Bachelorarbeit

zur Erlangung des akademischen Grades Bachelor of Arts (B.A.)
an der Fakultät für Linguistik und Literaturwissenschaft
- Universität Bielefeld -

Automatische syntaktische Annotation mit XSLT

Daniel Jettka
Matrikelnummer: 1633522
eMail: djettka@uni-bielefeld.de

Betreuer/Erstgutachter: Dr. Andreas Witt, Universität Tübingen
Zweitgutachter: Prof. Dr. Dieter Metzing, Universität Bielefeld

Ausgabedatum: 08. Februar 2006
Abgabedatum: 18. April 2006

Inhaltsverzeichnis

Einleitung

Die vorliegende Arbeit entstand im Kontext des Projekt-Seminars „Content Management und E-Publishing“, das im Sommersemester 2005 von Andreas Witt und Benjamin Birkenhake an der Universität Bielefeld durchgeführt wurde. Die Veranstaltung mitsamt der daran anschließenden Bachelor-Arbeit bildet das abschließende Modul im Bachelor-Studiengang Linguistik mit dem Profil Texttechnologie. Meine Bachelor-Arbeit ist der Versuch, typisch texttechnologische mit typisch linguistischen (auch computerlinguistischen) Fragestellungen zu verbinden.
Auf der einen Seite steht dabei ein zentrales Instrument der Texttechnologie: die Auszeichnungssprache XML. Sie bietet die Möglichkeit, Texte[1] auf verschiedenen Ebenen mit beliebiger Art von Information zu versehen. Darüber hinaus hat sich im Laufe der letzten zehn Jahre ein breites Spektrum von XML-Erweiterungen entwickelt, die eine komplexe Verarbeitung der Informationen ermöglichen (vgl. Kapitel 1). Dies ist eine gute Anknüpfungsmöglichkeit für die Linguistik. Diese operationalisiert Sprache anhand verschiedener sprachlicher Ebenen, die aufeinander aufbauen und Schnittstellen aufweisen. Mit XML steht ein Instrument zur Verfügung, mit dem Wissen über die verschiedenen Sprachebenen repräsentiert und verknüpft werden kann.
In meiner Arbeit widme ich mich der sprachlichen Ebene der Syntax. Diese kann nicht vollkommen losgelöst von den anderen Ebenen betrachtet werden. Daher wird auch die Morphologie eine Rolle spielen. Die Syntax-Forschung hat bereits eine große Anzahl von Grammatiktheorien zur Beschreibung syntaktischer Strukturen hervorgebracht. Diese können mit Hilfe von Computern implementiert und getestet werden. Dieser Zusammenhang wird im zweiten Kapitel thematisiert.
Die beiden großen Themengebiete XML und Syntax werden im praktischen Teil der Arbeit miteinander in Beziehung gebracht. Hier stelle ich ein XSLT-Stylesheet vor, welches einen Text mit syntaktischen Informationen anreichert. Das Stylesheet, sowie die anderen für die Transformation benötigten Dateien liegen der Arbeit auf einer CD-ROM bei.
Die Basis der syntaktischen Annotation ist ein Vollformen-Lexikon, welches Lexikoneinträge enthält, die mit morpho-syntaktischen Eigenschaften versehen wurden. In verschie-

[1] im weiteren Sinne auch audio-visuelle oder Daten ohne textuellen Charakter

denen Annotationsschritten wird der Ausgangstext „Hamlet“[2] stufenweise mit syntaktischen Informationen ausgezeichnet (Kapitel 3 und 4) Die höchste hier erreichte Auszeichnungsebene umfasst Adjektiv-, Nominal- und Präpositional-Chunks.
Im Fazit geht es darum, die Ergebnisse der Arbeit zusammenzufassen. Hierbei werden insbesondere bestimmte Probleme, die sich bei der Arbeit ergeben haben, zur Sprache kommen.

[2]TEI-annotierte Ausgabe des „Hamlet“ in deutscher Übersetzung von Wilhelm August von Schlegel

Teil I

Theoretische Grundlagen

Kapitel 1

XML, XSLT & Co.

1.1 Linguistische Annotation mit XML

1.1.1 Anforderungen an linguistische Annotationsformate

In der linguistischen Forschung wurde eine große Anzahl verschiedener Annotationsformate entwickelt. Viele, wenn nicht die meisten von ihnen, basieren auf XML. Bemerkenswert ist, dass die einfache Handhabung und Erweiterbarkeit der Auszeichnungssprache XML, die die weite Verbreitung und die extrem reichhaltigen Ressourcen zur Verarbeitung von XML erst ermöglichten, z.T. dazu geführt haben, dass eine kaum überschaubare Anzahl unterschiedlicher Annotationsformate entstanden ist. Die Fülle von unterschiedlichen Formaten ist nicht von vornherein schlecht, bringt sie doch möglicherweise eine große Anzahl sehr unterschiedlicher Forschungsinteressen zum Ausdruck. Jedoch kann dies auch schwerwiegende Probleme mit sich bringen.
Ule/Hinrichs (2004: 224) behandeln in ihrem Artikel die Ziele, die eine linguistisch motivierte Annotation verfolgen sollte. In diesem Zusammenhang erwähnen sie beispielsweise die *Wiederverwendbarkeit* als ein Ziel bei der Durchführung von Annotationsprojekten. Dieses Ziel könnte durch die starke Heterogenität von Annotationsformaten gefährdet werden. Als ein weiteres Ziel sehen Ule/Hinrichs die *Theorieneutralität*, die dem ersten Ziel zu Grunde liegt. Zwar könne eine absolute Theorieneutralität bei Annotationen in der Realität kaum erreicht werden, aber „unvermeidbar fließen in das zu Grunde gelegte Annotationsschema die Grundannahmen einer bestimmten Theorie ein, die eine Verwendung der annotierten Daten unter anderen theoretischen Vorzeichen erschweren“ (Ule/Hinrichs, 2004: 224). Ein gewisses Maß an Theorieneutralität sei jedoch die Voraussetzung für die Wiederverwendbarkeit einer Annotation. Die Wiederverwendbarkeit „ist gewährleistet, solange ein linguistisch annotiertes Korpus so umgewandelt werden kann, dass es ganz oder teilweise die Anforderungen erfüllt, die eine andere zu Grunde liegende Theorie an die lin-

guistische Annotation stellt“ (Ule/Hinrichs, 2004: 224, in Anlehnung an Sailer/Richter, 2001).
Neben diesen eher theoretischen Grundgedanken, gibt es zusätzlich ganz praktische Anforderungen an Annotationen. So sei laut Ule/Hinrichs (2004: 225f.) z.B. darauf zu achten, dass keine Information aus dem Ausgangstext verloren geht. In diesem Kontext geht es u.a. um die Entscheidung über eine Stand-off- oder In-Line-Annotation. Eine In-Line-Annotation führt den Ausgangstext und die Annotation in einem Dokument zusammen, während eine Stand-Off-Annotation beide getrennt voneinander verwaltet.
Bei der Stand-Off-Annotation ist ein Informationsverlust nicht zu befürchten, da der Ausgangstext unverändert bleibt und die Annotation mittels Verweisen auf den Inhalt dieses Texts referenziert. Auf diese Weise „bleibt die ursprüngliche, zu annotierende sprachliche Äußerung eine separate Einheit, auf die die linguistische Annotation nur verweist“ (Ule/Hinrichs, 2004: 231). Diese Art der Annotation erscheint vor allem für Annotationen mit mehreren Schichten (z.B. Annotation verschiedener Sprachebenen) sinnvoll. Da eine Stand-Off-Annotation jedoch recht aufwändig zu verwalten und zu verarbeiten ist (z.B. mit XSLT), bietet sich für kleinere Annotationsprojekte eher das Format der In-Line-Annotation an. Aus diesem Grund ziehe ich in diesem Projekt eine In-Line-Annotation vor. Natürlich werde ich darauf achten, dass möglichst keine Information aus dem Ausgangstext verloren geht.
Um den oben aufgezeigten Anforderungen an linguistische Annotationen gerecht zu werden, werde ich im Folgenden wichtige Annotationsformate, die diese Prämissen verfolgen, vorstellen und aus ihnen Schlüsse für das in dieser Arbeit verwendete Annotationsformat ziehen.

1.1.2 XML-basierte linguistische Annotationsformate

TEI

Die Text Encoding Initiative (TEI) wurde 1987 ins Leben gerufen. Sie entstand als internationale Initiative von Philologen und hatte zum Ziel, digitale Texte unabhängig von bestimmten Betriebssystemen oder Präsentationsformaten mit Informationen anreichern zu können und das Ergebnis in einem dauerhaften Format zu speichern (vgl. Jannidis, 1997). Als Grundlage dieses Formats bot sich die Auszeichnungssprache SGML an. So wurde das Auszeichnungsinventar der verschiedenen TEI-Tag Sets zunächst in SGML-DTDs festgelegt.
Die Struktur der Tag Sets der TEI ist modular aufgebaut. „Sobald feststeht, welche Art von Information ausgezeichnet werden soll, müssen die dazu notwendigen Tags ausgewählt werden“ (Jannidis, 1997). Man unterscheidet in der TEI drei verschiedene Arten von Tag

Sets. Das Core Tag Set bildet die Grundlage für die Struktur eines jeden TEI-Dokuments. Es wird also zwangsläufig immer verwendet. Die Base Tag Sets liefern eine Auswahl von Tag Sets für verschiedene Textsorten, im Fall der vorliegenden Hamlet-Version beispielsweise TEI.drama. Jannidis (1997) nennt noch sieben weitere Base Tag Sets, darunter z.B. TEI.spoken zur Auszeichnung von phonetisch transkribierten Äußerungen oder auch TEI.dictionaries zur Auszeichnung von Wörterbüchern. Für jedes TEI-Dokument kann genau ein Base Tag Set ausgewählt werden.
Zusätzlich können beliebig viele Additional Tag Sets verwendet werden. Diese spezifizieren z.B. die Auszeichnung von Hyperlinks (TEI.linking), Grafiken (TEI.figures) oder auch speziell korpus-linguistischen Merkmalen (TEI.corpus).
Die Spezifikationen der TEI (*Guidelines for Electronic Text Encoding and Interchange*) liegen aktuell in der XML-konformen Version TEI P4[1] vor. Für diese Arbeit interessant sind dabei die Auszeichnungsstandards für linguistische Merkmale von Textbestandteilen. So könnte ein Lexikoneintrag für das bei der syntaktischen Analyse verwendete Lexikon (Kapitel 3.1.3) nach der Spezifikation der standardisierten DTDs der TEI folgendermaßen aussehen:

```
<entry>
    <form>
        <orth>Anschein</orth>
    </form>
    <gramGrp>
        <number>sg</number>
        <gen>m</gen>
        <case>1</case>
        <pos>N</pos>
    </gramGrp>
    <gramGrp>
        <number>sg</number>
        <gen>m</gen>
        <case>3</case>
        <pos>N</pos>
    </gramGrp>
    <gramGrp>
        <number>sg</number>
        <gen>m</gen>
        <case>4</case>
```

[1] http://www.tei-c.org/P4X/; TEI P5 ist in Arbeit

```
        <pos>N</pos>
    </gramGrp>
</entry>
```

Dieser Lexikoneintrag weist allerdings einen entscheidenden Nachteil auf: Zwar ist er in dieser Form gut lesbar und auch verständlich; jedoch bei Einträgen, die eine extreme Ambiguität der morphologischen Merkmale aufweisen (Bsp.: *Gewissen* hätte 70 verschiedene `<gramGrp>`-Elemente), würde diese Lesbarkeit, vor allem aber die Übersichtlichkeit, auf eine harte Probe gestellt. Aus diesem Grund möchte ich im nächsten Abschnitt ein weiteres standardisiertes Annotationsformat vorstellen.

XCES

Der Corpus Encoding Standard (CES) wurde 1996 auf der Grundlage der SGML und den 1994 veröffentlichten Empfehlungen der TEI begründet. Schon früh wurde das verwendete Auszeichnungsformat der Auszeichnungssprache XML angepasst. Hierin sieht Lezius das Erfolgsgeheimnis des CES (XML-konforme Version: XCES). Er meint: „Insbesondere die Fokussierung auf den Datenauszeichnungsstandard XML zu einem Zeitpunkt, zu dem XML noch kaum verbreitet war, hat sich als Schlüssel zum Erfolg erwiesen“ (Lezius, 2002: 21). Die Grundideen des CES bzw. XCES basieren im Wesentlichen auf den folgenden drei Zielen (vgl. Lezius, 2002: 21, in Anlehnung an Ide, 1998):
Erstens könnten durch die Spezifikation spezieller linguistischer Kodierungsstandards möglichst viele bisherige Ansätze abgedeckt werden, gleichzeitig jedoch auch Empfehlungen für zukünftige Ansätze erfolgen. Zweitens könne auf diese Weise eine „Minimierung der Kosten für die Erstellung, Annotation und Verwendung von Korpora durch einen einheitlichen Standard“ (Lezius, 2002: 21) erreicht werden. Desweiteren unterstütze die Initiative eine möglichst hohe Wiederverwendbarkeit der für Korpora entwickelten Werkzeuge. Diese Argumente stehen in engem Zusammenhang mit den bereits in Kapitel 1.1.1 genannten Anforderungen an linguistische Annotationsformate.
Ein Lexikoneintrag, analog zu dem aus dem vorherigen Abschnitt, würde im XCES-Format folgendermaßen aussehen:

```
<tok>
    <orth>Anschein</orth>
    <lex>
        <msd>nn++++</msd>
    </lex>
    <lex>
        <msd>nn++++</msd>
```

```
    </lex>
    <lex>
        <msd>nn++++</msd>
    </lex>
</tok>
```

Die Besonderheit dieser Auszeichnung ist, dass die morpho-syntaktischen Informationen (Inhalt des `<msd>` -Elements) auf Grundlage des Biber Taggers des American National Corpus (ANC)[2] kodiert sind. Dieser orientiert sich speziell an den Anforderungen der englischen Morphologie. In dem obigen Beispiel ist der Code `nn++++` für „singular common nouns“ aufgeführt. Hier fehlt demnach die im Deutschen wichtige Kongruenzinformation für Genus und Kasus.
Desweiteren ist der XCES speziell auf Stand-Off-Annotationen (vgl. Kapitel 1.1.1) ausgerichtet. Mithilfe von Referenzmechanismen (im XCES durch XPointer realisiert) können auf diese Weise separate Annotationen erstellt werden. „Durch Referenzen auf die primäre Korpustextdatei werden dann die Bezüge zwischen den Annotationsebenen hergestellt“ (Lezius, 2002: 23).
Dieses sehe ich allerdings nicht als Hinderungsgrund, das Annotationsformat der syntaktischen Annotation in dieser Arbeit, weitgehend an die Standards des XCES anzupassen. Die im XCES vorgesehenen Referenzmechanismen werden hierbei aber nicht verwendet. Weitere Abweichungen vom XCES ergeben sich vor allem aus den oben genannten Gründen, d.h. der Darstellung deutscher Kongruenzinformation, sowie der Übersichtlichkeit, auf der Ebene der morpho-syntaktischen Information. Ein exemplarischer Lexikoneintrag wird im Kapitel 3.1.3 vorgestellt. Das Tag Set für die syntaktische Annotation selbst wird im Kapitel 3.1.2 „Das Tag Set“ thematisiert.

1.2 Transformation von XML mit XSLT

Dieses Kapitel behandelt die Hintergründe der Extensible Stylesheet Language Transformations (XSLT). Es wird darum gehen, zumindest ansatzweise auf die Funktionsweise der Sprache einzugehen. Dabei kann ich natürlich keine umfassende Befehlsreferenz aufführen. Das Ziel ist vielmehr, das generelle Prinzip von XSLT zu erläutern.

1.2.1 Die Funktionsweise von XSLT

Die Transformationssprache XSLT bildet einen Teil der Extensible Stylesheet Language (XSL). Diese bestand zunächst aus einer Transformationssprache, die für die strukturelle

[2] siehe http://americannationalcorpus.org/FirstRelease/Biber-tags.txt

Transformation von XML-Informationen verantwortlich ist, sowie einer Formatierungssprache, die für den Prozess der Formatierung für verschiedene Ausgabemedien zuständig war. Dieser zweistufige Prozess wurde später in zwei eigenständige Sprachen aufgeteilt: XSLT und XSL-FO (XSL Formatting Objects, offizielle Bezeichnung: XSL) (vgl. Kay, 2004: 21).
Bei der Entwicklung der Transformationssprache XSLT wurde deutlich, dass sich die Syntax derjenigen Sprache in XSLT, die der Navigation in einem, durch ein Baum-Modell repräsentierten, XML-Dokument dient, mit der Sprache XPointer überschneidet. Um dem Entstehen zweier Sprachen mit ähnlicher Syntax entgegenzuwirken, entschied das W3C eine Sprache zu entwickeln, die beiden Zwecken dient: die XPath Language. Sie dient seither als Untersprache von XSLT (vgl. Kay, 2004: 21f.).

XSLT ist eine deklarative Sprache. Das bedeutet, dass die Programmierung in XSLT keine prozeduralen Aspekte berücksichtigt, sondern lediglich eine gewünschte Transformation anhand von Regeln (Templates) festgelegt wird. Die Anwendung dieser Regeln bzw. der tatsächliche Ablauf der Transformation wird im Wesentlichen durch den eingesetzten XSLT-Prozessor (s.u.) determiniert. Die Templates können in einem Stylesheet oder mehreren (sog. modularisierten) Stylesheets definiert sein. Die Struktur der Stylesheets ist durch den XSLT-Namensraum festgelegt. XSLT-Stylesheets liegen daher in Form valider XML-Instanzen vor. Durch die Verwendung von Namensräumen können in einem Stylesheet Konstrukte verschiedener XML-basierter Sprachen verwendet werden.
Eine Transformation von XML mit Hilfe von XSLT verläuft in zwei Schritten: der erste Schritt ist eine strukturelle Transformation. In dieser werden das zu transformierende XML-Dokument und das XSLT-Stylesheet durch den XSLT-Prozessor in interne Baum-Repräsentationen überführt. Diese wiederum dienen als Basis der Erstellung einer Baum-Repräsentation des gewünschten Ausgabe-Dokuments. Die Repräsentation in Form eines Baumes ist dem Konzept des Document Object Model (DOM)[3] des W3C ähnlich. Im zweiten Schritt wird die Output-Struktur in das im Stylesheet festgelegte Ausgabeformat formatiert (Kay, 2004: 5).
Das Datenmodell für die Repräsentation von XML als Baum wird von XSLT, XPath und XQuery gleichermaßen verwendet. Dies ermöglicht einen freien Datenaustausch zwischen den drei Sprachen (Kay, 2004: 45). Die Verwendung von XPath spielt eine wichtige Rolle in XSLT. Mit Hilfe von XPath-Ausdrücken können Knoten in der Baum-Repräsentation referenziert werden. Auf diese Weise ist es möglich, gezielte Verarbeitungsanweisungen für Knotenmengen zu formulieren. Da XPath desweiteren eine Vielzahl von Funktionen bietet, ist sie ein sehr ausdrucksmächtiger Bestandteil von XSLT. Dieses wird auch deutlich,

[3] siehe http://www.w3.org/DOM/

wenn man sich das in dieser Arbeit entwickelte Stylesheet zur syntaktischen Annotation etwas näher ansieht.
Aufgrund der Komplexität von XSLT kann ich hier nicht auf einzelne Bestandteile der Sprache eingehen. Dieses erfolgt z.T. in der Dokumentation der syntaktischen Annotation im zweiten Teil der Arbeit.
Hinweisen möchte ich an dieser Stelle jedoch noch auf die Möglichkeit, unstrukturierten Text mit XSLT zu verarbeiten. Witt (1999: 109) weist auf dieses Verfahren im Kontext von SGML und DSSSL hin. Dieses läßt sich auf XML und XSLT übertragen. Bei diesem Verfahren „wird die den Text enthaltende Datei an eine allgemeine Entität gebunden und [...] zwischen den öffnenden und schließenden Elementen [des Wurzelelements eines XML-Dokuments] aufgerufen“ (ebd.). Auf diese Weise ist der Text in ein XML-Dokument eingebunden und läßt sich mit XSLT verarbeiten. Vor allem die Implementierung regulärer Ausdrücke in XSLT 2.0 macht dieses Verfahren sehr interessant. Sie ermöglichen eine sehr detaillierte Verarbeitung unstrukturierten Texts. Durch kleine Änderungen am Stylesheet wäre dies auch für das hier vorgestellte XSLT-Programm zu verwirklichen. So könnten noch nicht strukturierte Texte syntaktisch annotiert werden.

