semantik! */
        verb_imperativ(V).


/*****************************************************************************
 * Beispiel 10: Verb+Subjekt (1. Person)
 *
 * Test: "melon"
 * Test: "hiruvalnye"
 *
 *****************************************************************************/
s([verb_nye, V]) -->
        verb_nye(V).


/*****************************************************************************
 * Beispiel 11: Verb+Subjekt und Objekt
 *
 * Test: "melon massa"
 *
 *****************************************************************************/
```

```
s([[verb_nye, V], [np, NP]]) -->
        verb_nye(V),
        np(NP).


/*****************************************************************************
 * Beispiel 12: Verb+Subjekt (2. Person)
 *
 * Test: "hiruvalye"
 *
 *****************************************************************************/
s([verb_lye, V]) -->
        verb_lye(V).


/*****************************************************************************
 * Beispiel 13: Verb+Subjekt+Objekt
 *
 * Test: "melonlye"
 *
 *****************************************************************************/
s([verb_nye_lye, V]) -->
        verb_nye_lye(V).


/*****************************************************************************
******************************************************************************
***
*** VERBAL-PHRASE - Claudia
***
******************************************************************************
******************************************************************************/
/*
vp(F) --> Verb
vp(F) --> Verb, NP
vp(F) --> Copula, Adjektiv(Positiv)
vp(F) --> Copula, Adjektiv(Komparativ), NP
vp(F) --> Copula, Adjektiv(Superlativ)
*/


/*****************************************************************************
******************************************************************************
***
*** NOMINAL-PHRASE - Sonja, Anna
***
******************************************************************************
******************************************************************************/
np([[det, A],[noun, N]]) -->            /* Test: "i elda" */
        det(A),
        noun(N).
```

```
np([noun, N]) -->                       /* Test: "elda", "eldar" */
        noun(N).


/*****************************************************************************
*****************************************************************************
***
*** DIVERSES
***
*****************************************************************************
*****************************************************************************/

ergaenzung1(F)          -->
        satz(F).

ergaenzung2(F)          -->
        satz(F).

verb_imperativ(V)       -->
        verb(V).
        /* Kongruenz: V = imperativ. Morphologie?! */

verb_nye(V)             -->
        verb(V).
        /* Kongruenz: V hat Endung '-nye' bzw. '-n'. Morphologie! */

verb_lye(V)             -->
        verb(V).
        /* Kongruenz: V hat Endung '-lye'. Morphologie! */

verb_nye_lye(V)         -->
        verb(V).
        /* Kongruenz: V hat Endung '-nye'/'-n' gefolgt von '-lye'.
           Morphologie! */
```

Literatur

[Blackburn et al., 2004] Blackburn, P., Bos, J., and Striegnitz, K. Learn Prolog Now [online]. (2004) [cited 11.5.2004]. Available from: `http://www.coli.uni-sb.de/~kris/prolog-course/html/`.

[Clocksin and Mellish, 2003] Clocksin, W. F. and Mellish, C. S. (2003). *Programming in Prolog*. Springer Verlag, Heidelberg, Germany.

[Derzhanski, 2000] Derzhanski, I. A. TengTeX: TeX Spells for Typesetting in Tengwar [online]. (2000) [cited 11.5.2004]. Available from: `http://www.dcs.ed.ac.uk/misc/local/TolkLang/fonts/TengTeX/`.

[Helge Kåre Fauskanger, 2004] Helge Kåre Fauskanger. Quenya Course [online]. (2004) [cited 17.5.2004]. Available from: `http://www.uib.no/People/hnohf/qcourse.htm`.

[IMDB, 2004] Internet Movie Database [online]. (2004) [cited 22.7.2004]. Available from: `http://www.imdb.com/`.

[Namárië, 2004] Namárië [online]. (2004) [cited 3.7.2004]. Available from: `http://www.uib.no/People/hnohf/namarie.htm`.

[Tolkien, 2000] Tolkien, J. R. R. (2000). *Derr Herr der Ringe*. RM Buch und Medien Vertrieb GmbH, Stuttgart, Germany.