Dokumentation

Aufbereitung von ingenieurwissenschaftlichen Zeitschriftenartikeln für Gingko (Geschriebenes ingenieurwissenschaftliches Korpus)

Paul Knötgen – B. Sc. Wirtschaftsingenieurwesen, Angewandte Informatik und Formale Beschreibungsverfahren (AIFB) des Karlsruher Instituts für Technologie
Annette Portmann – M.A. Deutsch als Fremd- und Zweitsprache, Herder-Institut der Universität Leipzig

Version 2.0
Stand: 07. Mai 2021

Inhaltsverzeichnis

Listing-Verzeichnis

Tabellenverzeichnis

1 Einleitung

Das Projekt *Muster in der Sprache der Ingenieurwissenschaften: Gingko (Geschriebenes ingenieurwissenschaftliches Korpus)* verfolgt das Ziel, Muster in den Fachsprache(n) der Ingenieurwissenschaften am Beispiel der Automobiltechnik systematisch zu erfassen und linguistisch zu beschreiben. Die Grundlage dafür soll das Korpus *Gingko (Geschriebenes ingenieurwissenschaftliches Korpus)* darstellen. Für das Korpus werden 2498 Artikel aus den Zeitschriften *Automobiltechnische Zeitschrift (ATZ)* und *Motortechnische Zeitschrift (MTZ)* des Springer-Verlags verwendet. Die Artikel stammen aus den Jahrgängen 2007 bis 2016, umfassen insgesamt 4.667.656 Token.

Die vorliegende Korpusdokumentation soll die Korpusdaten beschreiben und die durchgeführten Operationen zur Korpuserstellung nachvollziehbar machen. Sie beginnt mit einem Kapitel zu den Daten des Korpus: die vom Springer-Verlag bereitgestellten Rohdaten (Zeitschriftenartikel im PDF- und XML-Format), die Metadaten für die Subkorpora und für die einzelnen Texte sowie die Annotationen zur Dokumentstruktur und die Lemma-, POS- und Satzannotation. Kapitel 3 widmet sich der Datenaufbereitung. Hierfür wurden die Daten zunächst in eine einheitliche XML-ähnliche Struktur überführt und mit Metadaten und Annotationen versehen. Der Import des Korpus ins Such- und Visualisierungstool ANNIS (Kapitel 3.5) ermöglichte das Herausfiltern von Schwachstellen der Lemmaannotation. Somit wurde zuletzt noch eine zweite verbesserte Lemmatisierung angelegt (Kapitel 4). Der Anhang der Korpusdokumentation beinhaltet eine Liste der genutzten Tools sowie längere Skripte zur Umformatierung der XML-Dateien und der Erstellung der zweiten Lemmaannotation.

2 Beschreibung der Daten

Während der Datenaufbereitung wurden alle Zeitschriftenartikel, die als Rohdaten im PDF und im XML-Schema JATS vorlagen, in ein einheitliches xml-Format gebracht. Zusätzlich wurden die XML-Skripte als eine Datenbank für das Such- und Visualisierungstool ANNIS (Krause & Zeldes, 2016) abgelegt.

2.1 Rohdaten

Die 2498 Artikel aus den Zeitschriften *Automobiltechnische Zeitschrift (ATZ)* und *Motortechnische Zeitschrift (MTZ)* stellte der Springer-Verlag dem *Gingko*-Projekt zur Verfügung. Insgesamt umfassten die bereitgestellten Daten Artikel aus den Jahren 2007 bis 2016 in zwei unterschiedlichen Formaten: PDF und XML. Die Artikel aus den Jahren 2007 bis 2010 wurden dem Projekt im PDF-Format übergeben, alle neueren Artikel aus den Jahren 2011 bis 2016 im XML-Format. Die beiden Gruppen der Artikel mussten daher mit verschiedenen Workflows aufbereitet werden. Ein wichtiger Unterschied ist, dass für die Jahrgänge 2011 bis 2016 eine zweite Lemmaannotation *lemma_cor* (Kapitel 4) eingepflegt wurde und für die Jahrgänge 2007 bis 2010 nicht. Die Korpusdateien sind so anhand der Metadatenvariable *lemma-unknown-corrected* (Kapitel 2.2.2) unterscheidbar.

Das besondere **des XML**-Formats ist, dass es sowohl von Maschinen als auch von Menschen gelesen werden kann und es Metainformationen zum Text abspeichert. Insgesamt wurden 897 XML-Dateien der Zeitschrift ATZ und 861 Dateien der Zeitschrift MTZ für die Jahrgänge 2011 bis 2016 zur Verfügung gestellt. Die Artikel lagen im XML-Schema JATS *(Journal Publishing Tag Set)* vor, welches für journalistische Texte benutzt wird. In diesem Schema werden neben dem Fließtext beispielsweise Metadaten zur Veröffentlichung und Informationen zu Textsektionen, Abbildungen, Formeln oder Tabellen hinterlegt (https://jats.nlm.nih.gov/publishing/). Die Bearbeitung dieser Dateien wird ab dem Kapitel 3.2 weiter beschrieben.

Für die Artikel aus den Jahren 2007 bis 2011 existierten leider keine XML-Dateien und stattdessen nur **PDFs**. PDF-Dateien sind nur bedingt maschinenlesbar, daher wurden die Dateien ebenfalls in XML überführt (Kapitel 3.1).

2.2 Metadaten

Das folgende Kapitel beschreibt die Metadaten, wie sie in den finalen Korpusdateien vorliegen.

2.2.1 Korpusbezogene Metadaten

Tabelle 1 bildet die korpusbezogenen Metadaten ab. Sie sind in einer separaten Text-Datei mit der Endung `.meta` abgelegt, werden bei der Konvertierung in die ANNIS-Datenbank integriert (Kapitel 3.6.2) und sind über den Korpusinfo-Knopf in ANNIS einsehbar.

Variable	**Erläuterung**	**Attribut**
a_project_name	Name des Erstellungsprojekts	Muster in der Sprache der Ingenieurwissenschaften
b_project_place	Ort des Erstellungsprojekts	Greifswald (2017-2019), Leipzig (2020-2021)
c_project_ institution	Einrichtung des Erstellungsprojekts	Universität Greifwald, Institut für Deutsche Philologie (2017-2019); Universität Leipzig, Herder-Institut (2020-2021)
d_project_type	Art des Erstellungsprojekts	Drittmittelprojekt; gefördert durch die DFG; AOBJ: 632723
e_project_head	Leiter/-in des Erstellungsprojekts	Jun.-Prof. Dr. Antje Heine (2017-2019), Prof. Dr. Christian Fandrych (2020-2021)
f_project_ duration	Laufzeit des Erstellungsprojekts	2017-2021
g_project_ description	Projektbeschreibung	Gingko (Geschriebenes ingenieurwissenschaftliches Korpus) besteht aus 2498 wissenschaftlichen Artikeln der Zeitschriften Automobiltechnische Zeitschrift (ATZ) und Motortechnische Zeitschrift (MTZ) der Jahrgänge 2007-2016 und umfasst insgesamt

		4.667.656 Tokens. Es ist im Rahmen des Forschungsprojektes „Muster in der Sprache der Ingenieurwissenschaften“ entstanden. Das Projekt hat das Ziel, Muster in der Sprache der Ingenieurwissenschaften (am Beispiel der Automobiltechnik) systematisch zu erfassen und zu beschreiben.
h_corpus_title	Titel des Korpus	Geschriebenes ingenieurwissenschaftliches Korpus
i_corpus_ acronym	Akronym des Korpus	Gingko
j_corpus_ publication	Publikation des Erstellungsprojekts	Schirrmeister, L., Rummel, M., Heine, A., Suppus, N. & Mendoza Sánchez, B. (2021). Gingko – ein Korpus der ingenieurwissenschaftlichen Sprache. Deusch als Fremdsprache 58.
k_corpus_head	Verantwortliche/-r des Korpus	Prof. Dr. Christian Fandrych
l_corpus_ contributors	Mitwirkende des Korpus	2017-2019: Antje Heine, Marlene Rummel, Lars Schirrmeister, Nina Suppus, Sarah Brauer, Rebekka Fricke, Anne Hertel, Marcel Knorn, Bárbara Mendoza Sanchez, Paul Knötgen, Agnes Koschmider; 2020-2021: Christian Fandrych, Cordula Meißner, Annette Portmann, Lars Schirrmeister, Franziska Wallner
m_corpus_version	Version des Korpus	2.0
n_corpus_ language	Sprache des Korpus	Deutsch
o_corpus_ sizeTexts	Anzahl der Transkripte	2.498
p_corpus_ sizeToken	Anzahl der Token	4.667.656
q_corpus_mode	Modalität des Korpus	geschrieben
r_corpus_textType	Textart	wissenschaftliche Artikel, 426 journalistische Texte
s_corpus_field	Sprachliche Domäne	Wissenschafts- und Fachsprache
t_corpus_ annotation	Annotationensebenen des Korpus	Lemma, POS, Satzspannen, Textsorte, Dokumentstruktur

u_corpus_ posTagset	POS-Tagset des Korpus	stts
v_source_ journalTitle	Titel der Zeitschriften, die die Korpusgrundlage bilden	ATZ - Automobiltechnische Zeitschrift und MTZ - Motortechnische Zeitschrift
w_source_years	Jahrgänge, die im Korpus enthalten sind	2007-2016
x_source_publisher	Verlag der Zeitschriften	Springer Fachmedien Wiesbaden
y_source_ issuesPerYear	Anzahl der Zeitschriftausgaben pro Jahr	11

Tabelle 1: Korpusbezogene Metadaten

2.2.2 Textbezogene Metadaten

Die textbezogenen Metadaten halten Autor/-in, Titel und Daten zur Veröffentlichung des Artikels fest.