1.2.2 XSLT-Prozessoren und -Editoren

Für die Verarbeitung von XSLT-Stylesheets stehen eine Reihe von Prozessoren zur Verfügung. Ihre Aufgabe besteht, wie oben bereits erwähnt, darin, ein XSLT-Stylesheet auf ein XML-Quelldokument anzuwenden und ein Ergebnisdokument zu erzeugen (vgl. Kay, 2004: 8).
Vor allem für XSLT 1.0 gibt es eine große Auswahl an Prozessoren. Beispiele sind: MSXML3, MSXML4, Xalan-J und Saxon bis Version 6.5.5. Die Version 2.0 von XSLT wird meines Wissens bisher leider nur von den neueren Versionen des XSLT-Prozessors Saxon unterstützt. Die Version Saxon-B 8.7 ist Open Source-Software[4].
Auch spezielle XML-Editoren mit integrierten XSLT-Modulen sind verfügbar. Allerdings gibt es sie normalerweise entweder als kommerzielle Ausgaben oder als 30-Tage-Test-Versionen. Der von mir verwendete Editor ist der Oxygen 6.2[5], der mittlerweile weiterentwickelt wurde (Oxygen 7.1). Dieser hat verschiedene integrierte XSLT-Prozessoren, von denen ich aufgrund der Verwendung von XSLT 2.0 den Saxon-B 8.7 benutzt habe. Eine sehr hilfreiche Komponente des Oxygen ist ein integriertes XPath-Modul, über das Ausdrücke direkt im Editor aufgerufen und auf deren Grundlage Knotenlisten erstellt werden können. Weitere XML-Editoren sind das Stylus Studio® 2006 XML, sowie der XMLSpy® 2006. Sie verfügen ebenfalls über diverse integrierte XSLT-Prozessoren.

[4] erhältlich unter http://prdownloads.sourceforge.net/saxon/saxonb8-7j.zip
[5] http://www.oxygenxml.com/download.html

Kapitel 2

Syntax & Parsing

Im vorangehenden Kapitel wurden die Werkzeuge für die syntaktische Analyse von Texten mit XSLT vorgestellt. Der formale Rahmen der syntaktischen Analyse ist demnach durch die XML-basierte Architektur, in deren Zentrum die Programmiersprache XSLT steht, festgelegt. Es ist nun notwendig, bestimmte linguistische Grundannahmen für die syntaktische Analyse zu treffen. Die entstehende theoretische Basis wird Annahmen beinhalten, wie Informationen, die der syntaktischen Analyse zugrundeliegen, repräsentiert werden können (Kapitel 2.1). Außerdem wird festzulegen sein, wie diese Annahmen durch Regeln erfasst und verarbeitet werden können (Kapitel 2.2).

2.1 Syntaxtheorie

2.1.1 Morphologische Information als Input für die Syntax

Dieser Abschnitt beschäftigt sich mit der Frage, welchen Input eine einigermaßen erfolgversprechende syntaktische Analyse benötigt.
Man kann zunächst davon ausgehen, dass dieser Input nicht nur aus den zu verarbeitenden Wörtern selbst besteht, sondern dass mit diesen von vornherein bestimmte Informationen verbunden sind, durch die sie zu anderen Wörtern in Beziehung gesetzt werden können. Bereits an dieser Stelle machen verschiedene Theorietraditionen sehr unterschiedliche Annahmen. Die Kategorialgrammatik beispielsweise verbindet Wörter mit Bedeutungskategorien, die diese hinsichtlich der Kombinierbarkeit mit anderen Wörtern bzw. Bedeutungskategorien charakterisieren[1]. Hier werden Wörter demnach nicht mit der traditionellen Kategorie der Wortart beschrieben. Das Konzept der Bedeutungskategorie selbst umfasst bestimmte Regeln der Kombinierbarkeit der einzelnen Einheiten eines Sat-

[1] Eine gute Einführung in das Feld der Kategorialgrammatiken bietet:
Wood, Mary McGee (1993): *Categorial Grammars*. London/New York: Routledge.

zes. Diese Regeln befinden sich also schon in den einzelnen Einheiten und müssen nicht extern formuliert werden. Dieses Vorgehen steht in der Tradition der lexikalisierten Grammatiken.
Die Konstituentenstrukturgrammatiken hingegen verwenden normalerweise die traditionellen Wortarten als Hauptinformation zur Bildung von Regeln. Auf dieser Basis werden syntaktische Regeln der Form $NP \rightarrow Det\ N$ gebildet. Die Konstituentenstrukturgrammatiken, speziell aber „Kontextfreie Grammatiken [...] sind – zumindest innerhalb der Tradition der konstituentenstrukturorientierten Grammatikmodelle – nach wie vor das Basisinstrument für syntaktische Analysen, wenngleich sie heutzutage nur noch selten in reiner Form verwendet werden“ (Langer, 2004: 234).
Auch in dieser Arbeit bilden kontextfreie Phrasenstrukturregeln, die auf der Information der Wortarten aufbauen, die Basis der syntaktischen Analyse. Diese Regeln sind allerdings in der Syntax nur beschränkt einsetzbar, da mit der reinen Information über die Wortart eines Wortes nicht unbedingt entscheidbar ist, ob eine Kombination mehrerer Wörter zu einer syntaktischen Konstituente wohlgeformt ist oder nicht. Hierzu bedarf es weiterer morpho-syntaktischer Informationen, wie z.B. Kongruenzinformationen. Inwieweit diese mit kontextfreien Grammatiken (KFG) zu repräsentieren und verarbeiten sind, wird im folgenden Abschnitt diskutiert. Festhalten läßt sich jedoch, dass die Information über die Wortart eines Wortes in vielen Fällen nicht ausreicht, eine zuverlässige syntaktische Analyse unter dem Aspekt der Wohlgeformtheit zu ermöglichen.

2.1.2 KFG zur Beschreibung natürlicher Sprache?

Kontextfreie Grammatiken werden sehr häufig als Basis zur Erstellung syntaktischer Regeln, die der syntaktischen Analyse natürlicher Sprache (zumeist geschriebene Sprache) dienen, verwendet. Genügen kontextfreie Grammatiken jedoch tatsächlich den Anforderungen, natürliche Sprache adäquat zu beschreiben?
Ein ganz praktisches Problem taucht bei Sprachen mit einer starken morphologischen Struktur auf[2]. Die Schwierigkeit läßt sich am Beispiel des Deutschen gut verdeutlichen. Bei der syntaktischen Analyse des Deutschen kommt es nicht nur auf die Kombination bestimmter Wortarten an. Um entscheiden zu können, ob es sich z.B. bei der Kombination eines Artikels mit einem Nomen um eine wohlgeformte Nominalphrase handelt, benötigt man zusätzliche Informationen über die jeweiligen Wortformen der beteiligten Wörter. So kann man feststellen, dass es sich bei der Kombination *das Männer* nicht um eine wohlgeformte Nominalphrase handelt, obwohl die Regel $NP \rightarrow Det\ N$ nicht verletzt wird. Die Kongruenzinformationen (Numerus, Genus, Kasus) der beiden Wörter, sind nicht mitein-

[2]solche Sprachen werden auch synthetische Sprachen genannt

ander kombinierbar.
Zusätzlich zur Wortart sollten die Phrasenstrukturregeln für Nominalphrasen folglich Informationen über die Kongruenz von Wörtern enthalten. Hierdurch ergäbe sich die Notwendigkeit, eine sehr große Anzahl an kategorialen Symbolen zu bilden. Allein zur korrekten Behandlung der Nominalphrasenkongruenz würde man über 50 nicht-terminale Symbole[3] benötigen, um Numerus-, Genus- und Kasusinformationen unterzubringen. Wenn man nun noch die Subjekt-Verb-Kongruenz, Subkategorisierungsinformationen von Verben und Rektion von Präpositionen beachtet, wird deutlich, dass auf diese Weise eine kaum überschaubare Anzahl kategorialer Symbole entstünde. Dieses wäre zwar unpraktisch, jedoch theoretisch unproblematisch. Das bedeutet, dass Phänomene wie Kongruenz von kontextfreien Grammatiken durchaus zu behandeln wären. Allerdings sind dabei nach Langer (2004) einige generelle Probleme zu erkennen:
Erstens werden „Generalisierungen wie ‚Innerhalb der NP herrscht Kongruenz bzgl. Kasus, Numerus und Genus'" (Langer, 2004: 237) nicht explizit gefasst, sondern lediglich notationell behandelt. Dadurch „zersplittert eine Generalisierung in eine Aufzählung einzelner Fälle" (ebd.). Zweitens, und dieses Problem hängt eindeutig mit dem ersten zusammen, gibt es in kontextfreien Grammatiken keine Möglichkeit, Parameter wie Numerus usw. angemessen zu behandeln. Sie werden einfach als Teil des Namens eines kategorialen Symbols erfasst: „die Symbole N_{mn} und N_{ma} sind sich auf formaler Ebene keinen Deut ähnlicher als die Symbole N_{mn} und VP" (ebd.).
Im Gegensatz zu den o.g. Problemen, die mit einigem Aufwand noch durch kontextfreie Grammatiken beschreibbar wären, gibt es jedoch auch Merkmale bestimmter Sprachen, die die Beschreibungsadäquatheit kontextfreier Grammatiken überschreiten. So zeigen Bresnan et al. (1982), dass die „cross-serial dependencies" im Niederländischen mit rein kontextfreien Regeln nicht adäquat beschrieben werden können. Sie postulieren, unter der Prämisse, dass ihre eigenen Annahmen über die Struktur des Niederländischen korrekt sind: „then there is no context-free grammar that can assign the correct structural descriptions to Dutch sentences" (Bresnan et al., 1982: 621). Weitere Beispiele für Sprachen, die nicht von kontextfreien Grammatiken generiert werden können, liefert Müller (1994: 23f.). Er benennt das Schweizer-Deutsch und Bambara (eine Sprache im Senegal) als ebensolche Sprachen.
Für das Deutsche, und die Annahmen über die Syntax des Deutschen speziell in dieser Arbeit, folgt, dass die Grundlage der Beschreibung durchaus durch eine kontextfreie Grammatik festgelegt werden kann (siehe Kapitel 4). Trotzdem gibt es bestimmte Ei-

[3] $2 * 2 * 3 * 4 = 48$ (2 für Artikel und Nomen; 2 für Singular und Plural; 3 für maskulin, feminin, neutrum; 4 für Nominativ, etc. + die Bezeichnungen für Nominalphrasen, die wiederum Numerus- und Kasusinformationen benötigen)

genschaften des Deutschen (z.B. Kongruenz), die durch andere Grammatikformalismen sehr viel anschaulicher behandelt werden können. Dieses wird im nachfolgenden Abschnitt deutlich.

2.1.3 Unifikationsgrammatiken

Die Unifikationsgrammatiken[4] bilden einen Grammatiktypus, der eine Erweiterung der kontextfreien Grammatiken darstellt. Die Grundidee der Unifikationsgrammatiken sind Modellierungen linguistischer Objekte anhand von „Merkmals-Wert-Spezifikationen oder auch Attribut-Wert-Spezifikationen“ (Witt/Müller, 2002: 425). Auf diese Weise können Wörter und andere linguistische Einheiten mithilfe von Merkmalsstrukturen mathematisch formal beschrieben werden. Eine gängige Repräsentation von Merkmalsstrukturen stellen die sogenannten Attribut-Wert-Matrizen (AWM) dar.
Die wichtigste Operation innerhalb der Unifikationsgrammatiken ist die Unifikation[5], nach der sie benannt sind. Vereinfacht ausgedrückt sind mit Hilfe der Unifikationsoperation AWMn in Abhängigkeit der in ihnen enthaltenen Attribute und Werte kombinierbar. Durch diese Operation sind syntagmatische Relationen[6] wie die Kongruenz sehr gut zu verarbeiten. In Unifikationsgrammatiken ist es möglich, Generalisierungen über syntagmatische Relationen wie die Kongruenz innerhalb von Nominalphrasen formal zu beschreiben. Auch Parameter wie Numerus, Genus und Kasus werden nicht bloß als Teil des Namens eines kategorialen Symbols, sondern formal erfasst. Das Prinizip der Unifikation hat die Behandlung der Nominalphrasenkongruenz in dieser Arbeit stark beeinflusst und wird so in Kapitel 4.2 erneut thematisiert.

2.2 Parsing natürlicher Sprache

Die Verarbeitung morpho-syntaktischer Informationen, die, wie deutlich wurde, durch unterschiedliche Formalismen darstellbar sind, erfolgt im Allgemeinen durch einen Parser. Der Begriff Parsing ist nicht nur in der Computerlinguistik, sondern insbesondere auch im benachbarten Gebiet der Informatik, vor allem im Zusammenhang mit Programmiersprachen, weit verbreitet. Auch in diesem Bereich geht es um die Analyse syntaktischer

[4] wichtigste Vertreter (nach Langer, 2004: 238): **Generalized Phrase Structure Grammar** (GPSG, *Gazdar et al.* 1985), **Lexical Functional Grammar** (LFG, *Bresnan*, 1982), **PATR-II** (*Shieber* 1986)), **Head-Driven Phrase Structure Grammar** (HPSG, *Pollard/Sag* 1994).

[5] gute formale Definitionen sind zu finden u.a. bei: Kolb (2004), Witt/Müller (2002) und Jurafsky/Martin (2000: 395-446)

[6] Eisenberg (2004: 32ff) nennt für das Deutsche vier Typen syntagmatischer Relationen: Rektion, Identität, Kongruenz und Positionsbezug.

Strukturen. Desweiteren kann Parsing ebenso „als Teil des menschlichen Sprachverstehensprozesses aufgefaßt werden“ (Naumann/Langer, 1994: 5). Zur Herkunft des Wortes Parsing gibt es unterschiedliche Hypothesen:
Naumann/Langer meinen z.B., der Begriff leite sich aus der lateinischen Bezeichnung für die Wortarten, den „partes orationes“, ab. Den Ideen der Grammatiktradition des 19. Jahrhundert zufolge beinhalte die Analyse grammatischer Zusammenhänge vorrangig die Wortartenbestimmung (vgl. Naumann/Langer, 1994: 3). Die reine Wortartenbestimmung werde heutzutage jedoch eher mit den Begriffen „Tagging (Wortarten-Tagging, Part-of-speech tagging [...])“ (Langer, 2004: 252) gekennzeichnet. Parsing bezeichnet dagegen „heute eher solche Prozesse, die substantiell über das bloße Annotieren eines Textes mit Wortarten hinausgehen und die die grammatische Struktur einer Äußerung festlegen“ (ebd.). Hier steht der linguistisch motivierte Gebrauch des Begriffs Parsing im Vordergrund. Pulman definiert den Begriff Parsing aus etwas formalerer Sicht wie folgt:

> „'Parsing' is the term used to describe the process of automatically building syntactic analyses of a sentence in terms of a given grammar and lexicon“ (Pulman, 1991: 1)

Unter diesen Begriff würde die syntaktische Verarbeitung in dieser Arbeit sicherlich fallen. Parser werden in der Linguistik allerdings nicht nur im Bereich der Syntax entwickelt und angewendet, sondern auch für andere sprachliche Ebenen, wie Phonologie, Morphologie und Semantik (vgl. Naumann/Langer, 1994: 4). In dieser Arbeit geht es aber natürlich um Parser, die der syntaktischen Analyse dienen.
Parser lassen sich anhand verschiedener Vorgehensweisen beim Parsing-Prozess klassifizieren.

2.2.1 Verschiedene Parsingalgorithmen

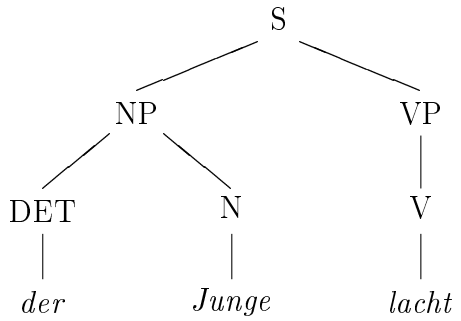
Die gängigsten, in fast jeder Einführung zum Parsing zu findenden, Parsingalgorithmen werden nach ihrer Verarbeitungsrichtung und Analyserichtung unterschieden. Am besten lassen sich die unterschiedlichen Strategien anhand einer einfachen kontextfreien Grammatik verdeutlichen. Gegeben sei also folgende KFG:

$$G = \langle \{S, NP,\ VP,\ DET,\ N,\ V\}, \{der,\ Junge,\ lacht\},$$
$$\{S \rightarrow NP\ VP,\ NP \rightarrow DET\ N,\ VP \rightarrow V,\ DET \rightarrow der,$$
$$N \rightarrow Junge,\ V \rightarrow lacht\},\ S \rangle$$

Die von dieser Grammatik erzeugte Sprache L(G) ist diese hier:

$$L(G) = \{der\ Junge\ lacht\}$$

Die Ableitung dieser Sprache läßt sich sehr übersichtlich mit Hilfe eines Ableitungsbaums visualisieren. Dieser Baum ermöglicht ein besseres Verständnis der vorgestellten Kategorien des Parsing.



Verarbeitungsrichtung

Es lassen sich generell zwei Arten von Verarbeitungsrichtungen bei Parsern unterscheiden. Bei einer unidirektionalen Verarbeitung bearbeitet ein Parser die Eingabe in genau einer Richtung. Die hierbei am häufigsten angewandte Richtung wird als Links-rechts-Verarbeitung bezeichnet (vgl. Langer, 2004: 254). Eine treffendere Bezeichnung wäre allerdings **Vom-Anfang-zum-Ende-Verarbeitung**. In Schriftsystemen, die von rechts nach links ausgerichtet sind, würde ein Parser, der von rechts nach links arbeitet, dem Prinzip der Links-Rechts-Verarbeitung in europäischen Schriften unterliegen. Daher wäre der allgemeiner gefasste Begriff Vom-Anfang-zum-Ende passender. Die verschiedenen Verarbeitungsrichtungen werde ich erst im nächsten Abschnitt auf die Beispielgrammatik beziehen, um den Faktor der Analyserichtung miteinbeziehen zu können.
Die andere unidirektionale Verarbeitungsrichtung ist die Rechts-links-Verarbeitung, oder wieder besser: **Vom-Ende-zum-Anfang-Verarbeitung**. In bidirektionalen Verarbeitungen sind die beiden Verarbeitungsrichtungen durchaus auch miteinander kombinierbar. „So wird z.B. beim sogenannten **Head-Corner-Parsing** (vgl. *Bouma und van Noord* 1993) von den lexikalischen Köpfen eines Satzes (d.h. in der Regel vom finiten Verb) ausgegangen und anschließend (bidirektional) nach links und rechts weiterverarbeitet“ (Langer, 2004: 254).

Analyserichtung

Als mögliche Analyserichtungen von Parsern werden in der Regel die Top-down- und die Bottom-up-Verarbeitung unterschieden. Diese sind nicht die einzigen: „Es gibt zwei extreme Möglichkeiten [top-down/bottom-up], den Suchraum zu spezifizieren (und viele

Varianten, die dazwischen liegen)“ (Russell/Norvig, 2003: 968). Allerdings sind die beiden hier vorgestellten die grundlegenden Analyserichtungen.
„A **top-down** parser searches for a parse tree by trying to build from the root node S down to the leaves“ (Jurafsky/Martin, 2000: 360). Dieses Verfahren kann als hypothesengetrieben oder „zielgesteuert“ (Naumann/Langer, 1994: 23) bezeichnet werden. Um zu einem Ergebnis zu kommen, stellt der Parser, anhand der ihm zur Verfügung stehenden Regeln, Hypothesen über die Struktur eines Satzes auf und vergleicht diese mit der Eingabe. Normalerweise erfolgt dies in der Verarbeitungsrichtung Vom-Anfang-zum-Ende eines Satzes. Im obigen Beispiel würde ein Parser folglich beim Startsymbol S mit der Analyse beginnen. Die einzige Hypothese, die als erstes zur inneren Struktur von S aufgestellt werden kann, ist die Anwendung der Regel $S \rightarrow NP\ VP$. Da die Verarbeitung Vom-Anfang-zum-Ende verläuft, würde danach versucht, eine Regel für NP zu finden. Diese ist in Form $NP \rightarrow DET\ N$ vorhanden. Dieser Prozess läuft weiter bis eine passende Analyse für die Eingabekette gefunden wurde oder herauskommt, dass die Regeln hierauf nicht anwendbar sind.
Eine andere, wohl noch häufiger verwendete Analyserichtung wird mit **bottom-up** bezeichnet. Anders als bei der Top-down-Verarbeitung „wird bei einer Bottom-up-Verarbeitung von der Eingabekette selbst ausgegangen, und die Anwendung der Regeln erfolgt als **Reduktion**, d.h. die Elemente der rechten Regelseite werden durch das Symbol der linken Regelseite ersetzt bis eine vollständige Reduktion der gesamten Eingabekette auf das Startsymbol S erzielt wurde“ (Langer, 2004: 254). Dieses Vorgehen kann man daher auch als daten-getrieben oder „datengesteuert“ (Naumann/Langer, 1994: 23) bezeichnen. Auch hier wird meist die Verarbeitungsrichtung Vom-Anfang-zum-Ende angewendet. In dem oben genannten Beispiel würde der Parser bei dem lexikalischen Symbol *der* beginnen und eine Regel suchen, die auf der rechten Regelseite dieses Symbol enthält ($DET \rightarrow der$). Es wird also von den Blättern des Baumes ausgegangen. Solange die Wurzel des Baumes (S) nicht erreicht ist, wird nach weiteren passenden Regeln gesucht. Diese Regeln werden quasi rückwärts, also von der rechten Regelseite ausgehend, angewendet.

2.2.2 Probleme beim Parsing natürlicher Sprache

Nach Langer (2004: 256ff) sind bei dem Versuch, geeignete Paser für die Analyse natürlicher Sprache zu konstruieren, vor allem drei Probleme besonders hervorzuheben. Neben den Fragen, wie ein möglichst großer Ausschnitt einer Sprache abgedeckt und ein Parser effizient implementiert werden kann, geht es um das Problem der Auflösung von syntaktischen Ambiguitäten. Diesem Problem möchte ich im Folgenden etwas genauer nachgehen. Natürliche Sprache ist sehr reich an Ambiguitäten. Das verdeutlicht folgendes Beispiel in Langer (2004: 257):

> *Hinter dem Betrug werden die gleichen Täter vermutet, die während der vergangenen Tage in Griechenland gefälschte Banknoten in Umlauf brachten.*

Nach Langer liefere das im Kontext des PARGRAM-Projekts (Kuhn/Rohrer, 1997) implementierte Parsing-System für diesen Satz 92 verschiedene Lesarten. Das ist im Besonderen auf ein unter dem Namen PP-Attachment bekanntes Phänomen zurückzuführen. Oftmals ist auch mit umfassenden syntaktischen Informationen nicht eindeutig entscheidbar, welcher syntaktischen Konstituente (*NP*, *VP*) eine Präpositionalphrase zugeordnet werden soll. „PP-Zuordnung ist ein typisch computerlinguistisches Problem, weil zu seiner Lösung komplexes semantisches Wissen erforderlich ist, das in keinem sprachverarbeitenden System zur Verfügung steht“ (Mehl/Langer/Volk, 1998: 97). Daher setzen Mehl/Langer/Volk auf statistische Verfahren bei der Zuordnung von Präpositionalphrasen. Ihnen zufolge seien gute Ergebnisse zu erzielen, wenn man „die quantitativ ermittelte Wahrscheinlichkeit heranzieht, daß ein bestimmtes Lexem eine PP mit einer bestimmten Präposition (eventuell auch noch einer bestimmten Inhaltsklasse) nach sich zieht“ (Mehl/Langer/Volk, 1998: 98f.). Dieses Verfahren dürfte allerdings mit nicht zu unterschätzendem Aufwand verbunden sein.
Unter anderem, um Ambiguitäten natürlicher Sprache adäquat handhaben zu können, entwickelte sich eine vom vollständigen Parsing eines Satzes abrückende Parsing-Strategie: das Chunk-Parsing.

2.2.3 Chunk-Parsing

Das Chunk-Parsing, das „auch partielles oder shallow Parsing“ (Halama, 2004: 228) bzw. „Text-Chunking“ (Bender, 2002: 17) genannt wird, weist einige grundlegende Unterschiede zum vollständigen Parsing eines Satzes auf. Die Basis für die theoretische Fundierung des Chunk-Begriffs bilden psycholinguistische Beobachtungen. Abney führt diese folgendermaßen ein:

> „I begin with an intuition: when I read a sentence, I read it a chunk at a time. For example, the previous sentence breaks up something like this:
>
> (1) [I begin] [with an intuition]: [when I read] [a sentence], [I read it] [a chunk] [at a time]“
>
> (Abney, 1991: 257)

Abney verweist auf die „psychological evidence for the existence of chunks“ (Abney, 1991: 257) und führt in diesem Zusammenhang die Arbeiten von Gee/Grosjean[7] an.

[7] Gee, J./Grosjean, F. (1983): *Performance Structures: A Psycholinguistic and Linguistic Appraisal.* In: Cognitive Psychology, Vol. 15, S. 411-458.

Die Frage, die für diese Arbeit aber viel bedeutsamer ist, ist die nach der Struktur von Chunks, d.h. inwieweit Annahmen über Chunks für die syntaktische Analyse operationalisierbar sind. Das Chunk-Parsing bietet in diesem Bereich deutliche Vorteile gegenüber dem vollständigen Parsing von Sätzen. Bestimmte Probleme von vollständigen Parsern, die häufig zu Ungenauigkeit (z.B. durch Ambiguitäten) und fehlender Performanz führen, werden beim Chunk-Parsing umgangen. Da Chunk-Parser „einerseits nur lokale Abhängigkeiten erkennen, andererseits keine rekursiven Strukturen aufbauen“ (Halama, 2004: 228), haben sie auch eine höhere Genauigkeit und Performanz. Je nach Erkenntnisinteresse kann das Ergebnis des Chunk-Parsings selbst im Mittelpunkt stehen (z.B. für Information Retrieval), oder es fungiert als eine „standard technique to efficiently and reliably pre-structure language data for further linguistic annotation“ (Hinrichs et al., 2002: o.A.), d.h. z.B. tiefergehenden syntaktischen Analysen.
Die Struktur von Chunks kann wie folgt definiert werden: „Chunks are defined as non-recursive continuous kernels of phrases. This means that chunks may contain chunks of other categories but that they may not contain chunks of the same category“ (Müller, 2002: 1f.). Aus dieser Definition folgt, dass viele syntaktische Ambiguitäten, wie sie z.B. beim PP-Attachment entstehen, nicht behandelt bzw. aufgelöst werden müssen. So werden Schwachstellen im Parsing-Prozess, die auf syntaktische Ambiguitäten der beschriebenen Form zurückzuführen sind, vermieden. Das Chunk-Parsing fokussiert eindeutig die Korrektheit des Parsingergebnisses und nicht die Vollständigkeit. Hieraus ergeben sich natürlich auch Nachteile. Langer merkt hierzu an: „Solche Programme, die lediglich eine *partielle* Analyse leisten können, sind allerdings für Anwendungen wie Fehlerdetektion und -korrektur ungeeignet, weil die grammatischen Komponenten nicht hinreichend präzise sind, um zwischen wohlgeformter und nicht wohl-geformter Eingabe [auf Satzebene] zu unterscheiden“ (Langer, 2001: 53).

Beim Chunk-Parsing sind nach Müller (2002: 3) fünf Haupttypen von Chunks zu differenzieren:

- verb chunks
- noun chunks
- adjective chunks
- adverb chunks
- prepositional chunks

Aus der Definition, dass Chunks keine rekursiven Strukturen aufbauen folgt, dass ein Chunk keine Chunks vom selben Typ enthalten darf. Das schließt beispielsweise die Anbindung von Präpositional-Chunks an Nominal-Chunks (Stichwort: PP-Attachment) aus, da ansonsten ein Nominal-Chunk einen weiteren enthielte.

Da ich mich nur z.T. an der bei Müller angenommenen inneren Struktur der Chunks orientiere, werde ich sie nicht im Einzelnen vorstellen. Die interne Struktur der Chunks wird im nächsten Kapitel deutlich werden. Dort geht es um die Implementierung des Parsing-Prozesses, der durch die, in den vorangehenden Abschnitten vorgestellten, Kategorien folgendermaßen charakterisiert werden kann: Es handelt sich um einen Vom-Anfang-zum-Ende-, bottom-up-verarbeitenden Chunk-Parsing-Algorithmus. Dieser umfasst mehrere Stufen, die jeweils die Eingabe für die nächsthöhere Analysestufe bilden.

Teil II

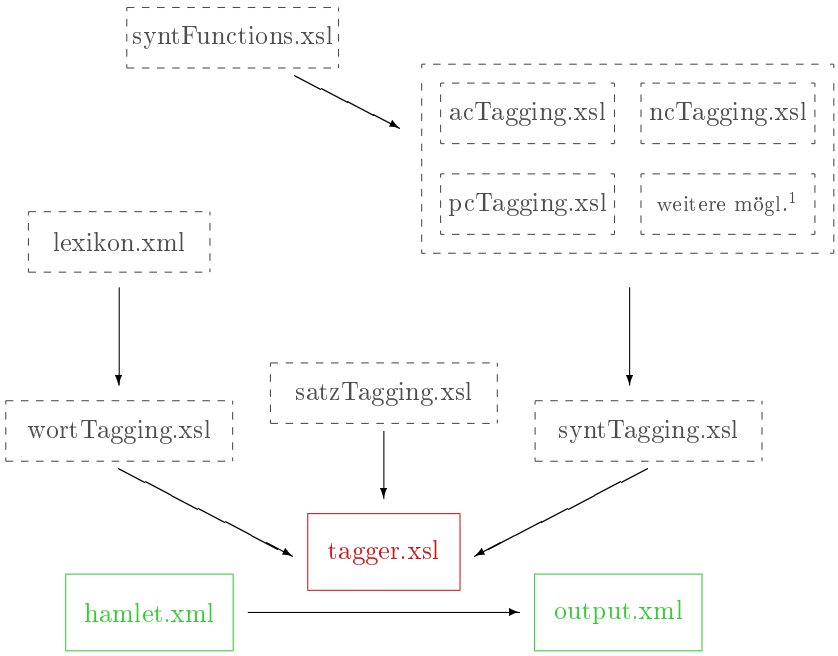
XSLT-Stylesheet zur syntaktischen Annotation

Kapitel 3

Vorverarbeitung

3.1 Die Architektur

Die Architektur des XSLT-Programms zur automatischen syntaktischen Annotation kann durch folgendes Schaubild dargestellt werden:



Die Basis der Transformation bilden der Ausgangstext, der in Form des TEI-Dokuments „hamlet.xml“ vorliegt, sowie das XSLT-Stylesheet „tagger.xsl“, das die einzelnen Schritte der Annotation beinhaltet. Diese einzelnen Schritte sind in weiteren Dateien, sogenannten Stylesheet-Modulen (vgl. Kay, 2004: 174), ausgelagert und werden mithilfe des XSLT-

[1] an dieser Stelle könnten weitere Schritte für die syntaktische Annotation hinzugefügt werden

Elements `<xsl:include>` in das Stylesheet „tagger.xsl“ eingebunden. Auf diese Weise können durch die Angabe eines Stylesheet-Parameters (namens „*ebene*“) beim Aufruf der XSLT-Transformation auch die Vorstufen der syntaktischen Annotation (Wort- und Satzannotation) als Output der Transformation festgelegt werden. Die Angabe dieses Parameters ist optional. Möchte man eine Vorstufe der syntaktischen Annotation als Output der Transformation erhalten, so ist für eine Annotation auf Wortebene für den oben genannten Parameter der Wert „*wortebene*“, für eine Annotation auf Satzebene „*satzebene*“ anzugeben.
Zusätzlich ist es möglich, die einzelnen Stufen der syntaktischen Annotation, also die verschiedenen Chunk-Typen, als Output zu erhalten. Hierzu können dem Parameter die folgenden Werte zugewiesen werden: „*AC-Ebene*“, „*NC-Ebene*“, „*PC-Ebene*“. Allerdings sollte man beachten, dass die Annotation der Nominal-Chunks in der hier gewählten Architektur die Annotation von Adjektiv-Chunks beinhaltet. Ebenso verwendet die automatische Annotation von Präpositional-Chunks die AC- und NC-Ebenen als Eingabe. Sofern dem Stylesheet-Parameter beim Aufruf der Transformation kein Wert zugewiesen wird, erfolgt die komplette syntaktische Annotation für alle behandelten Chunk-Typen.
In den folgenden Abschnitten werden die einzelnen Komponenten der automatischen syntaktischen Annotation erläutert.

3.1.1 Der zu annotierende Text: „hamlet.xml“

Die vorliegende Arbeit entstand im Rahmen des Seminars „Content Management und E-Publishing“ (im Sommersemester 2005), in dem die Arbeit an einem Korpus verschiedener Hamlet-Ausgaben im Vordergrund stand. Das Korpus stammt aus dem Kontext der Dissertation von Benjamin Birkenhake, in der es vorrangig um die narratologische Annotation, aber auch andere Annotationen der enthaltenen Hamlet-Ausgaben geht. Da ich eine eher linguistische Annotation verwirklichen wollte, habe ich die syntaktische Analyse einer Hamlet-Ausgabe thematisiert.
Der Ausgangstext für die syntaktische Annotation ist eine im Format der TEI.drama-DTD annotierte deutsche Ausgabe des „Hamlet“ von William Shakespeare[2]. Diese liegt in Form der Datei „hamlet.xml“ vor. Um einen Eindruck des Ausgangsformats zu bekommen, habe ich in den Anhang (Abschnitt B) einen kleinen Ausschnitt des TEI-Dokuments gestellt. Da das Ergebnis der Transformation des kompletten Ausgangstexts ziemlich umfangreich ist, habe ich in den Ordner „Texte“ TEI-Dokumente, die jeweils einen Akt des Stückes enthalten, gelegt. Um die Transformation zu testen, ist es besser eine dieser Da-

[2]deutsche Übersetzung von August Wilhelm von Schlegel; verfügbar im Projekt Gutenberg unter *http://gutenberg.spiegel.de/shakespr/hamlet1/hamlet.htm*

teien als Ausgangstext auszuwählen[3].
Die Ausgabe des Hamlet bringt einige Besonderheiten mit sich. Probleme entstehen vor allem dadurch, dass relativ viele im Text verwendete Wörter bzw. Wortformen nicht im Lexikon vorhanden sind. Dies betrifft ungefähr 1400 von insgesamt 29000 Wortformen[4]. Das sind immerhin ca. 5%. Leider sind ca. 95% der nicht vorhandenen Wortformen solche, die nur einmal im Text auftauchen, sodass die Quote ohne großen Aufwand manuell nicht sonderlich verbessert werden kann. Die hohe Quote der im Lexikon nicht vorhandenen Wortformen ist auf verschiedene Umstände zurückzuführen. Auf der einen Seite weist die Hamlet-Übersetzung aufgrund ihres Alters z.T. archaische Wörter (z.B.: „gestrengem“) auf. Diese sind nicht in dem Morphologiesystem Morphy, das die Grundlage des Lexikons bildet (vgl. Kapitel 3.1.3), vorhanden. Genauso verhält es sich mit Wörtern, deren Orthographie eher lyrischen Charakter besitzt, also eigenen Konventionen folgt (z.B.: „einge“, statt „einige“ oder auch „hatt“, anstatt „hatte“). Auf der anderen Seite verwendet Schlegel relativ häufig Klitika (z.B.: „hats“). Diese stellen für die syntaktische Annotation insofern ein Problem dar, als dass die orthographische Repräsentation wie in diesem Fall z.B. ein Verb und ein Personalpronomen in sich vereint. Dieses Problem könnte durch bestimmte Methoden der Annotation, z.B. das Wort in zwei Teile aufzuteilen, gelöst werden. Der Versuch einer Lösung ist in der Datei „wortTagging.xsl“ zu finden, wird aber hier nicht explizit vorgestellt.
Die o.g. Probleme für die Annotation des Ausgangstextes dürften generell mit der Textsorte Drama verbunden sein. Es sind aber nicht nur Probleme, die mit dieser Gattung einhergehen. Ein Vorteil, den sie bietet, ist beispielsweise das Fehlen von Abkürzungen. Diese können in Textsorten, in denen sie häufig verwendet werden, z.B. bei der Erkennung von Satzgrenzen zu Problemen führen. Vor allem in Sätzen, in denen eine Abkürzung den letzten Teil des Satzes bildet, ist u.U. schwer zu entscheiden, ob ein Punkt zu einer Abkürzung gehört, ein Satzzeichen ist oder evtl. beide Funktionen gleichzeitig erfüllt. Dieses Problem ergibt sich in dieser Arbeit nicht.