Variable	**Beschreibung**
article-title	Titel
author	Autor, wenn nur ein Autor vorhanden
author1, author2, …	Autoren werden durchnummeriert, wenn mehrere Autoren vorhanden
doi	Vollständiger Digital Object Identifier (z. B. "10.1365/s35148-011-0076-2")
firstpage	Seitenzahl, bei der der Artikel beginnt
id	Der Identifier, der im DeReKo über COSMASII oder KorAp angezeigt wird (z.B. „MTZ11/APR.00001“)
pub-date	Veröffentlichungsdatum (z. B. „1.2007“, „11.2014“)
issue	Heftnummer (z. B. „4“, „7-8“)
journal-title	Titel der Zeitschrift („ATZ - Automobiltechnische Zeitschrift“ oder „MTZ - Motortechnische Zeitschrift“)
lastpage	Seitenzahl, bei der der Artikel endet
lemma-unknown-corrected	„yes“ zeigt an, dass der Text mit einer zweiten korrigierten Lemmaannotation versehen ist (Kapitel 4), ein Text mit dem

	Attribut „no“ stammt aus den Jahrgängen 2007 bis 2010 und hat nur die Lemmaannotation des TreeTaggers
month	Monat der Veröffentlichung
subtitle	Untertitel, wenn vorhanden
text-type	Textsorte; die Variable hat nur einen Wert, wenn es sich um einen eher journalistischen Text handelt, bei eher wissenschaftssprachlichen Texten ist diese Variable leer (siehe Kapitel 2.2.3)
token	Tokenzahl des Textes
volume	Jahrgang (z. B. „110“)
year	Jahr der Veröffentlichung

Tabelle 2: Variablen der textbezogenen Metadaten

Die textbezogenen Metadaten sind als Attribute des Wurzelelements der XML-Skripte angegeben und gleich am Anfang des Skriptes zu finden. In ANNIS können Suchanfragen nach diesen Metadaten eingegrenzt werden (Suchoperator `& meta::` in ANNIS3 bzw. `@*` in ANNIS4).

```
<article article-title="Die Einspritzungals Schlüsseltechnik" author=""
doi="10.1365/s35146-011-0062-y" firstpage="250" id="MTZ11/APR.00001"
issue="4" journal-title="MTZ - Motortechnische Zeitschrift"
lastpage="251" lemma-unknown-corrected="yes" month="4" subtitle="" text-
type="Titelthema" token="450" volume="72" year="2011">
```

Listing 1: Metadaten im Wurzelelement des XML-Skripts

2.2.3 Variable zur Textsorte *text-type*

Auch wenn der Verlag dem Projekt vornehmlich wissenschaftssprachliche Texte übergeben hat, wurden während der Projektlaufzeit in den Jahrgängen 2011 bis 2016 426 kürzere Texte mit eher journalistischer Prägung gefunden. Sie umfassen beispielsweise Meldungen, Interviews und die Zeitschrift gliedernde Texte, machen insgesamt 249.766 Tokens aus und werden mit einer Metadatenvariable zur Textsorte ausgewiesen. Häufig bleibt die Variable *author* für diese Texte leer.

Wert	**Beschreibung**	**Tokenzahl gesamt**	**Anzahl Texte**	**Durchschn. Textlänge**
Aktuell	Kurzmeldungen zu Entwicklung in der Forschung und Wirtschaft und zu Tagungen	12006	19	632
Anzeige	Werbung von Springer, Anzeige zu gewonnenem Wettbewerb, Stellenanzeige	1534	3	511
Bild des Monats	Bild des Monats mit Kurzmeldung zu aktuellen Entwicklungen	2326	12	194

Bücher	Vorstellung von Buchneuerscheinungen	3525	7	504
Editorial	Editorial der Chefredaktion zu Inhalten und Entwicklungen der Zeitschrift	3959	11	360
FVV-Berichte	Meist zwei kurze Berichte der Forschungsvereinigung Verbrennungskraft Maschinen e.V.	4715	8	589
Gastkom-mentar	Gastkommentar zu freien Themen am Ende der Zeitschrift	42276	102	414
Historie	Zusammenfassung der Berichterstattung bezogen auf ein bestimmtes Thema für alle bisher erschienenen Zeitschriften	2950	5	590
Interview	Interview	20692	14	1478
Patente	Vorstellung aktueller Patente	4171	8	521
Personen + Unterneh-men	Kurzmeldungen zu Personen aus der Wirtschaft und Unternehmen	27638	25	1106
Porträt	Porträts von Personen aus der Automobilbranche	6238	12	520
Produkte	Vorstellung neuer Produkte	28174	26	1084
Special	Ankündigung eines thematischen Sonderteils mit drei bis sieben zugehörigen Artikeln	400	4	100
Tagungs-bericht	Berichte über Tagungen wie die Internationale Automobil-Ausstellung oder die ATZlive-Fachtagung oder	29449	28	1075
Titelthema	Ankündigung des Titelthemas mit ca. drei bis zehn zugehörigen Artikeln	58170	135	431
Vorschau	Vorstellung des Titelthemas der kommenden Zeitschrift	1543	7	221

Tabelle 3: Werte zur Variable text-type *und Beschreibung der Textsorten*

2.3 Annotationen

Das folgende Kapitel beschreibt die Annotationen, wie sie in den finalen Korpusdateien vorliegen.

2.3.1 Annotationen zur Dokumentstruktur

Die Annotationen zur Dokumentstruktur markieren Titel, Abschnitte, Zwischenüberschriften, Absätze, Abbildungen, Tabellen, Formeln und Danksagungen. Die Darstellung der Annotationen in ANNIS und in den XML-Struktur wird in Kapitel 2.4 genauer erläutert.

Spurname (ANNIS)	**Tags (ANNIS)**	**XML-Start-Tag**	**Beschreibung der Textebene *tok* oder Beispiel**
articleTitle	articleTitle	`<article-title articleTitle= "articleTitle">`	Titel des Artikels
subtitle	subtitle	`<subtitle subtitle="subtitle">`	Untertitel des Artikels
sec	sec0, sec1, sec2, …	`<sec sec="sec0">`	ein Abschnitt beginnt i. d. R. mit einer Zwischenüberschrift, Abstracts sind Beispiele für Abschnitte ohne Zwischenüberschrift
subheading	subheading	`<title subheading= "subheading">`	Zwischenüberschrift
paragraph	p	`<paragraph p="p">`	Absatz im Artikel
fig	fig1, fig2, …	`<fig fig="fig1">`	Label einer Abbildung und Bildunterschrift z. B. `Bild 1 Bild 1 : Architektur- und Funktionsentwurf im V-Modell`
table-wrap	tab1, tab2, …	`<table-wrap tab="tab1>`	Label der Tabelle, Tabellenunterschrift und Inhalte der Tabellenzellen aneinandergereiht `Tabelle 2 Tabelle 2 Gemessene Blendbeleuchtungsstärke am Auge der Testpersonen Automobil der Testperson (Adaptation) Blendendes Automobil Xenon D1 (Reflexion) Halogen H7 (Projektion) Halogen H7 (Reflexion) Halogen H7 (Reflexion) 0,23 lx 0,24 lx 0,36 lx Xenon D1 (Projektion) 0,36 lx 0,37 lx 0,52 lx`
caption	caption	`<caption caption="caption">`	Bild- oder Tabellenunterschrift im Text z. B. `Bild 1: Architektur- und Funktionsentwurf im V-Modell`
formula	formula	`<formula formula="formula">`	`GL . 1 Formeldarstellung`
list	list	`<list id="list">`	`– Spannungskonzentration – Schweißnahtverlauf und Schweißlage – Lastspielzahl und Lastgröße – Steifigkeit sowie – Crashverhalten`
acknow-ledgement	acknow-ledgement	`<ack acknowledgement= "acknowledgement">`	`Danke Besonderer Dank für ihre Mitarbeit gilt Andreas Borg , Christer Bergström […]`

Tabelle 4: Annotationen zur Dokumentstruktur

2.3.2 Lemma-, POS- und Satzannotation

Satzspannen wurden mithilfe eines *Python*-Algorithmus aus dem *Natural Language Toolkit* (Bird et al., 2009; https://www.nltk.org/, Kapitel 3.3) erkannt und Lemmata und Wortarten als POS-Tags (Kapitel 3.4) mithilfe des Treetaggers (Schmid, 1995; https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/) annotiert. Zusätzlich wurde eine weitere Lemmaannotationsspur für eine verbesserte Version der Annotation des *TreeTaggers* angelegt (Kapitel 4).