Bei der syntaktischen Annotation des Ausgangstextes wird darauf geachtet, dass die ursprüngliche Annotation so weit wie möglich erhalten bleibt und durch die syntaktische Annotation ergänzt wird. Dieses Vorgehen folgt aus den Überlegungen zu den Anforderungen an linguistische Annotationsformate, die in Kapitel 1.1.1 diskutiert wurden.
Die syntaktische Annotation berücksichtigt lediglich die Textknoten der `<l>`-Elemente (lines) des TEI-Formats. Für die `<l>`-Elemente bedeutet das, dass sie zu leeren Elementen

[3] die Transformationsergebnisse zu diesen Dateien liegen im Ordner „Output“

[4] diese und andere statistische Angaben lassen sich relativ einfach mit Hilfe von XPath-Ausdrücken ermitteln. Der XML-Editor Oxygen bietet hierzu ein Modul, das XPath-Ausdrücke direkt auf die vorliegende Datei anwendet und das Ergebnis ausgibt.

transformiert werden müssen. Dies verändert zwar die Ausgangsannotation, die Funktion der <l>-Elemente bleibt aber im Wesentlichen erhalten. Die Veränderung der Ausgangsannotation ist leider notwendig, um überlappende Hierarchien im XML-Format zu vermeiden. Ansonsten würden sich die Elemente zur Annotation von lines und zur Annotation von Sätzen überschneiden, was dazu führen würde, dass das XML-Dokument nicht wohlgeformt und damit keine XML-Instanz wäre. Es sind auch andere Strategien zur Bewältigung dieses Problems denkbar. Man könnte den umschließenden Charakter des <l>-Elements erhalten, indem man es in mehrere Elemente aufteilt, die eine line an den Satzgrenzen in mehrere Teile aufteilt, d.h. sie in mehrere Elemente untergliedert. Da dieses die ursprüngliche Annotation aber ebenso verändern würde und zudem komplizierter wäre, habe ich mich zur Transformation der <l>-Elemente in leere Elemente entschieden. Alle anderen Elemente aus dem TEI-Tag Set können einfach aus dem Ausgangstext in das Transformationsergebnis kopiert werden. Bei ihnen ergeben sich keine Überschneidungen mit der syntaktischen Annotation.

3.1.2 Das Tag Set

Das für die syntaktische Annotation verwendete Tag Set orientiert sich an den Spezifikationen des XCES (vgl. Kapitel 1.1.2, S. 7). Der Ausgangstext ist im Format der TEI.drama-DTD annotiert. Die Ausgabe der Transformation beinhaltet demnach eine Kombination aus TEI- und XCES-Annotationen. Da die TEI-Annotation kaum verändert wird, beschränke ich mich an dieser Stelle darauf, das Tag Set für die Annotation syntaktischer Strukturen vorzustellen.
Ein erster Eindruck hierzu hat sich bereits im Abschnitt über den XCES ergeben. Einzelne Wörter werden mit dem Element <tok> als Token getagged. Dieses Element enthält genau ein Unterelement <orth> für die orthographische Repräsentation des Tokens und unbeschränkt viele <lex>-Elemente, die die morpho-syntaktischen Informationen enthalten. Diese Informationen sind allerdings anders als ursprünglich im XCES vorgesehen, nicht als Textknoten, die sich am Format des Biber Taggers des American National Corpus orientieren, sondern als Attribute des <lex>-Elements realisiert. Je nach Wortart, die durch das Attribut w festgelegt wird und in jedem <lex> vorhanden sein muss, werden hierbei verschiedene Attribute verwendet. Die Attribute basieren im Wesentlichen auf den durch das Morphologiesystem Morphy (siehe nächster Abschnitt) festgelegten morpho-syntaktischen Informationen, die mit einem Wort verbunden werden. Diese Attribute und ihre zulässigen Werte sind tabellarisch im Anhang (Abschnitt A, S. 52ff) dargestellt.
Ein weiteres verwendetes Element ist <sg> (für Zeichen wie „,“, „-“, „≪“, „"“, usw.). Die orthographische Form dieser Zeichen wird auch hier in einem Unterelement <orth> des Elements <sg> festgehalten. Die Satzzeichen werden nach demselben Prinzip als <snSg>

annotiert.
Die syntaktische Kategorie des Chunks wird als `<chunk>` getagged. Dieses Element enthält zunächst ein Attribut `type`, das die Art des Chunks angibt. Mögliche Werte sind `ac` für Adjektic-Chunks, `nc` für Nominal- und `pc` für Präpositional-Chunks. Die Information über die orthographische Realisierung des gesamten Chunks, die sich aus denen seiner Bestandteile ergibt, liegt analog zu den Token in Form des `<orth>`-Elements vor. Darauf folgen die Informationen über die syntaktischen Eigenschaften des Chunks, wiederum als `<lex>`-Elemente. Die Kategorie der Wortart ist an dieser Stelle nicht mehr zulässig. Die syntaktischen Informationen des Chunks könnten für eine Verarbeitung, die über die Ebene der Chunks hinausgeht, nützlich sein. Die `<lex>`-Elemente werden wiederum von den einzelnen Token (`<tok>`), die Bestandteile des Chunks sind, gefolgt.
Die oben beschriebenen Elemente werden von dem Element `<s>` umschlossen. Dieses Element stellt die Grenzen eines Satzes dar. Das Format eines syntaktisch annotierten Satzes ist im Anhang im Abschnitt C auf Seite 58 oder natürlich in der Ausgabe der XSLT-Transformation nachzuvollziehen.

3.1.3 Das Lexikon: „lexikon.xml“

Die Basis für die syntaktische Annotation bildet ein Vollformenlexikon, das als XML-Dokument vorliegt[5]. Die verwendeten Elemente orientieren sich, wie oben bereits erläutert, am XCES. Das Lexikon enthält etwa 5000 verschiedene orthographische Repräsentationen von Wörtern. Für diese Wörter sind insgesamt ca. 29000 verschiedene Wortformen (morpho-syntaktische Informationen) spezifiziert. Das Lexikon wurde auf der Grundlage einer Wortliste der in der Hamlet-Ausgabe enthaltenen Wörter mit Hilfe des „Morphologiesystem Morphy“ erstellt. Da es sich bei der syntaktischen Annotation um einen bottom-up verlaufenden Parsing-Prozess handelt, bilden die Lexikoneinträge die Eingabe für die syntaktische Analyse. Doch vor der syntaktischen Verarbeitung müssen erst einmal die reinen Textdaten des Ausgangstexts mit morpho-syntktischen Informationen versehen werden.

Das Morphologiesystem Morphy

Für die morphologische Analyse von Wörtern schien mir das Programm Morphy sehr hilfreich. Die hier verwendete Version ist: „Morphologiesystem Morphy für Windows95 und NT, Version 1.1“[6].

[5] zur Pflege dieses Lexikons entstand parallel zu dieser Arbeit im Rahmen des Seminars „Einführung in die Programmierung: Java“ eine in Java implementierte Lexikonverwaltung. (*http://www.linguist.edu.tc/LexJava/Java-Projekt.zip* [14.04.2006])
[6] Download: *http://www.wolfganglezius.de/morphy/download.html*

Das von Wolfgang Lezius entwickelte Programm bietet verschiedene Funktionen rund um die morphologische Analyse. Man kann einerseits ganze Sätze oder Wortsequenzen, andererseits aber auch einzelne Wörter einlesen und morphologisch analysieren lassen. Zur Erstellung des Lexikons habe ich eine Datei mit einer Wortliste verwendet und die Wörter in einer „Tagging-Ausgabe, SGML-ähnlich“, die man neben anderen als Ausgabeformat auswählen kann, ausgeben lassen.

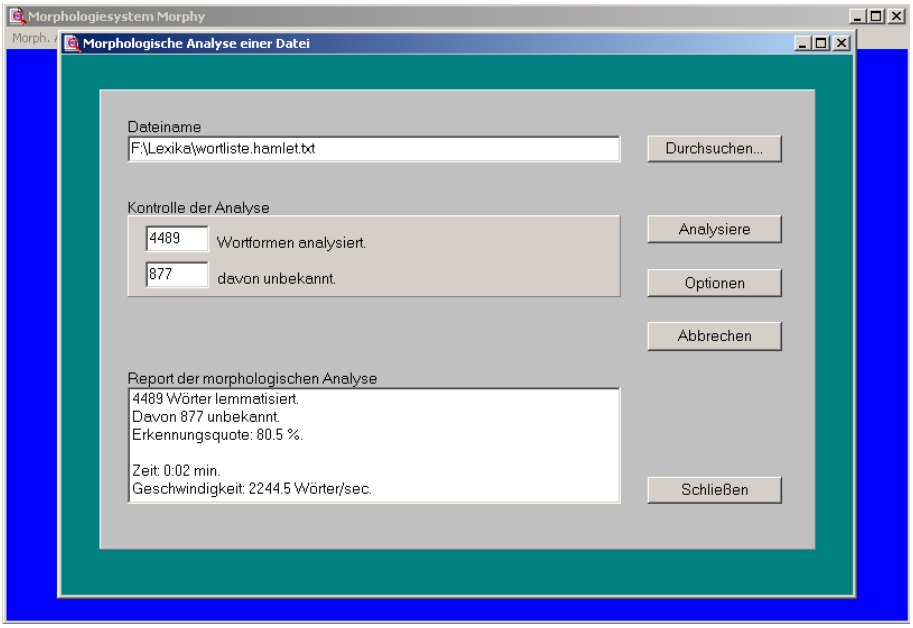


Abbildung 3.1: Morphologische Analyse in Morphy

Da ich das Lexikon nach der Erstellung durch Morphy noch weiterentwickelt habe, weichen die Angaben in der oberen Grafik von den tatsächlichen Statistiken ab. Das Ergebnis der morphologischen Analyse mit Morphy waren Einträge folgenden Formats:

```
<form>Abdruck</form>
<lemma wkl=SUB kas=NOM num=SIN gen=MAS>Abdruck</lemma>
<lemma wkl=SUB kas=DAT num=SIN gen=MAS>Abdruck</lemma>
<lemma wkl=SUB kas=AKK num=SIN gen=MAS>Abdruck</lemma>
```

Diese Ausgabe habe ich anschließend in das Format des oben beschriebenen Tag Sets überführt. Die Attribut- und Wert-Bezeichnungen habe ich zur Verbesserung der Verarbeitungsgeschwindigkeit des XSLT-Prozessors abgekürzt. Die Notwendigkeit hierzu resultierte aus dem Verfahren, das Lexikon als temporären Baum in das XSLT-Stylesheet einzubinden. Die Abkürzungen sind durch die Verwendung von Keys in XSLT, die die Verarbeitungsgeschwindigkeit deutlich verbesserten, evtl. überflüssig geworden. Ich habe sie

dennoch beibehalten. Die Aufschlüsselung der Abkürzungen ist im Anhang A tabellarisch aufgeführt.

Exemplarische Lexikoneinträge

Das Lexikon ist in der Datei „lexikon.xml“ zu finden. Das Format der Lexikoneinträge wird im Folgenden an drei Beispielen verdeutlicht. Dieser Lexikoneintrag ist ein recht simples Beispiel:

```
<tok>
    <orth>Hamlet</orth>
    <lex w="E" kgr="SI-M-1"/>
    <lex w="E" kgr="SI-M-2"/>
    <lex w="E" kgr="SI-M-3"/>
    <lex w="E" kgr="SI-M-4"/>
</tok>
```

Die Orthographie des Wortes ist durch den Inhalt des Elements `<orth>` repräsentiert. Das Wort „Hamlet“ ist mit vier verschiedenen syntaktischen Kontexten verbunden. In allen fungiert das Wort als Eigenname, was durch das Attribut `w` für Wortart und dessen Wert `E` für Eigenname verdeutlicht wird. Das Attribut `kgr` nimmt eine Ausnahmestellung ein. Es besitzt im Gegensatz zu den anderen verwendeten Attributen einen 'komplexen' Inhalt. Im Grunde sind in ihm drei verschiedene Attribute, nämlich die Numerus-, Genus- und Kasus-Informationen eines Wortes, vereint. In diesem Fall steht das Wort „Hamlet“ in jedem Kontext im Singular und Maskulinum. Es kann in allen im Deutschen vorhandenen Kasus (1: Nominativ, 2: Genitiv, usw.) auftreten. Das Wort „Hamlet“ ist somit wie die meisten Lexikoneinträge mit ambigen morpho-syntaktischen Informationen versehen.
Das Attribut `kgr` ist vor allem in Chunks vom Typ NC (Nominal-Chunks) von besonderer Bedeutung. Dort geht es darum, die Kongruenzinformationen der Bestandteile des Chunks (z.B. Artikel, Adjektiv-Chunks, Substantive) miteinander zu kombinieren bzw. erst einmal ihre Kombinierbarkeit zu überprüfen (siehe Kapitel 4.2).
Das zweite Beispiel zeigt, dass ein Wort auch verschiedenen Wortarten angehören kann. Es kommt auf den syntaktischen Kontext an, welche Wortart letztendlich die zutreffende ist.

```
<tok>
    <orth>gleich</orth>
    <lex w="B"/>
    <lex w="PR" r="3"/>
```

```
    <lex w="V" kgr="SI-IM-"/>
    <lex w="A" b="B"/>
</tok>
```

Das Wort „gleich“ kann als Adverb (B), als Präposition (PR), die den Dativ regiert (r="3"), als Verb (V) im Imperativ singular (kgr="SI-IM-") oder auch als Adjektiv (A) mit adverbialem Gebrauch (b="B") auftauchen.
Das kgr-Attribut hat bei Verben eine etwas andere Struktur als bei Determinierern, Adjektiven, Eigennamen oder Substantiven (vgl. Anhang Abschnitt A). Da es bei der Subjekt-Verb-Kongruenz (wird hier nicht behandelt) lediglich auf die Kongruenz des Numerus ankommt, ist dieser bei Verben analog zu nominalen Konstruktionen abgebildet. Sollte jedoch eine erweiterte Analyse verbaler Konstruktionen angedacht werden, wäre die Struktur des kgr-Attributs bei Verben evtl. noch einmal zu überdenken.
Auch im dritten Beispiel eines Lexikoneintrags ist eine starke Ambiguität hinsichtlich der morpho-syntaktischen Informationen zu bemerken. Es handelt sich um das Wort „euer“, das verschiedene pronominale Realisierungen annehmen kann.

```
<tok>
    <orth>euer</orth>
    <lex w="R" kgr="P--2" p="G"/>
    <lex w="PE" kgr="SI--2" p="G"/>
    <lex w="PO" kgr="SI-M-1" b="O"/>
    <lex w="PO" kgr="SI-N-1" b="O"/>
    <lex w="PO" kgr="SI-N-4" b="O"/>
    <lex w="PO" kgr="SI-M-1" b="T"/>
    <lex w="PO" kgr="SI-N-1" b="T"/>
    <lex w="PO" kgr="SI-N-4" b="T"/>
</tok>
```

Es kann einerseits als Reflexivpronomen (R) in der 2. Person (p="G") plural Genitiv fungieren. Im ersten kgr-Attribut bleibt der Genus unterspezifiziert. Eine weitere Möglichkeit ist das Vorkommen des Wortes als Personalpronomen (PE) in der 2. Person singular Genitiv. Desweiteren könnte es ein Possessivpronomen (PO) sein, das entweder pronominal gebraucht (b="O") oder attributiv (b="T"), also als Determinierer, verwendet werden kann.
Es wird deutlich, dass einzelne Wörter z.T. mit sehr ambigen morpho-syntaktischen Informationen verbunden sind. Da diese durch den syntaktischen Kontext disambiguiert werden können, wird eine Aufgabe der Regeln zur syntaktischen Analyse in genau dieser Disambiguierung der Einträge liegen.

3.1.4 Die Transformationsbasis: „tagger.xsl“

Das Stylesheet zur Transformation der Ausgangsdatei („hamlet.xml“) in die Outputdatei mit syntaktischer Annotation ist in der Datei „tagger.xsl“ zu finden. Die einzelnen Schritte der syntaktischen Annotation sind in Form von Funktionen in separaten Dateien ausgelagert (siehe 3.1). In dem Stylesheet „tagger.xsl“ werden zunächst nur die Elemente aus dem TEI-Tag Set kopiert. Das erfolgt durch das nachstehende Template:

```
<xsl:template match="node()|@*" mode="copy">
    <xsl:copy>
        <xsl:copy-of select="@*"/>
        <xsl:apply-templates mode="copy"/>
    </xsl:copy>
</xsl:template>
```

Die Kopie geschieht für jedes Element, das kein Element `<sp>` für speech ist. Für dieses Element ist ein eigenes Template definiert:

```
<xsl:template match="sp" mode="copy">
    <xsl:element name="sp">
        <xsl:copy-of select="@*"/>
        <xsl:copy-of select="speaker"/>

    <!-- ... hier folgen Wort-, Satz- und
                syntaktische Annotationsschritte ... -->

    </xsl:element>
</xsl:template>
```

Auf diese Weise werden zunächst die Attribute und das Unterelement `<speaker>` von `<sp>` kopiert.
Zur Annotation von Wort-, Satz- und Chunk-Strukturen wird danach (oben an der Stelle des Kommentars) ein temporärer Baum erzeugt, dessen Inhalt die Funktionsanwendung der Funktion `my:wortTagging()` auf alle `<l>`-Elemente (lines) innerhalb einer speech ist. Die Funktion ist in der Datei „wortTagging.xsl“ zu finden (siehe Kapitel 3.2.2). Der temporäre Baum enthält nun die `<l>`-Elemente, die, um üpperlappende Hierarchien mit der späteren Satz-Annotation zu verhindern, zu leeren Elementen transformiert wurden. Desweiteren wurden die Wörter innerhalb des Textknotens jedes `<l>`-Elements durch ihren jeweiligen Lexikoneintrag ersetzt. Die einzelnen Wörter sind dementsprechend jetzt als

<tok>-Elemente annotiert. Zeichen (Kommata, Anführungsstriche, etc.) und Satzendezeichen (./;/!/?) wurden ebenfalls durch entsprechende Elemente (<sg>, <snSg>) ersetzt. Der Ergebnisbaum der Wortannotation wird in einer Variablen gespeichert, steht also zur weiteren Verarbeitung als temporärer Baum zur Verfügung. Im nächsten Schritt wird die Funktion `my:satzTagging()` (Datei: „satzTagging.xsl“) aufgerufen (siehe Kapitel 3.3.2). Hier werden anhand der annotierten Satzendezeichen die einzelnen Sätze segmentiert. Auch das Ergebnis dieser Funktionsanwendung wird wiederum in einer Variable als temporärer Baum gespeichert. So steht nun der gesamte Inhalt des sp-Elements, wobei die vorherigen Textknoten nun mit morphologischen Informationen versehen und als Sätze annotiert wurden, zur syntaktischen Annotation bereit.
Die syntaktische Annotation erfolgt im nächsten Schritt durch die Anwendung der Funktion `my:syntTagging()`. Diese Funktion wird auf die einzelnen im Ergebnisbaum der Satzannotation vorhandenen Sätze (<s>-Elemente) angewendet. Hier werden die Chunks annotiert (siehe Kapitel 4). In den nächsten Abschnitten werden die einzelnen Annotationsschritte von der Wort- über die Satz-Annotation bis hin zur eigentlichen syntaktischen Annotation detaillierter vorgestellt.

3.2 Wörter

3.2.1 Wortannotation vor der Satzannotation

Es erscheint auf den ersten Blick ungewöhnlich, die Annotation einzelner Wörter derjenigen von Sätzen vorzuziehen. Dieses Vorgehen hat jedoch einen praktischen Hintergrund. Es ist durch das TEI-Format des Ausgangstextes nicht einfach möglich, Sequenzen von Wörtern mit einem Punkt dahinter als Sätze anzunehmen, weil diese Sätze nicht als Ganze in einem Textknoten stehen, sondern evtl. in Textknoten verschiedener <l>-Elemente verteilt sind. Die <l>-Elemente können Teile von Sätzen, ganze oder auch mehrere Sätze enthalten. Zwar wäre es nun möglich die Textknoten temporär z.B. in einer Variablen zu verbinden, die <l>-Elemente quasi auszublenden. Hierzu müßte jedoch auch die Position jedes einzelnen <l>-Elements vermerkt werden, was die Prozedur relativ umständlich macht. Daher habe ich mich entschlossen, zuerst die einzelnen Wörter und Satzzeichen zu annotieren und auf Basis dieser Annotation, die einzelnen Sätze zu segmentieren. Dieses Verfahren wird in den nächsten Abschnitten etwas deutlicher werden.

3.2.2 Tagging von Wörtern und Zeichen: „wortTagging.xsl“

Die Grundlage der Wortannotation (dieser Begriff soll hier auch die Annotation von Satz- und sonstigen Zeichen umfassen) sind die im Lexikon „lexikon.xml“ enthaltenen Lexikon-

einträge. Die einzelnen Wörter werden durch die in der Datei „wortTagging.xsl“ enthaltene und in die Transformationsbasis „tagger.xsl“ inkludierte Funktion `my:wortTagging()` mit den zu ihnen passenden Lexikoneinträgen in Verbindung gebracht. Die Funktion wird auf jedes `<l>`-Element innerhalb eines `<sp>`-Elements angewendet. Der Ergebnisbaum wird zur weiteren Verarbeitung als temporärer Baum in der Variablen `$wortTags` gespeichert:

```
<xsl:variable name="wortTags">
    <xsl:for-each select="l">
        <xsl:copy-of select="my:wortTagging(.)"/>
    </xsl:for-each>
</xsl:variable>
```

Die Funktion `my:wortTagging()` überprüft nun zunächst, für das als Parameter angegebene `<l>`-Element, ob dieses weitere Unterelemente besitzt. Wenn es Unterelemente enthält, so handelt es sich um `<stage>`-Elemente, die von der syntaktischen Annotation nicht erfasst werden. Folglich kann ein `<l>`-Element, das Unterelemente besitzt, einfach kopiert werden. Ein `<l>`-Element kann nicht gleichzeitig Unterelemente und Textknoten beinhalten.
Enthält `<l>` keine Unterelemente, so hat es Textknoten, die es weiterzuverarbeiten gilt. Hier wird als erstes das `<l>`-Element samt Attributknoten, jedoch ohne Textknoten kopiert. Die Textknoten werden mithilfe regulärer Ausdücke untersucht, d.h. es wird beispielsweise überprüft, ob sich Wörter aus dem Text im Lexikon befinden. Hierzu wurde das Lexikon am Anfang des Stylesheets „tagger.xsl“ als temporärer Baum gespeichert.

```
<xsl:key name="lexikon" match="tok" use="orth"/>
<xsl:variable name="lexikon">
    <xsl:copy-of select="document('../Lexika/lexikon.xml')"/>
</xsl:variable>
```

Die Angabe eines `<xsl:key>`-Elements vor der Erstellung des temporären Baum ermöglicht es, die einzelnen Lexikoneinträge (`<tok>`) anhand des im Attribut `use` angegebenen Elements `<orth>`, also anhand der orthographischen Repräsentation eines Wortes, zu referenzieren und durch die Funktion `key('lexikon', 'orth. Repr.')`, den zu einem Wort passenden Lexikoneintrag (`<tok>`) auszugeben. Dieses führt zu einer mehr als zehn mal schnelleren Verarbeitung durch den XSLT-Prozessor, als ohne Verwendung von Keys.
Aufgrund der Komplexität der Funktion `my:wortTagging()` kann ich hier nicht auf die einzelnen Verarbeitungsschritte eingehen. Stark vereinfacht läßt sich der Prozess der Wortannotation wie folgt darstellen: Reguläre Ausdrücke werden mit dem Inhalt eines `<l>`-Elements verglichen. In diesem Fall werden beispielsweise Wörter, Satzzeichen, sonstige

Zeichen und Whitespaces (Leerzeichen und Zeilenumbrüche) gesucht:

```
<xsl:analyze-string select="."
        regex="(([A-Z]|[a-z]|ö|ä|ü|ß|Ü|Ä|Ö)+)|
               (\.|\?|!|:|;)|(,|'|-|«|») |([\s|\n]+)">
```

Auf diese Weise segmentiert die Funktion einzelne Wörter, Satzzeichen und andere Zeichen. Strings, die auf diesen regulären Ausdruck matchen, können mit dem Element `<xsl:matching-substring>` und die einzelnen Gruppierungen innerhalb des Ausdrucks mit der Funktion `regex-group()` angesprochen und weiterverarbeitet werden. Die gefundenen Wörter werden mit Hilfe der `key()`-Funktion mit den bestehenden Lexikoneinträgen verglichen und bei einem positiven Ergebnis durch diese ersetzt. Die Satzzeichen und sonstigen Zeichen werden mit den entsprechenden Elementen `<sg>` und `<snSg>` annotiert.
Das Ergebnis dieses Verarbeitungsschritts könnte durch die Angabe des Werts „wortebene“ des Stylesheet-Parameters „ebene“ ausgegeben werden. Trägt der Parameter nicht diesen Wert, erfolgt im nächsten Schritt die Annotation von Satzgrenzen.

3.3 Sätze

3.3.1 Die Definition von Sätzen

> „Die Satzdefinition gehört zu den unlösbaren Problemen der Linguistik“ (Henschel/Weydt, 2003: 332).

Dieses Zitat verdeutlicht das Ausmaß der Schwierigkeit, eine allgemein gültige Definition von Sätzen zu finden.
Bei der Suche nach einer treffenden Definition des Satzes wird schnell deutlich, dass diese Definition nicht alle möglichen Aspekte eines Satzes beinhalten kann. Bereits im Jahr 1894 stellte J. Ries 140 unterschiedliche Satzdefinitionen zusammen (vgl. Henschel/Weydt, 2003: 332). Wie Ries zu dieser Fülle an Definitionen gelangt ist, wird klar, wenn man sich allein die unterschiedlichen Perspektiven vor Augen führt, die Henschel/Weydt liefern. Ihnen zufolge ließen sich z.B. folgende Perspektiven, einen Satz aufzufassen, unterscheiden (Henschel/Weydt, 2003: 333): der Satz als

(a) logisch-kognitive Einheit,
(b) philosophisch-logische Einheit,
(c) oberste grammatische Einheit,
(d) relativ selbständigen syntaktischen Komplex,

(e) grammatische Einheit mit Prädikat und Subjekt,
(f) Ausdruck einer vollständigen Mitteilung,
(g) Entsprechung eines Sprechaktes,
(h) intonatorisch bzw. durch Satzzeichen abgeschlossenen Einheit,
(i) sprachliche Einheit, die von einem Verb bestimmt ist,
(j) sprachliche Einheit, die ein finites Verb enthält.

Es wird deutlich, dass das Verständnis von Sätzen je nach Kontext stark voneinander abweichen kann. Entscheidend für die Definition eines Satzes ist also offensichtlich das Erkenntnisinteresse, mit dem diese Definition verbunden ist. Aus der Sicht dieser Arbeit läßt sich das Interesse und damit die notwendige Definition des Satzes relativ genau einschränken. Es geht zunächst darum, eine Definition zu finden, die alle in der hier vorliegenden Fassung des Hamlet befindlichen Sätze eindeutig als Sätze identifiziert. Aufgrund der maschinellen Verarbeitung des Textes bietet es sich an, die Interpunktion als Faktor für die Satzerkennung zu verwenden (vgl. Definition (h)).
Durch das bereits erwähnte Charakteristikum der Textgattung Drama, sehr wenige bis gar keine Abkürzungen zu verwenden, können mit dieser Strategie gute Ergebnisse erzielt werden. Die oben diskutierte Annotation des Ausgangstextes erschwert hier allerdings eine einfache Satzerkennung auf der Grundlage regulärer Ausdrücke. Stattdessen wurden die Wörter und Zeichen bereits vor der Satzgrenzenerkennung annotiert und helfen im nächsten Schritt, die Sätze zu segmentieren. Die Auszeichnung von Satzgrenzen erfolgt somit auf Grundlage von `<snSg>`-Elementen.

3.3.2 Tagging von Sätzen: „satzTagging.xsl“

Die erste Besonderheit der Satzannotation ist, dass sie nur durchgeführt wird, wenn dem Stylesheet-Parameter beim Aufruf der Transformation nicht der Wert „wortebene“ zugewiesen wird. Falls der Parameter mit dem Wert „wortebene“ versehen wurde, wird das Ergebnis der Wortannotation (als temporärer Baum in `$wortTags`) als Output der Transformation festgelegt:

```
<xsl:choose>
    <xsl:when test="$ebene='wortebene'">
        <xsl:copy-of select="$wortTags"/>
    </xsl:when>
    <xsl:otherwise>
        <xsl:variable name="satzTags">
            <xsl:for-each select="$wortTags/*">
```

```
                <xsl:sequence select="my:satzTagging(.)"/>
            </xsl:for-each>
        </xsl:variable>

        <!-- ... hier folgt die syntaktische Annotation ... -->

    </xsl:otherwise>
</xsl:choose>
```

Ist der Parameter beim Transformationsaufruf nicht spezifiziert oder wird eine höhere Ebene als die Wortannotation als Output angefordert, dann sorgt der Aufruf der Funktion `my:satzTagging()` für das Einfügen von Elementen für die Repräsentation von Satzgrenzen. Auch diese Funktion ist in eine separate Datei, nämlich „satzTagging.xsl“, ausgelagert. Sie wird auf jedes einzelne Kindelement des Wurzelknotens des temporären Baums `$wortTags` angewendet. Die Kindelemente sind `<l>`-, `<stage>`-, `<tok>`-, `<sg>`- und `<snSg>`-Elemente.
Die Funktion führt für den angegebenen Parameter eine Fallunterscheidung durch. Falls dieser ein `<l>`-Element ist, das nicht innerhalb eines Satzes steht, also nicht `<tok>` oder `<sg>` als direkte Geschwisterelemente hat, wird es einfach kopiert. Diese Definition trifft vor allem auf `<l>`-Elemente zu, die direkt am Anfang eines Satzes stehen und daher lieber außerhalb der Satzgrenzen ausgegeben werden sollen. Auch `<stage>`-Elemente werden einfach kopiert. Diese können ohnehin nur außerhalb der Satzgrenzen auftreten.
Die eigentliche Satzannotation wird nur in einem bestimmten Fall durchgeführt: Ist der Parameter der Funktion `my:satzTagging()` ein `<snSg>` (Satzzeichen), so startet die Annotation des Satzes. Zunächst wird hierzu die Anzahl der dem `<snSg>`-Element vorhergehenden `<snSg>`-Elemente in einer Variablen gespeichert. Danach wird das Satz-Element `<s>` erstellt. Dort hinein werden alle dem aktuellen `<snSg>`-Element vorhergehenden Geschwisterelemente kopiert, die zwischen dem aktuellen `<snSg>` und dem nächsten davor liegenden `<snSg>` stehen. Nun wird noch das aktuelle `<snSg>`-Element selbst als letztes Element in das `<s>`-Element kopiert. Auf diese Weise erhält man mit dem Element `<s>` annotierte Sätze, die wiederum die Elemente `<tok>`, `<sg>`, falls vorhanden `<l>` sowie `<snSg>` beinhalten. Diese Annotationsstufe kann durch die Angabe des Parameterwerts „satzebene“ beim Aufruf der XSLT-Transformation in der Datei „output-satzebene.xml“ gespeichert werden. Ist das Ziel eine weitere syntaktische Annotation, so bilden die beiden bisherigen Annotationsstufen die Grundlage für die bottom-up durchgeführte Annotation von Chunks.

Kapitel 4

Syntaktische Annotation

Die syntaktische Annotation erfolgt, wenn nicht die Ebenen der Wort- oder Satzannotation als Ausgabeformat spezifiziert wurden. Analog zu den vorherigen Annotationsstufen ist auch die Auszeichnung syntaktischer Strukturen in einer eigenen Funktion (`my:syntTagging()`) in eine eigene Datei mit dem Namen „syntTagging.xsl“ ausgelagert. Die Funktionsanwendung erfolgt folgendermaßen:

```
<xsl:for-each select="$satzTags/*">
    <xsl:choose>
        <xsl:when test="self::l">
            <xsl:copy-of select="."/>
        </xsl:when>
        <xsl:when test="self::stage">
            <xsl:copy-of select="."/>
        </xsl:when>
        <xsl:when test="self::s">
            <xsl:copy-of select="my:syntTagging(., $ebene)"/>
        </xsl:when>
    </xsl:choose>
</xsl:for-each>
```

Die Funktion ist Teil einer Fallunterscheidung, die für die Kindelemente des Wurzelknotens des temporären Baums `$satzTags` durchgeführt wird. Hierbei werden diejenigen Elemente, die für die syntaktische Annotation uninteressant sind, lediglich kopiert. Für die enthaltenen Sätze wird die Funktion `my:syntTagging()` aufgerufen, die als ersten Parameter das jeweilige `<s>`-Element und als zweiten Parameter die gewünschte Annotationsebene erhält (falls z.B. nur Adjektiv-Chunks annotiert werden sollen).
In der Datei „syntTagging.xsl“ sind nach dem gleichen Prinzip wie schon in der Transfor-

mationsbasis „tagger.xsl“ weitere Funktionen aus einzelnen Dateien eingebunden, die für die Annotation verschiedener Chunk-Typen verantwortlich sind (vgl. das Schaubild auf Seite 22). So kann auch hier die gewünschte Annotationsebene des Transformationsergebnisses durch die Verwendung des bereits mehrfach erwähnten Parameters bestimmt werden. Die einzelnen Funktionen zur Annotation der Chunks werden in den nachfolgenden Abschnitten erläutert. Einen annotierten Beispielsatz, an dem die einzelnen Chunk-Annotationen nachvollziehbar sind, habe ich im Anhang im Abschnitt C aufgeführt.
Eine Besonderheit der vorgestellten syntaktischen Annotation ist, dass keine Verbal- oder Adverbial-Chunks annotiert werden. Bei dem Versuch einer automatischen syntaktischen Annotation dieser Chunk-Typen stellte sich heraus, dass diese nur unzuverlässig funktionierte. Gemäß dem Grundsatz „Korrektheit vor Vollständigkeit“, der auf das Chunk-Parsing wohl zutrifft, werden Verbal- und Adverbial-Chunks daher nicht erfasst. Adverben können jedoch Bestandteile von Adjektiv-Chunks sein. Das verdeutlicht der nachstehende Abschnitt.

4.1 Adjektiv-Chunks

4.1.1 Die Struktur von Adjektiv-Chunks

Die Annotation von Adjektiv-Chunks bildet die erste Stufe der syntaktischen Annotation. Beim Starten der XSLT-Transformation kann dem Stylesheet-Parameter der Wert „AC-Ebene“ zugewiesen werden, um diese Annotationsstufe als Output zu erhalten.
Die Komplexität eines Adjektiv-Chunks bleibt hier auf einem recht niedrigen Level (vgl. Abschnitt 4.1.2). Die Struktur kann vereinfacht durch folgende Regel repräsentiert werden:

$$AC_{kgr='\#1'\ a='\#2'} \rightarrow (ADV)\ \ ADJ_{kgr='\#1'\ a='\#2'}$$

Die verwendeten Abkürzungen stehen hierbei für den Chunk-Typ (AC), die Wortarten der in ihm enthaltenen Wörter und deren morpho-syntaktische Eigenschaften. Im Anhang Abschnitt A werden die Abkürzungen tabellarisch aufgeschlüsselt. Nach obiger Regel besteht ein Adjektiv-Chunk aus einem Adjektiv und einem vorangehenden optionalen Adverb. Die morpho-syntaktischen Informationen des Adjektivs werden an den gesamten Chunk weitergegeben (ähnlich zum Head-Feature-Principle in der HPSG). Dieser Vorgang soll durch die tiefergestellten morpho-syntaktischen Informationen in der Regel dargestellt werden. Die Notation #1 impliziert hierbei, dass die Werte, die durch diese Bezeichnung repräsentiert werden, gleich sind. Das erinnert ein wenig an die Verwendung von „Tags“ in Unifikationsgrammatiken (vgl. Kapitel 2.1.3). Die Unifikationsoperation wird im Kapitel 4.2, in der es um die Annotation von Nominal-Chunks geht, noch eingehender thematisiert.