Spurname (ANNIS)	**Beschreibung der Tags**
pos	Wortarten als POS-Tags nach STTS (Schiller et al., 1999), durch den TreeTagger annotiert
lemma_tt	Lemma bzw. Grundform, durch den TreeTagger annotiert
lemma_cor	Lemmaannotation, in der einige dem TreeTagger unbekannten Wörter nachannotiert wurden (nur für die Jahrgänge 2011 bis 2016)
s	Satzspanne, der Tag heißt immer `s`

Tabelle 5: Linguistische Annotationen

2.4 Anordnung der Annotationen

2.4.1 Token- und Spannenannotation in ANNIS und in den XML-Skripten

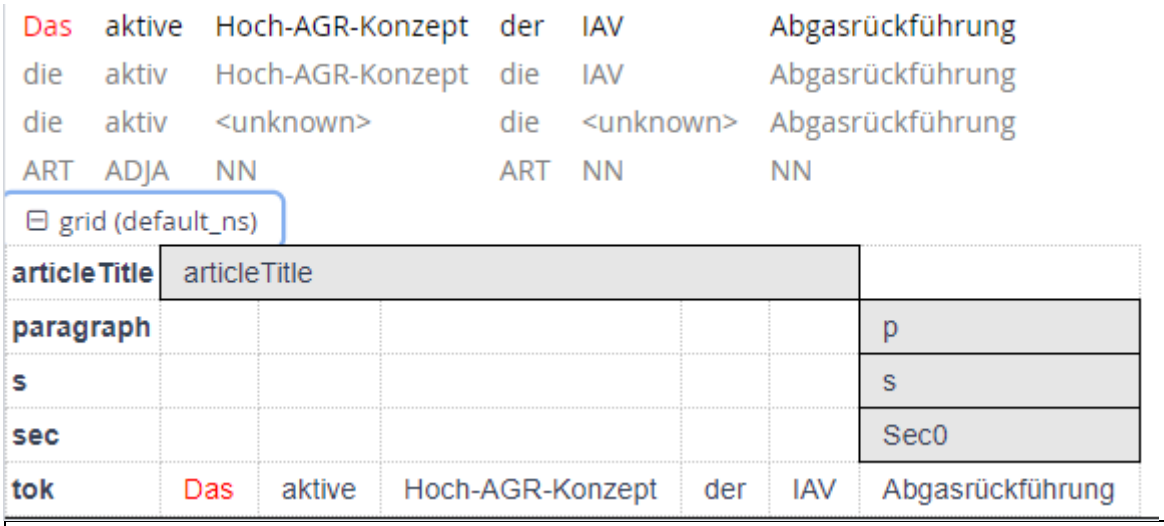
Abbildung 1 zeigt anhand eines Screenshots, wie in ANNIS die genannten Annotationen angezeigt werden. Listing 2 zeigt denselben Ausschnitt im XML-Skript.

ANNIS stellt Annotationen als Spuren mit Namen (z. B. `paragraph`) dar; in den Spuren reihen sich Tags (z. B. `p`) aneinander. Alle Annotationen der Dokumentstruktur und die Satzannotation sind Annotationen, die sich auf mehrere Tokens beziehen; sie liegen als Spanne über diesen Tokens. In den XML-Dateien sind die Spannenannotationen als XML-Elemente eingepflegt (siehe auch Kapitel 2.4.2), wobei die Attribute der XML-Elemente die Spurnamen und Tags für ANNIS beinhalten und die öffnenden bzw. schließenden XML-Tags die Grenzen der Annotation markieren[1].

[1] Für die Konvertierung in ANNIS wurde der *TreeTaggerImporter* genutzt (Kapitel 3.5).

Die POS- und Lemmaannotationen dagegen beziehen sich immer nur auf ein Token und sind in den XML-Skripten als Spalten in der Reihenfolge *tok* (Wortform im Text) – *pos* – *lemma_tt* – *lemma_cor*[2] angeordnet (Listing 2). Der Spurname der POS- bzw. Lemmaannotation wird während der Konvertierung zu ANNIS festgelegt (Kapitel 3.5) und ist nicht in den XML-Skripten zu finden.

Abbildung 1: Anordnung der Annotationen in ANNIS




```
[…]
<article-title articleTitle="articleTitle">
Das   ART   die   die
aktive      ADJA  aktiv aktiv
Hoch-AGR-Konzept  NN    &lt;unknown&gt;   Hoch-AGR-Konzept
der   ART   die   die
IAV   NN    &lt;unknown&gt;   IAV
</article-title>
<sec sec="Sec0">
<p paragraph="p">
<sentence s="s">
Abgasrückführung  NN    Abgasrückführung  Abgasrückführung
(     $(    (     (
AGR   NE    &lt;unknown&gt;   AGR
)     $(    )     )
ist   VAFIN sein  sein
eine  ART   eine  eine
bewährte    ADJA  bewährt     bewährt
Maßnahme    NN    Maßnahme    Maßnahme
zur   APPRART     zu    zu
Schadstoffreduzierung   NN    Schadstoffreduzierung
      Schadstoffreduzierung
bei   APPR  bei   bei
Dieselmotoren     NN    Dieselmotor Dieselmotor
.     $.    .     .
</sentence>
[…]
```

Listing 2: Anordnung der Annotationen in XML

[2] *lemma_cor* steht nur für die Jahrgänge 2011 bis 2016 zur Verfügung.

2.4.2 Kommentare zu einigen XML-Elementen

2.4.2.1 Element für eine Abbildung *fig*

Abbildungen werden in den Dateien der Jahrgänge 2007 bis 2010 sowohl im Element `<label>` als auch am Anfang der Bildunterschrift `<caption>` mit dem Bildlabel versehen. In den Jahrgängen 2011 bis 2016 steht das Bildlabel nur im Element `<caption>` (Listing 3). In ATZ (2011-2016) sind die Abbildungen mit einem Zahlsymbol (❻ `ADJD &lt;unknown&gt;`) nummeriert (Listing 4); in MTZ (2011-2016) sind die Abbildungen nicht nummeriert. Manchmal kommt in den älteren Jahrgängen auch die Nummerierung mit Buchstaben vor.

```
<fig fig="fig2">
<label>
Bild  NN    Bild
2     CARD  @card@
</label>
<caption caption="caption">
<p paragraph="p">
<sentence s="s">
2     CARD  @card@
Schnittbild NN    Schnittbild
des   ART   die
Hydraulik-Impuls-Speichers    NN    &lt;unknown&gt;
(     $(    (
HIS   NN    His
)     $(    )
</sentence>
</p>
</caption>
</fig>
```

Listing 3: Beispiel einer Abbildung im XML-Skript für für die Jahrgänge 2007 bis 2010

```
<fig fig="fig6">
Abbildung   NN    Abbildung   Abbildung
❻     ADJD  &lt;unknown&gt;   ❻
<caption caption="caption">
<p paragraph="p">
<sentence s="s">
Relative    ADJA  relativ     relativ
Häufigkeit  NN    Häufigkeit  Häufigkeit
von   APPR  von   von
Trennungen  NN    Trennung    Trennung
im    APPRART     in    in
Fahrzeugverband   NN    &lt;unknown&gt;   Fahrzeugverband
bei   APPR  bei   bei
Überholmanövern   NN    Überholmanöver    Überholmanöver
-     ADJA  &lt;unknown&gt;   -     &lt;unknown&gt;
Mittelwerte NN    Mittelwert  Mittelwert
und   KON   und   und
Standardabweichungen    NN    &lt;unknown&gt;   Standardabweichung
</sentence>
</p>
</caption>
```

```
<graphic href="35148_2011_96_Fig6_HTML.gif" />
</fig>
```