Die obige Regel kann die Struktur eines Adjektiv-Chunks allerdings nur z.T. aufzeigen. Die meisten Beschreibungen der Struktur von Chunks enthalten zusätzliche kontextsensitive Aspekte. Im Fall der Adjektiv-Chunks ist dies z.B. die Bedingung, dass ein Adjektiv innerhalb dieses Chunks nicht gleichzeitig als Substantiv verwendet werden könnte (Bsp.: „Großen“). Solche kontextsensitiven Erweiterungen sind jedoch nicht uneingeschränkt in der Lage, die morpho-syntaktischen Ambiguitäten korrekt aufzulösen. Man stelle sich vor, das Wort „Großen“ stünde am Anfang eines Satzes und würde von einem Substantiv gefolgt. Hier würde die genannte Regel versagen. Diese Beschränkungen ließen sich durchaus durch weitere Bedingungen zumindest teilweise aufheben. Allerdings werden die Regeln heirdurch recht schnell sehr unübersichtlich.
Es wird deutlich, dass die zugrundegelegten Annahmen über die Kombination von Wörtern zu Chunks nicht perfekt sind und mit größerem Aufwand (z.B. dem Einsatz statistischer Verfahren oder weiteren Disambiguierungsregeln) stark verbessert werden könnten. Diesen Aufwand konnte ich in dieser Arbeit allerdings nicht betreiben.

4.1.2 Tagging von Adjektiv-Chunks: „acTagging.xsl“

Die Funktion, die der Annotation von Adjektiv-Chunks dient, ist in der o.g. Datei „acTagging.xsl“ zu finden. Dem Prinzip dieser Funktion folgen auch die anderen Funktionen für die syntaktische Annotation.

```
<xsl:function name="my:acTagging">
    <xsl:param name="eingabe" as="element()"/>
    <xsl:element name="s">
        <xsl:for-each select="$eingabe/*">
            <xsl:choose>

            <!-- ... Fallunterscheidung ... -->

            </xsl:choose>
        </xsl:for-each>
    </xsl:element>
</xsl:function>
```

Als Parameter wird der Funktion ein einzelnes `<s>`-Element übergeben. Dieses entält die annotierten Wörter, Zeichen, Satzzeichen und u.U. auch `<l>`-Elemente. Diese direkten Kindelemente des `<s>`-Elements werden in einer `for-each`-Schleife und einer anschließenden komplexen Fallunterscheidung weiterverarbeitet. Hieran wird die verwendete „Vom-Anfang-zum-Ende“-Verarbeitung deutlich. Alle in einem `<s>`-Element vorhandenen Kin-

delemente werden der Reihe nach angesprochen und untersucht.
Die vereinfachten Regeln, die ich zu jedem Chunk-Typ angebe, werden in komplexerer Form in dieser Fallunterscheidung mit Hilfe von XPath-Ausdrücken angewendet. Als ein Beispiel hierfür kann dieser Ausdruck herangezogen werden:

```
self::tok[lex[(@w='A' or @w='P1' or @w='P2') and
              (not(@b='B') or exists(@kgr))]
          and not(lex/@w='S')]
```

Dieser Pfad referenziert ein `<tok>`-Element, dessen Unterelement `<lex>` einen Attributknoten mit dem Namen `w` besitzt. Der Wert des Attributs muss entweder `A`, `P1` oder `P2` sein. Außerdem muss die Bedingung erfüllt sein, dass entweder ein weiteres Attribut mit dem Namen `kgr` vorhanden ist oder ein evtl. vorhandenes Attribut `b` nicht den Wert `B` besitzt. Desweiteren darf das `<tok>`-Element kein `<lex>`-Element beinhalten, dessen Attribut `w` den Wert `S` hat.
Und nun in weniger technischen Worten: Es wird ein Wort gesucht, dessen Wortart Adjektiv, Partizip1 oder Partizip2 ist und das nicht gleichzeitig in adverbialem Gebrauch steht oder als Substantiv fungieren kann. Ein Wort mit diesen Eigenschaften ist mit hoher Wahrscheinlichkeit Bestandteil eines Adjektiv-Chunks. Demnach beginnt an dieser Stelle die Annotation eines Chunks, wenn ein solches `<tok>`-Element gefunden wird. In weiteren Schritten, ebenfalls durch Fallunterscheidungen und XPath-Ausdrücke realisiert, wird überprüft, ob direkt vor dem eben beschriebenen Element ein Element steht, das ein Adverb repräsentiert (XPath: `preceding-sibling::*[1][lex[@w='B' or @b='B']]`). Dieses würde mit in den Adjektiv-Chunk aufgenommen werden.
Alle Elemente, die durch diese XPath-Ausdrücke nicht angesprochen wurden, das bedeutet die nicht zu Adjektiv-Chunks gehörenden Elemente, werden einfach kopiert. Hier wird der Unterschied zu gängigen Syntax-Parsern, die eine strikte „von Anfang zum Ende“-Verarbeitung durchführen, deutlich. Zwar verläuft die Verarbeitung durch den XSLT-Prozessor auch hier vom Anfang zum Ende eines Satzes, aber nur die Elemente, die für den jeweils zu annotierenden Chunk-Typ von Bedeutung sind, unterliegen einer komplexeren Verarbeitung als einer reinen Kopie. Diese Strategie steht im Zusammenhang mit dem sog. Insel-Parsing, bei dem ebenfalls nur die interessanten Stellen weiterverarbeitet werden. Das Insel-Parsing erfolgt jedoch nicht zwangsläufig Vom-Anfang-zum-Ende.
Anzumerken ist an dieser Stelle desweiteren, dass die Annotation von Adjektiv-Chunks auf einer relativ flachen Ebene erfolgt. Komplexe Adjektiv-Chunks wie „der $[_{AC}$ des Englischen mächtige $_{AC}]$ Mann“ oder „der $[_{AC}$ auf den Startschuss wartende $_{AC}]$ Läufer“ werden nicht annotiert. Das resultiert aus der Reihenfolge der Chunk-Annotationen. Als erstes werden die oben beschriebenen Adjektiv-Chunks annotiert. Danach Nominal-Chunks, die theoretisch Bestandteile von Adjektiv-Chunks sein könnten. Um komplexe Adjektiv-Chunks

annotieren zu können, müßte die Nominal-Chunk-Annotation bereits durchgeführt worden sein oder die Funktion zur Annotation von Adjektiv-Chunks mehrmals aufgerufen werden. Da dieses Vorgehen einige Schwierigkeiten[1] mit sich brächte, habe ich diese Möglichkeit nicht weiter verfolgt.

4.2 Nominal-Chunks

4.2.1 Die Struktur von Nominal-Chunks

Die folgenden Regeln verdeutlichen die möglichen Bestandteile von Nominal-Chunks und die Bedingungen ihrer Kombinierbarkeit. Auch hier gilt, dass diese Regeln nur den Kern der tatsächlich in XSLT realisierten Bedingungen demonstrieren.

$$NC_{kgr='\#1'} \rightarrow [S|E]_{kgr='\#1'}$$

Die erste Regel verdeutlicht die einfachste Art von Nominal-Chunks. Diese bestehen lediglich aus einem Substantiv oder Eigennamen[2]. Dessen Kongruenzinformationen werden an den Nominal-Chunk „vererbt“. Wichtig für das generelle Verständnis der hier angestrebten Annotation von Nominal-Chunks ist, zu wissen, dass die obige Regel nur dann zur Anwendung kommt, falls das Substantiv bzw. der Eigenname nicht Bestandteil eines komplexeren Nominal-Chunks (s.u.) sein kann. Diese Bedingung wird wiederum durch kontextsensitive Bestandteile der Regel, die hier nicht aufgeführt sind, gewährleistet. Dasselbe gilt für alle Regeln, bei denen Bestandteile von Nominal-Chunks theoretisch Teil eines komplexeren Nominal-Chunks sein können. So auch bei folgender Regel:

$$NC_{kgr='\#1'} \rightarrow [[DE|IR|PO|RL|I]_{b='O'}|[Z|PE|R]|I_{exists(p)}]_{kgr='\#1'}$$

Hier ist der einzige Bestandteil des Chunks ein Pronomen bzw. ein Zahlwort. Die Pronomen müssen allerdings je nach Wortart bestimmte Bedingungen erfüllen. Demonstrativ-, Interrogativ- Possessiv-, Relativ- und Indefinitpronomen müssen für pronominalen Gebrauch (b='O') spezifiziert sein. Außerdem gibt es Indefinitpronomen, die kein Attribut für den Gebrauch besitzen. Diese verfügen stattdessen über eine Information zur Person (p). Ein Beispiel hierfür ist das Wort „seinesgleichen“.
Eine etwas komplexere Form von Nominal-Chunks charakterisiert diese Regel:

$$NC_{kgr='\#1'} \rightarrow [[AR|Z]|[DE|IR|PO|RL|I]_{b='T'}]_{kgr='\#1'} \; [S|E]_{kgr='\#1'}$$

[1] Welche Reihenfolge der Chunk-Annotation soll ausgeführt werden?
Wie oft sollen die Funktionen aufgerufen werden? usw.

[2] dies wird ausgedrückt durch die Notation $[S|E]$. Die eckigen Klammern beinhalten Mengen, der Operator | stellt ein logisches Oder dar.

Die Bestandteile sind ein Determinierer und ein Substantiv/Eigenname. Wieder muss der Determinierer abhängig von seiner Wortart bestimmte Bedingungen erfüllen. Die verwendbaren Pronomen müssen eine morpho-syntaktische Information über attributiven Gebrauch (*b='T'*) haben. Artikel und Zahlwörter müssen keine weitere Bedingung erfüllen, außer derjenigen, die ohnehin für alle Bestandteile eines Nominal-Chunks gilt: Ihre Kongruenzinformationen müssen kombinierbar sein. Diese Kombinierbarkeit ist kein triviales Problem, da die Informationen unterspezifiziert sein können. Zum Beispiel würden *kgr='- -2'* (gilt z.B. für „wessen“) und *kgr='SI-M-2'* (gilt z.B. für „Mannes“) durchaus auf passende Kongruenzinformationen hinweisen. Um diese Informationen zu verarbeiten, habe ich zwei Funktionen erstellt. Die erste überprüft, ob eine Kombination möglich ist und gibt einen Wahrheitswert zurück. Die zweite Funktion kombiniert die Kongruenzinformationen (sofern überhaupt möglich), und gibt die kombinierte Information zurück. Mit den obigen beiden Kongruenzinformationen wäre dies: *kgr='SI-M-2'*. Diese kombinierte Kongruenzinformation wird als Kongruenzinformation des gesamten Nominal-Chunks übernommen. Dieses Vorgehen erinnert im Ansatz an die Unifikationsoperation in Unifikationsgrammatiken (siehe Kapitel 2.1.3). Jedoch ist sie keineswegs damit gleichzusetzen. Die hier verwendete Funktion (`my:unifyKgr()`, siehe Abschnitt 4.2.2) ist nur auf Kongruenzinformationen und das festgelegte Format des `kgr`-Attributs beschränkt und nicht für andere Kombinationen verwendbar. Sie ist zwar von der Unifikation inspiriert, besitzt jedoch nur einen Bruchteil ihrer Ausdrucksstärke.
Die beschriebenen Kongruenz-Operationen erfolgen in gleicher Weise bei der nächsten Art von Nominal-Chunk. Dieser besteht aus einem Determinierer und einem Adjektiv-Chunk:

$$NC_{kgr='\#1'} \rightarrow [[AR_{t='\#2'}|Z]|[DE|IR|PO|RL|I]_{b='T'}]_{kgr='\#1'}$$
$$AC_{kgr='\#1'\ a='\#2'}$$

Hier wird deutlich, warum Adjektiv-Chunks mit Informationen über Kongruenz und Artikelgebrauch versehen wurden. Die Kongruenzinformationen der Bestandteile werden analog zur eben erläuterten Regel verarbeitet. Von Bedeutung ist auch die Übereinstimmung zwischen dem Typ des Artikels (t) und dem Artikelgebrauch für den Adjektiv-Chunk. Auf diese Weise können Konstruktionen wie „ein große“ von wohlgeformten Kombinationen wie „das große“ unterschieden werden. Für Pronomen als Determinierer muss wiederum gewährleistet sein, dass sie für pronominalen Gebrauch spezifiziert sind. Ausgeblendet wird in dieser Regel, dass dem Adjektiv-Chunk kein Substantiv oder Eigenname folgen darf, der zu diesem Nominal-Chunk gehören könnte. Dies wird jedoch im zugehörigen XPath-Ausdruck im XSLT-Stylesheet ausgedrückt.
Der Artikelgebrauch ist auch in folgender Regel von Bedeutung. In solch einem Nominal-Chunk dürfen nur Adjektiv-Chunks, die nicht durch einen Determinierer modifiziert wer-

den, vorkommen. Solche Adjektiv-Chunks sind durch *a='SO'* (für Artikelgebrauch='alleinstehend', also kein Artikelgebrauch) spezifiziert.

$$NC_{kgr='\#1'} \rightarrow AC_{kgr='\#1'\ a='SO'}\ \ [S|E]_{kgr='\#1'}$$

Diese Einschränkung verhindert Wortfolgen wie „die große Häuser". Dagegen würde „große Häuser" ohne Determinierer als wohlgeformt erkannt und in einen Nominal-Chunk gefasst werden.
Der komplexeste Nominal-Chunk ist wie folgt zu beschreiben. Er besteht aus einem Determinierer, einem Adjektiv-Chunk und einem Substantiv oder Eigennamen.

$$\begin{aligned} NC_{kgr='\#1'} \rightarrow & [[AR_{t='\#2'}|Z]|[DE|IR|PO|RL|I]_{b='T'}]_{kgr='\#1'} \\ & AC_{kgr='\#1'\ a='\#2'}\ \ [S|E]_{kgr='\#1'} \end{aligned}$$

Auch hier gelten in entsprechender Weise die Bedingungen der Kongruenz, Übereinstimmung von Artikeltyp und Artikelgebrauch des Adjektiv-Chunks und der attributive Gebrauch der Pronomen. Die Funktion dieser Bedingungen wurde bereits in den anderen Regeln erläutert. Daher werde ich nun dazu übergehen, die Implemetierung der Regeln in XSLT vorzustellen.

4.2.2 Tagging von Nominal-Chunks: „ncTagging.xsl"

Die in die Datei „ncTagging.xsl" ausgelagerte Funktion `my:ncTagging()` erhält wie alle für die Annotation von Chunks verantwortlichen Funktionen einen Satz (`<s>`) als Parameter (vgl. Kapitel 4.1.2). Für jedes direkte Unterelement dieses `<s>`-Elements wird dann mit Hilfe von XPath-Ausdrücken die Möglichkeit überprüft, ob es Teil eines Nominal-Chunks sein könnte. Dieser Prozess verläuft über die Formulierung komplexer Fallunterscheidungen.
Auf diese Weise könnte z.B. ein Element, das ein Substantiv oder einen Eigennamen repräsentiert, erreicht werden (XPath: `self::tok[lex/@w='S' or lex/@w='E']`). Dieses wäre auf jeden Fall Teil eines Nominal-Chunks, sodass eine weitere Fallunterscheidung durchgeführt werden kann. Diese überprüft, ob das Substantiv/der Eigenname vielleicht direkt hinter einem Determinierer oder Adjektiv-Chunks steht. Diese könnten dann u.U. demselben Nominal-Chunk angehören. Der Fall, in dem vor einem Substantiv/Eigennamen ein Determinierer und ein Adjektiv-Chunk stehen, ist durch diesen XPath-Ausdruck beschreibbar:

```
preceding-sibling::*[2][lex[(@w='AR' or @w='Z') or
                            ((@w='PO' or @w='DE' or @w='IR' or
                              @w='I' or @w='RL') and @b='T')]]
```

```
  and
preceding-sibling::*[1][exists(lex/@a!='SO')]
  and
my:existsUnifyKgr(my:unifyKgr(self::tok, preceding-sibling::*[1]),
                  preceding-sibling::*[2])
```

Dieser Ausdruck wird durch einen Teil der letzten Regel für Nominal-Chunks im vorigen Abschnitt dargestellt. An der Stelle, an der dieser Ausdruck angewendet wird, steht bereits fest, dass das aktuelle Element (`self::`) ein Wort mit der Wortart Substantiv oder Eigenname repräsentiert. Für das direkt vorhergehende Element (in XPath-Ausdruck: `preceding-sibling::*[1]`) wird getestet, ob es ein Unterelement `<lex>` gibt, das ein Attribut `a` besitzt, dessen Wert nicht `SO` ist. Angesprochen werden hiervon die zuvor annotierten Adjektiv-Chunks, die von einem Determinierer modifiziert werden müssen (`SO` steht für alleinstehend). Das an zweiter Stelle vorhergehende Element soll ein Wort mit Determinierer-Charakter darstellen.
Um nun entscheiden zu können, ob es sich tatsächlich um einen wohlgeformten Nominal-Chunk handelt, kommen die Funktionen `my:existsUnifyKgr()` und `my:unifyKgr()` zum Einsatz. Sie werden in den nächsten Abschnitten erklärt. Liefert `my:existsUnifyKgr()` den Wahrheitswert 1, so beginnt die Erstellung des `<chunk type="nc">`-Elements. Mit Hilfe von `my:unifyKgr()` können die Kongruenzinformationen des gesamten Chunks durch die Kombination der Kongruenzinformationen der einzelnen Bestandteile erstellt werden. Das Ergebnis der Kombination kann im weiteren Verlauf dazu genutzt werden, die morpho-syntaktischen Informationen der einzelnen Chunk-Bestandteile zu disambiguieren. Diejenigen Informationen, die nicht zu denen des gesamten Chunks passen, können daher bei der Transformation der Bestandteile in Unterelemente des `<chunk>`-Elements weggelassen werden. Diese Prozedur ist bei `<tok>`-Elementen, die Unterelemente des `<chunk>` sind, relativ leicht durchzuführen. Es wird ein temporärer Baum erstellt, der die Kongruenzinformation des gesamten Chunks enthält. Daraufhin erfolgt ein Vergleich der morpho-syntaktischen Informationen und es werden bei der Transformation nur die `<lex>`-Elemente, die Unterelemente von `<tok>` sind und deren Kongruenz mit der des gesamten Chunks übereinstimmt, kopiert.
Bei Bestandteilen des Nominal-Chunks, die mehrere Ebenen von `<lex>`-Elementen enthalten (also Adjektiv-Chunks[3]), wird die kongruenz-abhängige Kopie von `<lex>`-Elementen von der Funktion `my:copyDescAC()` übernommen (s.u.).