Listing 4: Beispiel einer Abbildung im XML-Skript für ATZ (2011-2016)

2.4.2.2 Element für eine Formel *formula*

In ATZ und MTZ (2007-2010) sind Formeln nur durch Platzhalter `Formeldarstellung GL . [ZAHL]` übernommen worden. In ATZ und MTZ (2011-2016) steht meistens zusätzlich ein Bildverweis als Leertag `<graphic href="" />` (Listing 5).

```
<formula formula="formula">
Formeldarstellung NN    &lt;unknown&gt;   &lt;unknown&gt;
GL    NN    &lt;unknown&gt;   GL    &lt;unknown&gt;
.     $.    .     .
Gl.   ADJA  &lt;unknown&gt;   Gl.
1.3   CARD  @card@      @card@
<graphic href="35148_2011_Article_92_TeX2GIF_Equ5.gif" />
</formula>
```

Listing 5: Beispiel einer Formel im XML-Skript für die Jahrgänge 2011 bis 2016

2.4.2.3 Element für eine Tabelle *table-wrap*

Bei Tabellen blieb die Tabellenstruktur aus den PDF-Dokumenten nicht erhalten. In ATZ und MTZ (2007-2010) sind die Inhalte der einzelnen Zellen durch Leerzeichen getrennt aneinandergereiht (Listing 6). In ATZ und MTZ (2011-2016) sind die Zellen in den XML-Dateien mit `<td>` getrennt. Außerdem gibt es im Unterschied zu ATZ und MTZ (2007-2010) nicht die Unterelemente `<label>` und `<table>` (Listing 7). In ATZ und MTZ (2007-2010) ist das Tabellenlabel im XML-Attribut immer klein geschrieben mit Zahl (`tab1`); in ATZ und MTZ (2011-2016) existieren die Label `Tab1, Tab2, Tab2a, Tab3, Tab4, Taba, Tabb, Tabc, Tabd` und `Tabe`.

```
<table-wrap tab="tab1">
<label>
Tabelle     NN    Tabelle
1     CARD  @card@
</label>
<caption caption="caption">
<p paragraph="p">
<sentence s="s">
Tabelle     NN    Tabelle
:     $.    :
Beispielhafte     ADJA  beispielhaft
Klasseneinteilung NN    Klasseneinteilung
des   ART   die
Energiemanagement-Ansatzes    NN    &lt;unknown&gt;
</sentence>
</p>
</caption>
<table>
Klasse      NN    Klasse
0     CARD  @card@
Enthält     VVFIN enthalten
```

```
die   ART   die
maximale    ADJA  maximal
Abgabeleistung    NN    &lt;unknown&gt;
aller PIAT  alle
Erzeuger    NN    Erzeuger
und   KON   und
Speicher    NN    Speicher
Klasse      NN    Klasse
1     CARD  @card@
[…]
</table>
</table-wrap>
```

Listing 6: Beispiel einer Tabelle im XML-Skript für die Jahrgänge 2007 bis 2010

```
<table-wrap tab="Tab1">
Tabelle     NN    Tabelle     Tabelle
<caption caption="caption">
<p paragraph="p">
<sentence s="s">
Tabelle     NN    Tabelle     Tabelle
1     CARD  1     1
Kraftstoffeigenschaften NN    &lt;unknown&gt;   &lt;unknown&gt;
(     $(    (     (
©     NN    &lt;unknown&gt;   ©     &lt;unknown&gt;
CMT   NE    CMT   CMT
)     $(    )     )
</sentence>
</p>
</caption>
ROZ94,6     CARD  &lt;unknown&gt;   &lt;unknown&gt;
<td>
MOZ   NN    &lt;unknown&gt;   MOZ
</td>
<td>
84,8  CARD  @card@      @card@
</td>
<td>
AKI   NN    &lt;unknown&gt;   AKI   AKI
=     $(    =     =
(ROZ+MOZ)/2 CARD  &lt;unknown&gt;   &lt;unknown&gt;
</td>
<td>
89,7  CARD  @card@      @card@
</td>
<td>
H/C-Verhältnis    NN    &lt;unknown&gt;   H/C-Verhältnis
</td>
[…]
</table-wrap>
```

Listing 7: Beispiel einer Tabelle im XML-Skript für die Jahrgänge 2011 bis 2016

2.4.2.4 Element für eine Auflistung *list*

Mal stehen Symbole der Auflistungspunkte (z. B. Bindestriche, Doppelpunkte, etc.) in den XML-Skripten (Listing 8), mal nicht; in einigen Transkripten ist im Element `<list-item>` ein Absatz `<p paragraph="p">` markiert (Listing 9).

```
<list list="list">
<list-item>
–     $(    -
Spannungskonzentration  NN    &lt;unknown&gt;
</list-item>
<list-item>
–     $(    -
Schweißnahtverlauf      NN    &lt;unknown&gt;
und   KON   und
Schweißlage NN    &lt;unknown&gt;
</list-item>
[…]
<list-item>
–     $(    -
Crashverhalten    NN    &lt;unknown&gt;
</list-item>
</list>
```

Listing 8: Beispiel einer Auflistung im XML-Skript

```
<list list="list">
[…]
<list-item>
<p paragraph="p">
<sentence s="s">
Erfüllung   NN    Erfüllung   Erfüllung
der   ART   die   die
neuesten    ADJA  neu   neu
Gesetzesanforderungen   NN    &lt;unknown&gt;   &lt;unknown&gt;
und   KON   und   und
der   ART   die   die
sozialen    ADJA  sozial      sozial
Akzeptanz   NN    Akzeptanz   Akzeptanz
.     $.    .     .
</sentence>
</p>
</list-item>
</list>
```

Listing 9: Beispiel einer Auflistung im XML-Skript

2.4.2.5 Element für Danksagungen *ack*

Der Aufbau des Elements für eine Danksagung wird in den vier Subkorpora einheitlich verwendet.

```
<ack acknowledgement="acknowledgement">
<title subheading="subheading">
Danke PTKANT      danke danke
</title>
<p paragraph="p">
```

```
<sentence s="s">
Unser PPOSAT      unser unser
Dank  NN    Dank  Dank
gilt  VVFIN gelten      gelten
der   ART   die   die
Arbeitsgemeinschaft     NN    Arbeitsgemeinschaft     Arbeitsgemeinschaft
industrieller     ADJA  industriell industriell
Forschungsvereinigungen NN    Forschungsvereinigung
      Forschungsvereinigung
(     $(    (     (
AIF   NE    &lt;unknown&gt;   AIF
)     $(    )     )
für   APPR  für   für
[…]
</sentence>
[…]
</p>
</ack>
```

Listing 10: Beispiel einer Danksagung im XML-Skript

2.4.2.6 Elemente für Literaturhinweise

Literaturhinweise sind in den Jahrgängen 2007-2010 nicht in die XML-Datei übernommen worden. In den Jahrgängen 2011-2016 stehen sie als XML-Leertags `<Literaturhinweis […] />`, z. B.:

```
<Literaturhinweis mixed-citation="Frambourg, M.; Rohrssen, K.: Ein neues
Schraubenverdichterkonzept zur Realisierung sehr hoher Abgasrückführraten
beim Dieselmotor. VDI-Tagung Schraubenmaschinen Dortmund 10/2010"
publication-type="other" />
```

3 Datenaufbereitung

Dieses Kapitel beschreibt die Aufbereitung der Rohdaten. Ziel der Aufbereitung war es, die Texte im PDF-Format maschinenlesbar zu machen, nicht benötigte Daten aus den XML-Dateien zu löschen, Metadaten in eine Form zu bringen, die von ANNIS interpretiert werden kann, sowie Satz-, Lemma- und POS-Annotation hinzuzufügen. Nach der Aufbereitung sollten alle Zeitschriftenartikel in einer einheitlichen XML-ähnlichen Struktur vorliegen.

3.1 Konvertierung von PDF zu XML

Um die Artikel der Jahrgänge 2007 bis 2010 im PDF-Format maschinenlesbar zu machen, wurden sie manuell mittels einer Vorlage und Schnellbausteinen im Programm *Microsoft Word*[3] ins XML-Format überführt. Die Vorlage wird in Listing 11 gezeigt. Die Schnellbausteine beinhalteten die XML-Elemente, die die Textbestandteile der Zeitschriftenartikel (Zwischenüberschriften, Textblöcke, Abbildungen, Formeln, Auflistungen,

[3] In der Menüleiste von *Word* sind die Schnellbausteine unter *Einfügen>Schnellbausteine>Organizer für Bausteine* zu finden. Es öffnet sich ein Fenster, in dem vordefinierte Baustein durch Anklicken eingefügt werden können.

Tabellen und Danksagungen) markieren sollten (Kapitel 2.3.1 und 2.4). Bei Bedarf wurden die XML-Elemente in die Vorlage eingefügt und anschließend wurden die Textbestandteile aus den PDFs in die Struktur der Vorlage kopiert. Die Form der Vorlage und der Bausteine orientierte sich daran, wie die XML-Dateien der Jahrgänge 2011 bis 2016 nach ihrer Aufbereitung vorlagen (Kapitel 3.2)

```
<?xml version="1.0" encoding="UTF-8"?>

<article journal-title="ATZ - Automobiltechnische Zeitschrift" article-
title="" subtitle="" author1="Nachname, Vorname" author2="" author3=""
pub-date="" volume="" issue="" firstpage="" lastpage="">

<article-title articleTitle="articleTitle"></article-title>
<subtitle subtitle="subtitle"></subtitle>

<sec sec="sec0">
<p paragraph="p">
</p>
</sec>

<sec sec="sec1">
<title subheading="subheading"></title>
<p paragraph="p">
</p>
<p>
</p>
</sec>

<sec sec="sec2">
<title subheading="subheading"></title>
<p paragraph="p">
</p>
<p paragraph="p">
</p>
</sec>

</article>
```

Listing 11: Grundstruktur eines Artikels als XML-Dokument

3.2 Umformatierung der XML-Dateien aus JATS

Die vom Verlag bereitgestellten XML-Dateien im JATS-Schema der Jahrgänge 2011 bis 2016 mussten bereinigt und umformatiert werden, da die Originaldateien eine Vielzahl von nicht benötigten Metadaten beinhalteten. Außerdem wurden die für das Korpus relevanten Metadaten in eine Form gebracht, die von ANNIS interpretiert werden können. Des Weiteren wurden in den XML-Dateien die Anzahl der Hierarchieebenen reduziert. Dies erhöhte die Übersichtlichkeit der Daten und erleichterte die weitere Verarbeitung.

Um die beschriebenen Veränderungen vorzunehmen, wurde ein XSLT-Skript *(eXtensible Stylesheet Language Transformation)* geschrieben (Anhang B), welches auf alle bereitgestellten XML-Dateien angewandt wurde. XSLT ist eine Programmiersprache, die explizit zur Transformation von XML-Dateien entwickelt wurde. Für die Konzeption sowie für

das Testen des XSLT-Skriptes wurde die Software *NetBeans* (https://netbeans.org) verwendet. Zur Anwendung des Skriptes auf die XML-Dateien wurde die Kommandozeile *Terminal* benutzt.

Im *Terminal* wurde der Befehl aus Listing 12 eingegeben. Dieser Befehl besteht dabei aus drei Bereichen. Zunächst wird mit dem Schlüsselwort `for` eine Schleife geöffnet. Anschließend wird der Dateipfad zu den XML-Dateien angegeben und der Pfad der Variable `f` übergeben. Danach wird nach `do` der Operator `xsltproc` sowie der Pfad zum XSLT-Skript eingefügt. Abschließend wird dem Dateinamen der XML-Datei `_Transformiert` hinzugefügt. Diese for-Schleife wird so lange ausgeführt, bis sie über alle Dateien gelaufen ist, die im angegeben Dateipfad für die XML-Dateien zu finden sind.

```
for f in /[PATH_FROM_THE_XML_DOKUMENTS]/*.xml; do xsltproc
/[PATH_FROM_THE_XSLT_SKRIPT]/newstylesheet_2.xsl $f >
${f/.xml/_Transformiert.xml}; done
```

Listing 12: Befehl zur Ausführung des XSLT-Skripts

3.3 Satzannotation

Da beim späteren Taggen der Daten mit Hilfe des *TreeTaggers* keine automatische Satzgrenzenannotation vorgenommen wurde, erfolgte dies gesondert mit einem *Python*-Skript (https://www.python.org), das Algorithmen aus dem *Natural Language Toolkit* (https://www.nltk.org/) einband. Das Skript las die XML-Dateien aller Zeitschriftenartikel (Jahrgänge 2007 bis 2016) ein und fügte die Satzgrenzenannotation hinzu.

Im *Python*-Skript muss der korrekte Pfad für die XML-Dateien gesetzt werden. Diese Anpassung geschieht in Zeile 14 des Skriptes (Listing 13). Anschließend kann das Skript ausgeführt werden. Die bearbeiteten Dateien werden unter demselben Pfad mit verändertem Dateinamen gespeichert. Hier wurde dem Dateinamen wird ein `_sent` hinzugefügt.

```
from nltk import sent_tokenize, word_tokenize, pos_tag
import nltk.data
import xml.etree.ElementTree as ET
import glob
import os

#Variable zum Programmcheck
nummer=0

#Laden von Tokenizer Daten
tokenizer = nltk.data.load('tokenizers/punkt/german.pickle')

#Setzen von Pfad für die zu bearbeitenden Dateien
path = '/Users/XXXXX/XXXXX/'

#Laden von allen Dateien nacheinander
for filename in glob.glob(os.path.join(path, '*.xml')):
    print(filename)

    #Da Dateien im XML-Format eingelesen werden, wird hier mit dem
ElementTree gearbeitet. Dieser garantiert, dass Daten richtig eingelesen
werden und bearbeitet werden.
    tree = ET.parse(filename)
    root = tree.getroot()
```

```
    #Es wird in alle <p>-Elemente der XML-Datei gegangen, dabei der Text
innerhalb des Elements an den Tokenizer übergeben.
    for p in root.iter('p'):
        text = ""
        try:
            text = (p.text)
            sents = tokenizer.tokenize(text)
            #Der Tokenizer eröffnet für jeden getaggten Satz ein neues
SubElement in der XML-Struktur und fügt diesen das Attribut "Sentence"
hinzu.
            for x in sents:
                sentence = ET.SubElement(p, 'sentence')
                sentence.text = x
                sentence.set('id', 'sentence')
            p.text = ""
        except(TypeError):
            print('Catch Error Text: ' +filename)

        #Da auch Texte innerhalb von Kindelementen der XML-Struktur
vorhanden sein können, müssen auch hier alle Textelemente gesucht und
getaggt werden.
        for children in p.iter('fig'):
            try:
                text = (children.tail)
                sents = tokenizer.tokenize(text)
                for x in sents:
                    sentence = ET.SubElement(p, 'sentence')
                    sentence.text = x
                    sentence.set('id', 'sentence')
                children.tail = ""

                text = children.tail
            except(TypeError):
                print('Catch Error Fig: ' + filename)



        for children in p.iter('table'):
            try:
                text = (children.tail)
                sents = tokenizer.tokenize(text)
                for x in sents:
                    sentence = ET.SubElement(p, 'sentence')
                    sentence.text = x
                    sentence.set('id', 'sentence')
                children.tail = ""

                text = children.tail
            except(TypeError):
                print('Catch Error Tab: ' + filename)

        for children in p.iter('formular'):
            try:
                text = (children.tail)
                sents = tokenizer.tokenize(text)
                for x in sents:
                    sentence = ET.SubElement(p, 'sentence')
                    sentence.text = x
                    sentence.set('id', 'sentence')
                children.tail = ""

                text = children.tail
```

```
            except(TypeError):
                print('Catch Error Form: ' + filename)

        for children in p.iter('list'):
            try:
                text = (children.tail)
                sents = tokenizer.tokenize(text)
                for x in sents:
                    sentence = ET.SubElement(p, 'sentence')
                    sentence.text = x
                    sentence.set('id', 'sentence')
                children.tail = ""

                text = children.tail
            except(TypeError):
                print('Catch Error list: ' + filename)

    #Schreibt alle Änderungen wieder in eine XML-Datei und benennt diese
mit dem Orginal Titel und fügt _sent hinzu.
    nummer=nummer+1
    filename_new=filename[0:-4]+"_sent.xml"
    tree.write(filename_new, encoding="UTF-8", xml_declaration=True)

    print(nummer)
```

Listing 13: Sentencer-Modul in Python

Listing 14 zeigt einen Ausschnitt einer XML-Datei nach diesem Schritt.

```
<sec id="sec1">
<title id="subheading">1 Einleitung</title>
<p><sentence id="sentence">Die Zunahme der installierten
Verbraucherleistung – insbesondere seit den 90er-Jahren des vergangenen
Jahrhunderts – führte zu einem deutlichen Anstieg der Batterieausfälle in
dem darauf folgenden Jahren [1].</sentence>
```

Listing 14: Auszug aus eine satzannotierten XML-Datei[4]

3.4 Lemma- und POS-Annotation

Die Lemma- und POS-Annotation wurde für die XML-Dateien aller Zeitschriftenartikel (Jahrgänge 2007 bis 2016) vollautomatisch durch die Software *TreeTagger* (Schmid, 1995; https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/) übernommen. Um den *TreeTagger* auszuführen, ist es notwendig, innerhalb des *Terminals* in das Verzeichnis des *TreeTaggers* zu navigieren[5]. Innerhalb des Ordners des *TreeTaggers* kann dann der *Terminal*-Befehl aus Listing 15 ausgeführt werden.