[3] Adjektiv-Chunks enthalten selber morpho-syntaktische Informationen (<lex>-Elemente), aber auch <tok>-Elemente, die wiederum morpho-syntaktische Informationen beinhalten

Die Funktion „my:unifyKgr()“

Die Funktion `my:unifyKgr()` ist in der Datei „syntFunctions.xsl“ zu finden. Auf diese Weise kann sie von den verschiedenen ebenfalls in separaten Dateien vorliegenden Funktionen zur Chunk-Annotation gleichermaßen verwendet werden.
Die Parameter der Funktion sind zwei Elemente, die als direkte Kindelemente `<lex>`-Elemente mit `kgr`-Attribut enthalten müssen, um die Funktion sinnvoll auszuführen. Für jedes `<lex>`-Element des ersten Parameters wird dessen Kongruenzinformation, also der Wert des `kgr`-Attributs, mit jeder Kongruenzinformation des zweiten Parameters einzeln verglichen. Das Attribut hat folgendes Format: `kgr='`*num-gen-kas*`'`. Bei dem Vergleich werden verschiedene Fälle betrachtet. Der einfachste Fall ist der, in dem die beiden Kongruenzinformationen gleich sind. Dann wird in einem temporären Baum ein Element `<lex>` erstellt, dem ein `kgr`-Attribut mit einem der beiden Werte zugewiesen wird. Haben die beiden Kongruenzinformationen in mindestens einem Teil (Numerus, Genus, Kasus) nicht kombinierbare Angaben, so wird kein Element als Repräsentant für die kombinierte Information erstellt. Dies tritt beispielsweise bei derartigen Informationen ein: 1.: `kgr='SI-N-3'`; 2.: `kgr='SI-F-3'`.
Nun können auch Fälle auftreten, in denen eine oder mehrere Angaben unterspezifiziert sind: 1.: `kgr='SI--3'`; 2.: `kgr='-F-3'`. Hier würde zur Darstellung des Ergebnisses der Kombination ein `<lex>`-Element in den temporären Baum geschrieben werden, dessen Attribut so aussieht: `kgr='SI-F-3'`. Durch dieses Verfahren kann mit Unterspezifikationen der Kongruenzinformation adäquat umgegangen werden. Der erstellte temporäre Baum kann zum Schluß dazu verwendet werden, die Kongruenzinformationen eines kompletten Nominal-Chunks darzustellen. Hierzu werden die im Baum enthaltenen `<lex>`-Elemente als Unterelemente des `<chunk>`-Elements realisiert. Im temporären Baum wurden zuvor doppelte Vorkommen gleicher morpho-syntaktischer Information gelöscht. Dies geschieht durch die Funktion `my:delDoubleKgr()`, die alle `<lex>`-Elemente ihres Parameters daraufhin überprüft, ob sie vorangehende Geschwisterelemente haben, die die gleichen Kongruenzinformationen enthalten, wie sie selbst. Ist dies nicht der Fall werden sie in den Ergebnisbaum kopiert.

Die Funktion „my:existsUnifyKgr()“

Die Funktion `my:existsUnifyKgr()` (ebenfalls in „syntFunctions.xsl“) ist relativ eng mit `my:unifyKgr()` verbunden. Sie testet, ob zwei Elemente kombinierbare Kongruenzinformationen beinhalten. Dazu nimmt die Funktion das Ergebnis von `my:unifyKgr()` zu Hilfe. Wenn das Ergebnis `<lex>`-Elemente enthält, sind die beiden Elemente, die als Parameter angegeben wurden, kombinierbar. Demnach wird hier der Wahrheitswert 1 als

Rückgabewert festgelegt. Sind in dem Ergebnisbaum der Funktion `my:unifyKgr()` keine `<lex>`-Elemente zu finden, wird der Wahrheitswert 0 zurückgegeben. Die Anwendung dieser Funktion wurde bereits vorgestellt (siehe Kapitel 4.2.2).

Die Funktion „my:copyDescAC()“

Um die Kongruenzinformationen von Adjektiv-Chunks und den in ihnen enthaltenen `<tok>`-Elementen (Adjektive) innerhalb eines Nominal-Chunks disambiguieren zu können, habe ich die Funktion `my:copyDescAC()` definiert. Ihr Name verrät bereits den Zweck dieser Funktion. Sie soll alle Nachfahren (Desc für descendants) eines `<chunk type='ac'>`-Elements in Abhängigkeit verschiedener Faktoren kopieren. Der erste Parameter der Funktion ist das `<chunk>`-Element selbst. Zwei weitere Parameter sind der sich in der Kombination aus Determinierer und Adjektiv-Chunk im Nominal-Chunk ergebende Artikelgebrauch, sowie die Kongruenzinformation des Nominal-Chunks.
Als nächstes werden nun alle Nachfahren des ersten Parameters (Adjektiv-Chunk) durch eine Schleife und eine Fallunterscheidung bearbeitet. Das Element `<chunk type='ac'>` kann nur drei Typen von Elementen enthalten: `<lex>`-, `<orth>`- und `<tok>`-Elemente. Die Fallunterscheidung bewirkt, dass nur diejenigen `<lex>`-Elemente ausgegeben werden, die die zum gesamten Adjektiv-Chunk passenden syntaktischen Eigenschaften haben. Hier kommt es demnach auf die Kongruenz (2. Parameter der Funktion) und den Artikelgebrauch (3. Parameter) an. Da die `<tok>`-Elemente wiederum `<lex>`-Elemente enthalten, wird für sie die Funktion `my:copyDescAC()` erneut aufgerufen. Dabei bildet nun das `<tok>`-Element den ersten Parameter. Die beiden anderen werden aus dem ersten Funktionsaufruf übernommen. Die Funktion ist folglich rekursiv angelegt. Auf diese Weise können die morpho-syntaktischen Informationen der Bestandteile des Adjektiv-Chunks in Abhängigkeit der Kongruenz des Nominal-Chunks disambiguiert werden.

Die Funktion „my:delDoubleKgr()“

Um doppelte Vorkommen morpho-syntaktischer Informationen (`<lex>`-Elemente) zu verhindern, wurde die Funktion `my:delDoubleKgr()` definiert. Sie wird dort verwendet, wo es zu Wiederholungen der gleichen Information kommen kann. Dies ist v.a. bei der Kombination der Kongruenzinformationen der Bestandteile eines Nominal-Chunks der Fall. Daher wird die Funktion u.a. von der Funktion `my:unifyKgr()` verwendet.
Der Parameter der Funktion `my:delDoubleKgr()` ist ein Element oder temporärer Baum, in dem `<lex>`-Elemente vorhanden sind. Die doppelten Vorkommen gleicher Information werden durch folgende Schleife herausgefiltert:

```
<xsl:for-each select="$input//lex">
    <xsl:variable name="kgr1">
        <xsl:value-of select="@kgr"/>
    </xsl:variable>
    <xsl:if test="not(exists(preceding::lex[@kgr=$kgr1]))">
        <xsl:copy-of select="."/>
    </xsl:if>
</xsl:for-each>
```

Das Ergebnis des Funktionsaufrufs ist eine Liste von `<lex>`-Elementen. In dieser Liste gibt es keine zwei `<lex>`-Elemente, die für das Attribut `kgr` den gleichen Wert besitzen.

4.3 Präpositional-Chunks

4.3.1 Die Struktur von Präpositional-Chunks

Die Annotation von Präpositional-Chunks folgt einem relativ leicht nachzuvollziehenden Muster. Präpositionen besitzen ein morpho-syntaktisches Merkmal, das den von ihnen regierten Kasus festlegt. Folgende Regel kann für Präpositional-Chunks aufgestellt werden.

$$PC \rightarrow PR_{r='\#1'} \quad NC_{kgr='..-..-\#1'}$$

Der Kasus des Nominal-Chunks muss mit dem Kasus, der durch das Rektionsmerkmal der Präposition festgelegt wird, übereinstimmen. Diese Regel unterliegt allerdings der Einschränkung, dass keine postpositionalen Konstruktionen wie „den Weg entlang“ erfasst werden.

4.3.2 Tagging von Präpositional-Chunks: „pcTagging.xsl“

Die Annotation von Präpositional-Chunks erfolgt durch die Funktion `my:pcTagging()`, die in der Datei „pcTagging.xsl“ zu finden ist und wie die oben erläuterten Funktionen durch `<xsl:include>` in der Datei „syntTagging.xsl“ verfügbar ist.
Der rechte Teil der Regel im vorigen Abschnitt ist in XPath ausdrückbar:

```
self::tok[lex/@w='PR'] and
following-sibling::*[1][local-name()='chunk' and @type='nc']
```

Zu diesem Ausdruck ist die Rektions-Bedingung hinzuzufügen. Diese kann umgesetzt werden, indem erst überprüft wird, ob die Kongruenzinformationen des Nominal-Chunks den erforderlichen Kasus enthalten. Wiederum sind es Variablen, die diesen Vergleich

ermöglichen. Mittels `substring(@kgr, string-length(@kgr), 1)` kann für jedes einzelne `<lex>`-Element der angegebene Kasus extrahiert werden und in einer Variable zwischengespeichert werden. Auch der Wert des `r`-Attributs der `<lex>`-Elemente der Präposition werden in eine Variable gespeichert und danach mit den Kasus des Nominal-Chunks verglichen. Gibt es übereinstimmende Kasusinformationen, so wird die Erstellung eines Präpositional-Chunks begonnen. Das `<chunk type='pc'>`-Element erhält als erstes Unterelement die orthographische Repräsentation des gesamten Chunks. Diese besteht aus denen der einzelnen Bestandteile, also aus den Werten der jeweiligen `<orth>`-Elemente. Danach folgt das `<tok>`-Element, das die Präposition repräsentiert. Von ihm werden nur die `<lex>`-Elemente übernommen, die die Information mit der zum Nominal-Chunk passenden Rektion haben. Der Nominal-Chunk innerhalb des Präpositional-Chunks wird mit Hilfe der Funktion `my:copyDescNC()` erstellt. Hierbei können die morpho-syntaktischen Informationen des Nominal-Chunks und die seiner Bestandteile disambiguiert werden.

Die Hilfsfunktion „my:copyDescNC()“

Diese Funktion ähnelt der Funktion `my:copyDescAC()`, verwendet jedoch andere Bedingungen zur Kopie der `<lex>`-Elemente des Chunks und den `<lex>`-Elementen seiner Bestandteile. Für den Nominal-Chunk innerhalb eines Präpositional-Chunks ist nämlich die Information des Kasus von besonderer Bedeutung. So werden nach dem gleichen Prinzip wie in `my:copyDescAC()` die Unterelemente des Nominal-Chunks, allerdings nur in Abhängigkeit von deren Kasus rekursiv verarbeitet. Das Ergebnis ist ein Nominal-Chunk, dessen `<lex>`-Elemente auf beliebiger Ebene zu der Rektionsinformation der Präposition passen.

4.4 Das Ergebnis der syntaktischen Annotation

Durch die vorgestellten Annotationsschritte wurde der Ausgangstext mit der Annotation von Wörtern, Sätzen und schließlich Adjektiv-, Nominal- und Präpositional-Chunks angereichert. Zum Teil tauchen hierbei Probleme auf. So kann es z.B. durch fehlende Lexikoneinträge vorkommen, dass Chunks nicht annotiert werden können. Auch die Regeln zur Struktur von Chunks bieten Verbesserungspotential. Trotzdem wird die Mehrzahl der vorhandenen Chunks der erwähnten Typen erkannt. Als Erfolg würde ich die teilweise Disambiguierung der morpho-syntaktischen Eigenschaften von Nominal-Chunks und deren Bestandteilen werten. Man vergleiche nur die Informationen im Präpositional-Chunk „mit der andern Hand“ (siehe Anhang C) mit denen, die die einzelnen Wörter noch als Lexikoneinträge beinhalten. Das Wort „andern“ kann vor der syntaktischen Analyse mit 43 verschiedenen morpho-syntaktischen Kontexten verbunden werden. Nach der Analyse

konnte in dem genannten Beispiel genau ein Kontext als an dieser Stelle zutreffend annotiert werden.
Die hier vorgestellte syntaktische Annotation bietet natürlich die Möglichkeit der Weiterverarbeitung. Zunächst einmal wäre es sinnvoll, die annotierten Informationen auch visuell besser zugänglich zu machen. Hierzu habe ich ein weiteres Stylesheet definiert, das die XML-Annotation in ein HTML-Dokument transformiert. Auf diese Weise kann man sich die syntaktischen Informationen in einem Browser ansehen. Das Stylesheet „Output2HTML.xsl“ sowie diverse HTML-Dokumente befinden sich auf der beiliegenden CD-ROM im Ordner „Output“. Denkbar wären auch etwas aufwändigere Visualisierungen. So könnten z.B. mit Hilfe von SVG auf Grundlage der syntaktischen Annotation Ableitungsbäume erstellt werden.

Kapitel 5

Fazit

In den ersten beiden Kapiteln der Arbeit wurden die theoretischen Hintergründe zur automatischen syntaktischen Annotation mit XSLT aufgezeigt. Es ging hier zunächst darum, die Rolle von XML als Basis linguistischer Annotationsformate und die Verarbeitung von XML mit XSLT zu skizzieren. In diesem Zusammenhang hat sich herausgestellt, dass der XCES ein geeignetes Annotationsformat für linguistische (oder speziell syntaktische) Strukturen darstellt. Dieses Format habe ich für diese Arbeit geringfügig modifiziert.
Im nächsten Schritt (Kapitel 2) wurde die Reräsentation morpho-syntaktischer Informationen (Lexikoneinträge/Regeln) als Grundlage für die syntaktische Analyse diskutiert. Kontextfreie Grammatiken sind eine Möglichkeit, Regeln zum Aufbau syntaktischer Strukturen zu formulieren. Sie sind mit gut ausgearbeiteten Verarbeitungsparadigmen (Parsing, Kapitel 2.2) verbunden. Allerdings weisen sie auch einige Schwächen auf, die mit Hilfe von Formalismen aus anderen Grammatiktheorien wie den Unifikationsgrammatiken verbessert werden können. Ein Beispiel hierfür ist die Behandlung der Kongruenz im Deutschen.
Im zweiten Teil der Arbeit habe ich ein XSLT-Stylesheet vorgestellt, das den Ausgangstext „Hamlet“ mit flachen syntaktischen Strukturen (Adjektiv-, Nominal- und Präpositional-Chunks) annotiert. In verschiedenen Annotationsstufen wurden hierzu als erstes Wörter, dann Sätze und zuletzt die unterschiedlichen Chunk-Typen ausgezeichnet. Das Ergebnis der automatischen syntaktischen Annotation, ein XML-Dokument mit TEI- und XCES-Annotationen, wurde durch ein weiteres XSLT-Stylesheet in einem HTML-Dokument visualisiert.
Neben der einfachen Visualisierung der Chunk-Annotation sind aber auch tiefergehende Weiterverarbeitungen denkbar. Da sich die Annotation in dieser Arbeit, wie schon gesagt, auf einem relativ flachen Level bewegt, ist für die Erweiterung der ausgezeichneten syntaktischen Strukturen noch Spielraum vorhanden. Mit einer Erweiterung des Lexikons durch weitere Lexikoneinträge oder sogar weitergehende morpho-syntaktische Informatio-

nen, wie z.B. Subkategorisierungsrahmen von Verben, könnte eine Annotation komplexerer Konstituenten (evtl. inklusive rekursiver Strukturen) angestrebt werden. Der relativ kleine Umfang des Lexikons stellt ein generelles Problem für die syntaktische Analyse dar. Vom Umfang und der Genauigkeit des Lexikons hängen die Erfolgsaussichten einer syntaktischen Annotation ab. Hier sähe ich für eine Verbesserung oder Erweiterung der syntaktischen Analyse einen ersten Ansatzpunkt. Im Kontext einer Erweiterung wäre auch die Einbeziehung expliziter Theorien zu syntaktischen Strukturen zu erwägen. Zum Beispiel wäre durchaus denkbar, verschiedene Grammatiktheorien (z.B. constraint-basierte Grammatiken) oder die Theorie der topologischen Felder, in die Analyse miteinzubeziehen. Es wird deutlich, dass auf der Ebene der Syntax noch einige Erweiterungsmöglichkeiten bestehen. Vor allem wäre es wünschenswert auch komplexere Chunks automatisch annotieren zu können. Um z.B. „des Englischen mächtige“ als Adjektiv-Chunk zu erkennen, müsste der Aufruf der einzelnen Funktionen zur Chunk-Annotation deutlich dynamischer geregelt werden. Das bedeutet, es sollte keine festgelegte Reihenfolge der Funktionsanwendungen geben, sondern sie sollten in Abhängigkeit bestimmter Faktoren aufgerufen werden.
Das Ergebnis der bisherigen, relativ flachen syntaktischen Annotation kann aber nicht nur der weiteren syntaktischen Analyse dienen, sondern auch für andere Anwendungsfelder genutzt werden. Ich denke hier z.B. an das Gebiet des Information Retrieval, in dem flach annotierte syntaktische Strukturen helfen können, Texte inhaltlich zu klassifizieren. Hier ist eine Schnittstelle von Syntax und Semantik festzustellen. Syntaktische Strukturen können Anhaltspunkte zur automatischen Erkennung bzw. Klassifizierung semantischer Zusammenhänge bereit stellen. Von Vorteil kann nach meiner Ansicht in diesem Kontext die Annotation morpho-syntaktischer Informationen von Chunks, die über den Chunk-Typ hinausgehen, sein. Ein Beispiel hierfür ist die Kongruenzinformation ganzer Nominal-Chunks. Auf deren Grundlage lassen sich z.B. Subjekte und Objekte in Sätzen relativ leicht identifizieren,

Anhang

Anhang A

Notationskonventionen für das Lexikon

A.1 Abkürzungen

Die Notationskonventionen für syntaktische Merkmale von Lexikoneinträgen orientieren sich sehr stark an den von Wolfgang Lezius (1998) aufgestellten Tag Sets. In Kapitel 3.1.3 wurde deutlich, welche Veränderungen sinnvollerweise vorgenommen wurden. Die folgenden Tabellen sind im Wesentlichen bei Lezius (1998) wiederzufinden, jedoch nicht eins zu eins übernommen.

Lexikoneinträge können durch die nachstehenden Merkmale syntaktisch spezifiziert werden. Das bedeutet, dass die Attribute eines `<lex>`-Elements (syntaktische Eigenschaften) die folgenden Bezeichnungen haben können. Bei den Attributnamen handelt es sich stets um Kleinbuchstaben.

Syntaktische Merkmale	
Kürzel	**Bedeutung**
a	Artikelgebrauch
b	Gebrauch
kgr	Kongruenz
p	Person
r	Rektion
t	Typ
w	Wortart

In der zweiten Tabelle sind die möglichen Attributwerte aufgeführt. Wie diese Werte zueinander in Beziehung stehen, also welche Attribute und Werte beispielsweise für bestimmte Wortarten angemessen sind, werden die Tabellen im Abschnitt „Wortklassensysteme“ verdeutlichen.