Angepasst werden muss hier das Verzeichnis der XML-Dateien, die getaggt werden sollen. Ähnlich zum *Python*-Skript (Kapitel 3.2, Anhang B) arbeitet dieser Befehl als Schleife, welcher alle XML-Dateien innerhalb des angegebenen Ordners bearbeitet. Innerhalb der for-Schleife

[4] Die Bezeichnungen der Attribute der XML-Elemente wurden nachträglich verändert, damit die unterschiedlichen Annotationen in ANNIS jeweils als eigene Spur angezeigt wurden: Aus `id` wurde hier `sec`, `subheading` bzw. `s`.

[5] In einem UNIX-Betriebssystem (*Mac* oder *Linux*) können beispielsweise die Kommandos ls (Anzeige von allen Dateien im aktuellen Ordner) sowie cd (Wechsel in einen Ordner) benutzt werden. Für weitere Informationen wie innerhalb des *Terminals* zu Ordnern navigiert werden kann:
UNIX: https://github.com/0nn0/terminal-mac-cheatsheet
Windows: https://commandwindows.com/command3.htm

dient der Befehl `cmd/tree-tagger-german` als Operator um den *TreeTagger* zu aktivieren und die übergebenen XML-Dokumente zu taggen. Die bearbeiteten Dateien werden unter demselben Pfad mit verändertem Datennamen gespeichert. Hier wurde dem Dateinamen `_Tagged` hinzugefügt. Weiterhin verändert sich in diesem Schritt das Dateiformat zu einem Tab-getrennten Datenformat, sodass *tok*-, Lemma-, und POS-Spur in Spalten angeordnet sind, wie es Listing 2 auf Seite 9 zeigt. Durch die Spalten sind die Dateien keine XML-Skripte im eigentlichen Sinne mehr.

```
for f in /[PATH_FROM_THE_XML_DOCUMENTS]/*_sent.xml; do cmd/tree-ta
gger-german $f > ${f/.xml/_Tagged.tab}; done
```

Listing 15: Befehl zur Ausführung des Treetaggers

3.5 Durch XML reservierte Zeichen

Tabelle 6 zeigt Zeichen, die in XML bereits eine spezifische Bedeutung haben. Beispielsweise markiert das Größerzeichen den Beginn eines XML-Tags. Damit die Zeichen trotzdem im Text dargestellt werden können, wurden sie in den Jahrgängen 2007 bis 2010 durch Entitäten ersetzt, in den Jahrgängen 2011 bis 2016 lagen die Zeichen schon als Entitäten vor. Da der TreeTagger nicht für Dateien im XML-Format geschrieben wurde und die Tokenisierung die Semikolons von den Entitäten abtrennte, wurde dies nachträglich wieder korrigiert. Auch das POS-Tag `<unknown>` wurde in `&lt;unknown&gt;` umgewandelt.

Reserviertes Zeichen	**Benannte Entität**
<	&lt;
>	&gt;
&	&amp;
"	&quot;
'	&apos;

Tabelle 6: Durch XML reservierte Zeichen

3.6 Erstellung von ANNIS-Datenbanken

In diesem Kapitel wird die Bereitstellung der aufbereiteten Dateien in ANNIS (Krause & Zeldes, 2016) beschrieben, das sowohl Format als auch Suchtool ist. Als erster Schritt wurden die XML-Entitäten (Kapitel 3.5) als reguläre Zeichen gesetzt und XML-Leertags wie `<subtitle />` oder die Leertags der Literaturangaben (Kapitel 2.4.2.6) gelöscht. Dann wurde mit der Konvertierungssoftware *Pepper* (Zipser & Romary, 2010) ANNIS-Datenbanken erstellt. Da die Artikel der Jahrgänge 2007 bis 2010 nicht dieselbe Anzahl an Annotationsebenen haben, konnten nicht dieselben *Pepper*-Workflows genutzt werden und es wurden vier Subkorpora gebildet: ATZ_1, ATZ_2, MTZ_1 und MTZ_2 für die Zeitschriften ATZ und MTZ jeweils für die Jahrgänge 2007 bis 2010 und für die Jahrgänge 2011 bis 2016.

Um die Software auszuführen, ist es notwendig, innerhalb des *Terminals* in das Verzeichnis von *Pepper* zu navigieren[6]. Dann kann die Software gestartet (Listing 16) und ein automatisierter Workflow angestoßen werden (Listing 17).

```
pepperStart.bat   //(Windows)
bash pepperStart.sh     //(UNIX-Systeme)
```

Listing 16: Befehl zum Starten von Pepper

Anschließend kann mit dem folgenden Befehl ein automatisierter Workflow (Kapitel 3.6.1) gestartet werden:

```
convert [PATH_TO_WORKFLOW_FILE/myConversion].pepper
```

Listing 17: Befehl zum Ausführen des Pepper-Workflows

Die fertig erstellten Datenbanken können am Ende auf die lokale Kickstarter- oder die Server-Version von ANNIS hochgeladen werden (Zipser & Romary, o. J.).

3.6.1 Workflow-Datei

Die Workflow-Datei ist in Listing 18 abgebildet. Jeweils nach der Auswahl des *Importers* und *Exporters* (`importer name, exporter name`) muss nach `path=` der Pfad zu den Dateien, die verwendet werden sollen, bzw. der Pfad zum Export des fertigen Korpus angepasst werden. Weitere Einstellungen werden zwischen den Klammern `customization` angegeben. Die erste Anpassung (`property key`) macht *Pepper* die Meta-Dateien innerhalb der einzelnen Artikel bekannt. Die zweite Anpassung bindet eine Meta-Datei für das Gesamtkorpus mit in den Prozess ein. Die dritte Anpassung (`<property key="columnNames">`) beschreibt die Anordnung der Token-Annotationen: `tok, pos, lemma_tt, lemma_cor`[7] zeigt die Reihenfolge der tab-getrennten Spalten in den Dateien an. Weitere Informationen zu den Modulen in Pepper können unter folgenden Link abgerufen werden: http://corpus-tools.org/pepper/knownModules.html

```
<?xml version='1.0' encoding='UTF-8'?>

<pepper-job id="deem92em" version="1.0">

<importer name="TreetaggerImporter" path="/[PATH_TO_CORPUS-FOLDER]/">
      <customization>
      <property key="treetagger.input.metaTag">article</property>
      <property key="pepper.before.readMeta">meta</property>
      <property key="columnNames">tok, pos, lemma_tt,
lemma_cor</property>
      </customization>
</importer>

<exporter name="ANNISExporter" path="/[PATH_TO_SAVE_CORPUS_IN_THE_END]/">
      <customization>
      <property key="corpusName">Gingko_ATZ_2</property>
```

[6] In einem UNIX-Betriebssystem (*Mac* oder *Linux*) können beispielsweise die Kommandos ls (Anzeige von allen Dateien im aktuellen Ordner) sowie cd (Wechsel in einen Ordner) benutzt werden. Für weitere Informationen wie innerhalb des *Terminals* zu Ordnern navigiert werden kann:
UNIX: https://github.com/0nn0/terminal-mac-cheatsheet
Windows: https://commandwindows.com/command3.htm

[7] *lemma_cor* steht nur für die Jahrgänge 2011 bis 2016 zur Verfügung.

```
            </customization>
</exporter>

</pepper-job>
```
Listing 18: Worflow-Datei

3.6.2 Metadaten-Datei

Pepper nutzt eine separate Meta-Datei, um korpusübergreifende Metadaten abzulegen. Hierbei werden beliebige Variablen mit einem Tab getrennt den gewünschten Werten zugeordnet. Eine derartige Datei kann mit einem Texteditor, wie NotePad++ (https://notepad-plus-plus.org/), erstellt und im txt-Format abgespeichert werden. Abschließend muss die Dateiendung (`.txt`) manuell zu `.meta` verändert werden.

```
01_project_name   Muster in der Sprache der Ingenieurwissenschaften
02_project_place  Greifswald (2017-2019), Leipzig (2020-2021)
03_project_institution  Universit\u00E4t Greifwald, Institut für Deutsche
Philologie (2017-2019); Universit\u00E4t Leipzig, Herder-Institut (2020-
2021)
04_project_type   Drittmittelprojekt; gef\u00f6rdert durch die DFG; AOBJ:
632723
05_project_head   Jun.-Prof. Dr. Antje Heine (2017-2019), Prof. Dr.
Christian Fandrych (2020-2021)
06_project_duration     2017-2021
07_project_description  Gingko (Geschriebenes ingenieurwissenschaftliches
Korpus) besteht aus 2498 wissenschaftlichen Artikeln der Zeitschriften
Automobiltechnische Zeitschrift (ATZ) und Motortechnische Zeitschrift
(MTZ) der Jahrg\u00E4nge 2007-2016 und umfasst insgesamt 4.667.656
Tokens. Es ist im Rahmen des Forschungsprojektes „Muster in der Sprache
der Ingenieurwissenschaften“ entstanden. Das Projekt hat das Ziel, Muster
in der Sprache der Ingenieurwissenschaften (am Beispiel der
Automobiltechnik) systematisch zu erfassen und zu beschreiben.
08_corpus_title   Geschriebenes ingenieurwissenschaftliches Korpus
09_corpus_acronym Gingko
10_corpus_publication   Schirrmeister, L., Rummel, M., Heine, A., Suppus,
N. & Mendoza S\u00E1nchez, B. (2021). Das Fachsprachenkorpus Gingko.
Deusch als Fremdsprache(3).
11_corpus_head    Prof. Dr. Christian Fandrych
12_corpus_contributors  2017-2019: Antje Heine, Marlene Rummel, Lars
Schirrmeister, Nina Suppus, Sarah Brauer, Rebekka Fricke, Anne Hertel,
Marcel Knorn, B\u00E1rbara Mendoza Sanchez, Paul Kn\u00f6tgen, Agnes
Koschmider; 2020-2021: Christian Fandrych, Cordula Mei\u00dfner, Annette
Portmann, Lars Schirrmeister, Franziska Wallner
13_corpus_version 2.0
14_corpus_language      Deutsch
15_corpus_sizeTexts     2498
16_corpus_sizeToken     4667656
17_corpus_mode    geschrieben
18_corpus_textType      wissenschaftliche Artikel, 426 journalistische
Texte
19_corpus_field   Wissenschafts- und Fachsprache
20_corpus_annotation    Lemma, POS, Satzspannen, Textsorte,
Dokumentstruktur
21_corpus_posTagset     stts
```

```
22_source_journalTitle  ATZ - Automobiltechnische Zeitschrift und MTZ -
Motortechnische Zeitschrift
23_source_years   2007-2016
24_source_publisher     Springer Fachmedien Wiesbaden
25_source_issuesPerYear 11
```

Listing 19: Meta-Datei für ATZ_2

4 Verbesserung der Lemmaannotation

In diesem Kapitel wird erläutert, wie die Lemmaannotation des *TreeTaggers* für die Texte der Jahrgänge 2011 bis 2016 verbessert und in der separaten Annotationsspur `lemma_cor` abgespeichert wurde. Der *TreeTagger* nutzt zur Lemmatisierung eine Wortliste. Ist ein Wort des Korpus nicht in der Wortliste des *TreeTaggers* enthalten, setzt er das Tag `<unknown>`. Damit auch diesen Wörtern ein Lemma zugewiesen wird, veränderten *Python*-Skripte von Martin Klotz das Tagging des *TreeTaggers*, wobei sie den Duden zur Grundlage nahmen.

4.1 Erstellung von Lemmalisten

Bevor die *Python*-Skripte von Martin Klotz ausgeführt werden konnten, wurden zwei Wortlisten aus ANNIS exportiert: eine Liste mit den dem *TreeTagger* unbekannten Lemmata und eine Liste mit den ihm bekannten Lemmata. Hierfür wurden die Suchanfragen `lemma!="<unknown>"` sowie `lemma="<unknown>"` genutzt und anschließend exportiert. Verwendet wurde dabei der *TokenExporter* mit den Einstellungen `Left Context=0` und `Right Context=0` (Abbildung 2). Im zweiten Schritt wurden nicht benötigte Daten aus dem ANNIS-Export entfernt. Dazu wurde ein *Python*-Skript verwenden, das im Listing 21 in Anhang **Fehler! Verweisquelle konnte nicht gefunden werden.** zu finden ist.

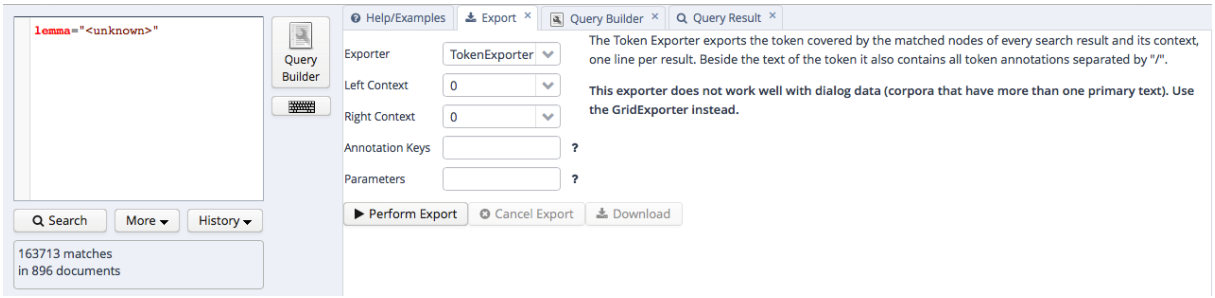


Abbildung 2: Exporteinstellungen in ANNIS für die Lemmalisten

4.2 Austausch der *unknown*-Tags

Mithilfe dreier weiterer Python-Skripte wurde versucht demm TreeTagger unbekannte Wortformen Lemmata zuzuordnen. Der hier verwendete Code wurde von Martin Klotz entwickelt.

Das erste Skript (Listing 22, Anhang **Fehler! Verweisquelle konnte nicht gefunden werden.**) versuchte, den Wortformen mit *unknown*-Tag Lemmata zuzuordnen, indem es die Wortformen mit den Lemmata aus der mit ANNIS erstellten Lemmaliste verglich. Dabei sollte das Wort mit demselben Buchstaben wie das Lemma beginnen und Wortform und Lemma sollten eine

maximale Levenstein-Distanz von 3 aufweisen[8]. Danach glich das Skript die Wörter mit einer Lemmaliste des Dudens ab. Output des Skriptes waren eine Textdatei mit zu Lemmata zugeordneten Wortformen und eine Textdatei mit noch nicht zugeordneten Wortformen.

Das zweite Skript (Listing 23, Anhang **Fehler! Verweisquelle konnte nicht gefunden werden.**) bearbeitete die Wortformen, die vom ersten Skript nicht zugeordnet werden konnten. Es verringerte die Zuordnungsgenauigkeit um 0,25, damit weitere Wörter zugeordnet werden können. Beim Festlegen der Zuordnungsgenauigkeit wurde darauf geachtet, sie nur so weit abzusenken, dass die Zuordnung mehrheitlich korrekt funktionierte. Dies wurde stichprobenartig überprüft. Der Skriptdurchlauf produzierte eine weitere Outputdatei mit neu lemmatisierten Wörtern.

Mithilfe eines dritten Skriptes (Listing 24, Anhang 0**Fehler! Verweisquelle konnte nicht gefunden werden.**) wurden zuerst die beiden Dateien mit den neu lemmatisierten Wörtern eingelesen und anschließend die vom *TreeTagger* getaggten Dateien. Dann tauschte das Skript die veränderten Lemma-Tags aus und speicherte die Lemmaannotation unter dem neuen Spurnamen `lemma_cor` ab. Das unveränderte, automatische Tagging des TreeTaggers sollte erhalten bleiben, um Informationsverlust zu vermeiden.

[8] "The edit distance is the number of characters that need to be substituted, inserted, or deleted, to transform s1 into s2. For example, transforming 'rain' to 'shine' requires three steps, consisting of two substitutions and one insertion: 'rain' -> 'sain' -> 'shin' -> 'shine'. These operations could have been done in other orders, but at least three steps are needed." (Quelltext für *nltk.metrics.distance* in Bird et al. (2009))

5 Literaturverzeichnis

Bird, S., Loper, E. & Klein, E. (2009). *Natural Language Processing with Python*. O'Reilly Media Inc.