Wortarten		weitere Werte der syntaktischen Merkmale	
Kürzel	**Bedeutung**	**Kürzel**	**Bedeutung**
AK	Abkürzung	1	Nominativ
A	Adjektiv	2	Genitiv
B	Adverb	3	Dativ
AR	Artikel	4	Akkusativ
DE	Demonstrativpronomen	C	2. (Person)
E	Eigenname	D	definit
I	Indefinitpronomen	EZ	erweiterter Infinitiv mit zu
J	Interjektion	F	feminin
IR	Interrogativpronomen	G	1. (Person)
K	Konjunktion	H	3. (Person)
P1	Partizip1	I	indefinit
P2	Partizip2	IN	Infinitiv
PE	Personalpronomen	M	maskulin
PO	Possessivpronomen	MO	modal
PR	Präposition	N	neutrum
R	Reflexivpronomen	NE	nebenordnend
RL	Relativpronomen	NO	ohne Genus
S	Substantiv	O	pronominal
V	Verb	P	plural
Y	keine Wortart gefunden	Q	proportional
Z	Zahlwort	SI	singular
		SO	alleinstehend
		T	attributiv
		U	unterordnend
		W	vergleichend
		X	auxiliar

A.2 Wortklassensysteme

Die Wortklassensysteme stellen in tabellarischer Notation den Gebrauch von unterschiedlichen Attributen bei verschiedenen Wortarten von Lexikoneinträgen dar. Die Attributnamen sind im Kopf jeder Tabelle angegeben. Darunter stehen die möglichen Attributwerte. Die Verwendung der in Klammern angegebenen Attribute ist optional, d.h. ihre Angabe ist nicht für jeden Lexikoneintrag der jeweiligen Wortart sinnvoll.
Die kursiv geschriebenen Attribute (*n* [Numerus], *g* [Genus], *k* [Kasus] und *f* [Form]) gehören eigentlich nicht zum verwendeten Tag Set. Sie verdeutlichen lediglich den komplexen Aufbau des Kongruenz-Attributs (`kgr`), dessen Wert aus den Werten des Numerus, Genus und Kasus eines Wortes zusammengesetzt ist (Bsp.: kgr='SI-F-2'). Diese Informationen können allerdings auch unterspezifiziert sein (Bsp.: kgr='SI- -1').
Ein exemplarischer Lexikoneintrag wurde in Kapitel 3.1.3 vorgestellt

Substantive und Eigennamen

w	kgr		
	(*n*)	(*g*)	(*k*)
S	SI	M	1
E	P	F	2
		N	3
		NO	4

Entgegen den Annahmen Lezius, der in Morphy Eigennamen und Substantive in unterschiedliche Klassen fasst, werden diese hier nicht gesondert behandelt. Für die syntaktische Analyse spielt z.B. die Art von Eigennamen (Orte, Menschen, ...) eine zu vernachlässigende Rolle. Das einzige zusätzliche Attribut, das diesen Wortarten zugeordnet wird, ist `kgr`, das Kongruenzinformationen beinhaltet.

Adjektive und Partizipien

w	kgr			a	(b)
	(*n*)	(*g*)	(*k*)		
A	SI	M	1	D	B
P1	P	F	2	I	
P2		N	3	SO	
			4		

Adjektive und Partizipien können einerseits attributiv gebraucht werden. In diesem Fall werden ihnen Informationen über die Kongruenz, sowie den Artikelgebrauch mitgegeben. Andererseits können sie aber auch in adverbialem Gebrauch stehen. Dann ist nur das Attribut `b` vorhanden, dessen fester Wert den adverbialen Gebrauch darstellt.

Artikel

w	kgr			t
	(n)	(g)	(k)	
AR	SI	M	1	D
	P	F	2	I
		N	3	
			4	

Die morpho-syntaktischen Informationen von Artikeln umfassen Kongruenz und Artikeltyp. Mithilfe des Artikeltyps können sie mit dem Artikelgebrauch von Adjektiven und Partizipien verglichen werden.

Verben

w	kgr		(p)	(t)
	(n)	(f)		
V	SI	P1	C	MO
	P	P2	G	X
		IN	H	
		IM		
		EZ		

Elemente, in denen das Attribut `w` mit dem Wert `V` spezifiziert ist (Verben), enthalten Informationen über Kongruenz, Person und Typ des repräsentierten Verbs. Das `kgr`-Attribut hat hier ein anderes Format als z.B. bei Substantiven und Adjektiven. Da Verben in dieser Arbeit nicht als Bestandtteile übergeordneter Konstituenten vorkommen können, ist das Format nicht von großer Bedeutung.

Pronomen

w	kgr			b	p[a]
	(n)	(g)	(k)		
DE	SI	M	1	O	
IN	P	F	2	T	
IR		N	3		
RL			4		
PO					
PE					C
R					G
					H

[a] für PE optional

Alle Arten von Pronomen werden durch das `kgr`-Attribut mit Informationen zur Kongruenz versehen. Für Personal- und Reflexivpronomen ist desweiteren ein Attribut zur Person (`p`) spezifiziert. Die anderen oben aufgeführten Arten von Pronomen besitzen dieses Attribut nicht, sondern verfügen über eine Information zu deren Gebrauch. Dieser kann pronominal (`b='O'`) oder attributiv (`b='T'`) sein.

weitere Wortarten

w	t[a]	r
AK		
B	O	
J		
K	U NB Q IN W	
PR		1 2 3 4
Z		

[a] für B optional

Diese Tabelle verdeutlicht die Attribute der etwas weniger komplexen Elemente. Die Elemente, die Abkürzungen (`AK`), Interjektionen (`J`) und Zahlwörter (`Z`) repräsentieren haben keine Attribute außer `w` für die Wortart. Konjunktionen und Adverben werden durch das Attribut `t` (Typ) näher beschrieben. Präpositionen werden mit der Information über den von ihnen regierten Kasus versehen.

Anhang B

Hamlet im TEI-Format

Die im Format der TEI.drama-DTD annotierte Hamlet-Ausgabe (in „hamlet.xml“) hat folgendes Erscheinungsbild:

```
<div2 type="scene" n="4">
    <stage type="" n="12" f1nr="" q1nr="" q2nr="" moby="">
        Die Terrasse
    </stage>
    <stage type="" n="13" f1nr="" q1nr="" q2nr="" moby="">
        Hamlet, Horatio und Marcellus treten auf.
    </stage>
    <sp n="162" f1nr="" q1nr="" q2nr="" moby="" who="">
        <speaker>HAMLET</speaker>
        <l n="643" f1nr="" q1nr="" q2nr="" moby="">
            Die Luft geht scharf, es ist entsetzlich kalt.
        </l>
    </sp>
    <sp n="163" f1nr="" q1nr="" q2nr="" moby="" who="">
        <speaker>HORATIO</speaker>
        <l n="644" f1nr="" q1nr="" q2nr="" moby="">
            ’s ist eine schneidende und strenge Luft.
        </l>
    </sp>
    ...
</div2>
```

Anhang C

Syntaktisch annotierter Satz

Im Folgenden wird ein syntaktisch annotierter Satz aufgeführt. Das Beispiel ist natürlich nicht ganz zufällig ausgewählt. Man wird nicht lange suchen müssen, um im Ergebnis der automatischen syntaktischen Annotation weniger erfolgreiche Auszeichnungen zu finden. Auch in diesem Beispiel ist bereits eine Schwachstelle zu bemerken. Das Adverb „so“ wird dem Adjektiv-Chunk „so überm Auge“ zugeordnet. Diese Zuordnung verdeckt jedoch die Ambiguität, die durch das „so“ gegeben ist.

```
<s>
    <tok>
        <orth>Und</orth>
        <lex w="K" t="NB"/>
    </tok>
    <chunk type="pc">
        <orth>mit der andern Hand</orth>
        <tok>
            <orth>mit</orth>
            <lex w="PR" r="3"/>
        </tok>
        <chunk type="nc">
            <lex kgr="SI-F-3"/>
            <tok>
                <orth>der</orth>
                <lex w="AR" kgr="SI-F-3" t="D"/>
            </tok>
            <chunk>
                <orth>andern</orth>
```

```
                <lex kgr="SI-F-3" a="D"/>
                <tok>
                    <orth>andern</orth>
                    <lex w="A" kgr="SI-F-3" a="D"/>
                </tok>
            </chunk>
            <tok>
                <orth>Hand</orth>
                <lex w="S" kgr="SI-F-3"/>
            </tok>
        </chunk>
    </chunk>
    <chunk type="nc">
        <orth>so uberm Auge</orth>
        <lex kgr="SI-N-3"/>
        <chunk type="ac">
            <orth>so uberm</orth>
            <lex kgr="SI-N-3" a="SO"/>
            <tok>
                <orth>so</orth>
            </tok>
            <tok>
                <orth>uberm</orth>
                <lex w="A" kgr="SI-N-3" a="SO"/>
            </tok>
        </chunk>
        <tok>
            <orth>Auge</orth>
            <lex w="S" kgr="SI-N-3"/>
        </tok>
    </chunk>
    <l n="1082" f1nr="" q1nr="" q2nr="" moby=""/>
    <tok>
        <orth>Betrachtet</orth>
        <lex w="V"/>
        <lex w="V" p="G" kgr="P--"/>
        <lex w="V" kgr="P-IM-"/>
```

```
        <lex w="V" kgr="-P2-"/>
        <lex w="P2" b="B"/>
    </tok>
    <sg>
        <orth>'</orth>
    </sg>
    <chunk type="nc">
        <orth>er</orth>
        <lex kgr="SI-M-1"/>
        <tok>
            <orth>er</orth>
            <lex w="PE" kgr="SI-M-1" p="H"/>
        </tok>
    </chunk>
    <tok>
        <orth>so</orth>
        <lex w="K" t="Q"/>
        <lex w="K" t="U"/>
        <lex w="B"/>
    </tok>
    <tok>
        <orth>prufend</orth>
        <lex w="V" kgr="-P1-"/>
        <lex w="P1" b="B"/>
    </tok>
    <chunk type="nc">
        <orth>mein Gesicht</orth>
        <lex kgr="SI-N-1"/>
        <lex kgr="SI-N-3"/>
        <lex kgr="SI-N-4"/>
        <tok>
            <orth>mein</orth>
            <lex w="PO" kgr="SI-N-1" b="T"/>
            <lex w="PO" kgr="SI-N-4" b="T"/>
        </tok>
        <tok>
            <orth>Gesicht</orth>
```

```
            <lex w="S" kgr="SI-N-1"/>
            <lex w="S" kgr="SI-N-3"/>
            <lex w="S" kgr="SI-N-4"/>
        </tok>
    </chunk>
    <sg>
        <orth>,</orth>
    </sg>
    <l n="1083" f1nr="" q1nr="" q2nr="" moby=""/>
    <tok>
        <orth>Als</orth>
        <lex w="PR" r="1"/>
        <lex w="K" t="W"/>
        <lex w="K" t="U"/>
        <lex w="K" t="NB"/>
        <lex w="PR" r="3"/>
        <lex w="PR" r="4"/>
    </tok>
    <tok>
        <orth>wollt</orth>
        <lex w="V" kgr="P--" t="MO" p="G"/>
    </tok>
    <chunk type="nc">
        <orth>er</orth>
        <lex kgr="SI-M-1"/>
        <tok>
            <orth partOf="ers">er</orth>
            <lex w="PE" kgr="SI-M-1" p="H"/>
        </tok>
    </chunk>
    <chunk type="nc">
        <orth>s</orth>
        <lex kgr="SI-N-1"/>
        <lex kgr="SI-N-4"/>
        <tok>
            <orth partOf="ers">s</orth>
            <lex w="PE" kgr="SI-N-1" p="H"/>
```

```
            <lex w="PE" kgr="SI-N-4" p="H"/>
        </tok>
    </chunk>
    <tok>
        <orth>zeichnen</orth>
        <lex w="V" kgr="P--" p="C"/>
        <lex w="V"/>
        <lex w="V" kgr="-IN-"/>
    </tok>
    <snSg>
        <orth>.</orth>
    </snSg>
</s>
```

Literaturverzeichnis

Abney, Steven P. (1991): *Parsing by Chunks.* In: Berwick, Robert C./Abney, Stephen P./Tenny, Carol (Hrsg.): Principle-Based Parsing: Computation and Psycholinguistics. Dordrecht/Boston/London: Kluwer Academic Publishers, S. 257-278.

Bresnan, Joan et al. (1982): *Cross-serial dependencies in Dutch.* In: Linguistic Inquiry, Vol. 13, Nr. 4, S. 613-635.

Eisenberg, Peter (2004): Grundriß der deutschen Grammatik. Band 2: Der Satz. 2. überarbeitete und aktualisierte Auflage, Metzler: Stuttgart/Weimar.

Extensible Markup Language (2004): *Extensible Markup Language (XML) Version 1.1.* Bray, Tim et al. (Hrsg.): W3C Recommendation, 04 February 2004, edited in place 15 April 2004.
(http://www.w3.org/TR/2004/REC-xml11-20040204/ [24.02.2006])

Extensible Stylesheet Language (2001): *Extensible Stylesheet Language (XSL) Version 1.0.* Adler, Sharon et al. (Hrsg.): W3C Recommendation, 15 October 2001.
(http://www.w3.org/TR/2001/REC-xsl-20011015/ [24.02.2006])

Halama, André (2004): *Flache Satzverarbeitung.* In: Carstensen, Kai-Uwe et al. (Hrsg.): Computerlinguistik und Sprachtechnologie. 2. Auflage. München: Elsevier, S. 218-231.

Hentschel, Elke/Weydt, Harald (2003): *Handbuch der deutschen Grammatik.* 3. völlig neu bearbeitete Auflage. Berlin/New York: De Gruyter.

Hinrichs, Erhard W./Kübler, Sandra/Müller, Frank H./Ule, Tylman (o.A.): *A Hybrid Architecture for Robust Parsing of German.*
(http://www.sfb441.uni-tuebingen,de/a1/Publikationen/robustpars.pdf [10.03.2006])

Jannidis, Fotis (1997): *TEI in der Praxis.*
(http://computerphilologie.uni-muenchen.de/praxis/teiprax.html [10.03.2006])

Jurafsky, Daniel/Martin, James H. (2000): *Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* New Jersey: Prentice-Hall.

Kay, Michael (2004): *XSLT 2.0 Programmers Reference.* 3. Auflage. Indianapolis: Wiley Publishing.

Kolb, Peter (2004): *Graphentheorie und Merkmalsstrukturen.* In: Carstensen, Kai-Uwe et al. (Hrsg.): Computerlinguistik und Sprachtechnologie. 2. Auflage. München: Elsevier, S. 91-110.

Langer, Hagen (2001): *Parsing-Experimente: praxisorientierte Untersuchungen zur automatischen Analyse des Deutschen.* Frankfurt a.M.: Peter Lang Europäischer Verlag der Wissenschaften.

Langer, Hagen (2004): *Syntax und Parsing.* In: Carstensen, Kai-Uwe et al. (Hrsg.): Computerlinguistik und Sprachtechnologie. 2. Aufl., München: Elsevier, S. 232-275.

Lezius, Wolfgang (1998): *Die Wortklassensysteme von Morphy.* (http://www.wolfganglezius.de/morphy/wklassen.pdf [12.01.2006])

Lezius, Wolfgang (1999): *MORPHY für Windows. Ein Tool zur Deutschen Morphologie – Benutzerhandbuch –.* (http://www.wolfganglezius.de/morphy/handbuch.pdf [10.03.2006])

Lezius, Wolfgang (2002): *Ein Suchwerkzeug für syntaktische annotierte Textkorpora.* Dissertation, Institut für Maschinelle Sprachverarbeitung der Universität Stuttgart.

Mehl, Stephan/Langer, Hagen/Volk, Martin (1998): *Statistische Verfahren zur Zuordnung von Präpositionalphrasen.* In: Bernhard Schröder et al. (Hrsg.): Computer, Linguistik und Phonetik zwischen Sprache und Sprechen. Proc. KONVENS 98, Frankfurt: Peter Lang, S. 97-110.

Müller, Frank Henrik (2002): *Shallow-Parsing Stylebook for German. Technical Report.* Seminar für Sprachwissenschaft, Universität Tübingen. (http://www.sfs.uni-tuebingen.de/dereko/stylebook.pdf [23.03.2006])

Müller, Stefan (1994): *Prolog und Computerlinguistik.* (http://www.cl.uni-bremen.de/~stefan/PS/prolog.pdf [12.03.2006])

Pulman, Stephen G. (1991): *Basic Parsing Techniques: an introductory survey.* (http://www.ling-phil.ox.ac.uk/people/staff/pulman/pdfpapers/parsing.pdf [10.03.2006])

Russell, Stuart/Norvig, Peter (2003): *Künstliche Intelligenz. Ein moderner Ansatz.* 2. Aufl., München: Pearson Studium.

Sailer, Manfred/Richter, Frank (2001): *Eine XML-Kodierung für AVM-Beschreibungen.* (http://www.sfb441.uni-tuebingen.de/a1/Publikationen/GLDV2001-sailer.pdf [10.03.2006])

Witt, Andreas (1999): *DSSSL zur Verarbeitung linguistischer Korpora.* In: Gippert, J. (Hrsg.): Multilinguale Corpora: Codierung, Strukturierung, Analyse. Prag: Enigma, S. 107-114. (http://coli.lili.uni-bielefeld.de/~andreas/dsssl/witt.pdf [10.03.2006])

Witt, Andreas/Müller, Stefan (2002): *Grundlagen für den Computereinsatz in der Linguistik: Attribute, Werte, Unifikation.* In: Müller, Horst M. (Hrsg.): Arbeitsbuch Linguistik. Paderborn: Schöningh, S. 425-442.

XML Path Language (2005): *XML Path Language (XPath) Version 2.0.* Berglund, Anders et al. (Hrsg.): W3C Candidate Recommendation, 3 November 2005. (http://www.w3.org/TR/2005/CR-xpath20-20051103/ [24.02.2006])

XSL Transformations (2005): *XSL Transformations (XSLT) Version 2.0.* Kay, Michael (Hrsg.): W3C Candidate Recommendation, 3 November 2005. (http://www.w3.org/TR/2005/CR-xslt20-20051103/ [24.02.2006])

Erklärung: Die/der Unterzeichnete versichert, dass er/sie die vorliegende schriftliche Arbeit selbstständig verfasst und keine anderen als die von ihm/ihr angegebenen Hilfsmittel benutzt hat. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinne nach entnommen sind, wurden in jedem Fall unter Angabe der Quellen kenntlich gemacht. Dies gilt auch für beigegebene Zeichnungen, bildliche Darstellungen, Skizzen und dergleichen.

Der/dem Unterzeichneten ist bewusst, dass jedes Zuwiderhandeln (Einreichen einer Arbeit, die wörtlich oder nahezu wörtlich, ganz oder zu Teilen aus einer Arbeit oder mehreren Arbeiten [publiziert im Internet, in Zeitschriften, Monographien etc.] anderer übernommen ist) als Täuschungsversuch (siehe §18 BPO) gelten kann, der die Bewertung der Arbeit mit „nicht ausreichend“ zur Folge hat.

Ort,Datum	Unterschrift