Krause, T. & Zeldes, A. (2016). ANNIS3: A new architecture for generic corpus query and visualization. *Digital Scholarship in the Humanities*, *31*(1), 118–139. https://doi.org/10.1093/llc/fqu057

Schiller, A., Teufel, S., Stöckert, C. & Thielen, C. (1999). *Guidelines für das Tagging deutscher Textcorpora mit STTS (Kleines und großes Tagset).* Universität Stuttgart, Institut für Maschinelle Sprachverarbeitung. http://www.sfs.uni-tuebingen.de/resources/stts-1999.pdf

Schmid, H. (1995). Improvements in Part-of-Speech Tagging with an Application to German. In *Proceedings of the ACL SIGDAT-Workshop.*

Zipser, F. & Romary, L. (o. J.). *ANNIS User Guide*. http://korpling.github.io/ANNIS/3.6/user-guide/

Zipser, F. & Romary, L. (2010). A model oriented approach to the mapping of annotation formats using standards. In *Proceedings of the Workshop on Language Resource and Language Technology Standards (LREC)*, Malta. http://hal.archives-ouvertes.fr/inria-00527799/en/

Anhang

A Liste der benutzten Tools

Alle verwendeten Tools stehen für die Betriebssysteme Mac und Windows zur Verfügung.

Microsoft Word

Microsoft Word ist eine sehr verbreitete Textbearbeitungssoftware des Unternehmens *Microsoft*. Die Software *Word* gehört mit in das Softwarepaket *Microsoft Office*. Das erste Mal wurde die Software im Jahr 1983 veröffentlicht und wurde seitdem stetig weiterentwickelt. Inzwischen ist Microsoft Word sowohl für Windows als auch für Mac Computersysteme erhältlich.

Weitere Informationen zu Microsoft Word:

Support-Center: https://support.office.com/de-de/word

Program-Download: https://products.office.com/en-us/word

NetBeans (XSLT)

NetBeans IDE ist eine freie, integrierte Entwicklungsumgebung, die komplett in der Programmiersprache Java geschrieben wurde und auf der NetBeans-Plattform läuft. *NetBeans IDE* wurde hauptsächlich für die Programmiersprache Java entwickelt, unterstützt jedoch auch C, C++ und dynamische Programmiersprachen.

In diesem Projekt wurde die Software Netbeans in der Version 8.2 benutzt.

Homepage: https://netbeans.org

Python 3.6

Python ist eine dynamische Skript-Programmiersprache und wurde im Beginn der 1990er Jahre entwickelt. Die Software ist heutzutage als Open-Source-Software verfügbar. Das Paket *Natural Language Toolkit* der Stanford University umfasst State-of-the-Art Implementierungen für so gut wie alle wichtigen Algorithmen aus der Sprachverarbeitung und ist in den meisten Python-Distributionen enthalten. In diesem Projekt wurde die Software Python in der Version 3.6.X benutzt.

Homepage: https://www.python.org

Dokumentation Python: https://docs.python.org/3/

Eingebundene Pakete in Python:

Natural Language Toolkit: https://www.nltk.org/index.html

Duden: https://github.com/radomirbosak/duden

TreeTagger

Der *TreeTagger* ist eine Software, die Text mit Part-of-Speech- und Lemma-Informationen annotieren kann. Entwickelt wurde sie von Helmut Schmid von der Universität Stuttgart. Der *TreeTagger* wurde bereits erfolgreich eingesetzt um Deutsch, Englisch, Französisch, Italienisch, Dänisch und viele weitere Sprachen zu taggen.

Homepage: http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/

Pepper (Transformationssoftware)

Pepper ist eine Software zum Konvertieren oder Manipulieren von linguistischen Daten innerhalb von Dateien, Sets oder Korpora.

Homepage: http://corpus-tools.org/pepper/

User-Guide: http://corpus-tools.org/pepper/userGuide.html

ANNIS (Such- und Visualisierungsarchitektur)

ANNIS ist eine quellcodeoffene, plattformübergreifende (Linux, Mac, Windows), webbrowserbasierte Such- und Visualisierungsarchitektur für komplexe mehrschichtige linguistische Korpora mit verschiedenen Arten von Annotationen. ANNIS, welches für *ANNotation of Information Structure* steht, sollte ursprünglich den Zugriff auf die Daten des Sonderforschungsbereich 632 - "Informationsstruktur: Die sprachlichen Mittel zur Strukturierung von Äußerungen, Sätzen und Texten" ermöglichen. Seitdem wurde ANNIS jedoch stetig weiterentwickelt und bietet heute eine Vielzahl von Möglichkeiten.

Homepage: http://corpus-tools.org/annis/

Offizielle Dokumentation über ANNIS: http://corpus-tools.org/annis/documentation.html

Weiterführende Dokumentation über ANNIS (vor allem für Server/Datenbank Updates hilfreich): http://grepcode.com/snapshot/repo1.maven.org/maven2/de.hu-berlin.german.korpling.annis/annis-service/3.3.3/

Terminal (Command-Line Software // Betriebssystem intern)

Die Software *Terminal* ist eine betriebssysteminterne *Command-Line-Software*. Sie existiert auf UNIX-PCs sowie auf Windows-Computern. Die Software bietet die Möglichkeit, Bearbeitungsbefehle ausführen zu lassen.

Terminal-Kommandos (UNIX-System): https://github.com/0nn0/terminal-mac-cheatsheet

Terminal-Kommandos (Windows): https://commandwindows.com/command3.htm

Notepad++

Notepad++ ist ein freier Texteditor für Windows

Homepage: https://notepad-plus-plus.org/

B XSLT-Skript zur Umformatierung der XML-Dateien aus JATS

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xlink="http://www.w3.org/1999/xlink" version="1.0">
    <xsl:output method="xml" indent="yes" encoding="UTF-8" omit-xml-
declaration="no"/>

    <xsl:template match="/">

        <xsl:element name="article">
            <xsl:attribute name="journal-title">
                <xsl:value-of select="//journal-meta/journal-title-
group/journal-title" />
            </xsl:attribute>
            <xsl:attribute name="publisher-id">
                <xsl:value-of select="//article-id" />
            </xsl:attribute>
            <xsl:attribute name="article-title">
                <xsl:value-of select="//article-title" />
            </xsl:attribute>
            <xsl:choose>
                <xsl:when test="count(//subtitle) > '0'">
                    <xsl:attribute name="subtitle">
                        <xsl:value-of select="//subtitle" />
                    </xsl:attribute>
                </xsl:when>
            </xsl:choose>
            <xsl:for-each select="//contrib-group/contrib">
                <xsl:attribute name="{concat(@contrib-type,xref)}">
                    <xsl:value-of select="concat(name/surname,',
',name/given-names)" />
                </xsl:attribute>
            </xsl:for-each>
            <xsl:attribute name="pub-date">
                <xsl:value-of select="concat(//pub-
date[2]/month,'.',//pub-date[2]/year)" />
            </xsl:attribute>
            <xsl:attribute name="volume">
                <xsl:value-of select="//volume" />
            </xsl:attribute>
            <xsl:attribute name="issue">
                <xsl:value-of select="//issue" />
            </xsl:attribute>
            <xsl:attribute name="firstpage">
                <xsl:value-of select="//fpage" />
            </xsl:attribute>
            <xsl:attribute name="lastpage">
                <xsl:value-of select="//lpage" />
            </xsl:attribute>
```

```
        <xsl:element name="article-title">
            <xsl:attribute name="id">
                <xsl:value-of select="'article-title'"/>
            </xsl:attribute>
            <xsl:value-of select="//article-title" />
        </xsl:element>

        <xsl:choose>
            <xsl:when test="count(//subtitle) > '0'">
                <xsl:element name="subtitle">
                    <xsl:attribute name="id">
                        <xsl:value-of select="'subtitle'"/>
                    </xsl:attribute>
                    <xsl:value-of select="//subtitle" />
                </xsl:element>
            </xsl:when>
        </xsl:choose>
        <xsl:apply-templates/>
    </xsl:element>
</xsl:template>



<xsl:template match="front">
</xsl:template>



<xsl:template match="body">
    <xsl:choose>
        <xsl:when test="./p">
            <xsl:element name="sec">
                <xsl:attribute name="id">
                    <xsl:value-of select="'Sec0'"/>
                </xsl:attribute>
               <xsl:for-each select="./p">
                            <xsl:element name="p">
                <xsl:apply-templates/>
                </xsl:element>
               </xsl:for-each>
                </xsl:element>
                <xsl:apply-templates/>
        </xsl:when>
        <xsl:otherwise>
            <xsl:apply-templates/>
        </xsl:otherwise>
    </xsl:choose>

</xsl:template>

<xsl:template match="sec">
```

```
        <xsl:element name="sec">
            <xsl:attribute name="id">
                <xsl:value-of select="@id"/>
            </xsl:attribute>
            <xsl:apply-templates/>
        </xsl:element>
    </xsl:template>

    <xsl:template match="title">
        <xsl:element name="title">
            <xsl:apply-templates/>
        </xsl:element>
    </xsl:template>

    <xsl:template match="p">
        <xsl:choose>
            <xsl:when test="parent::body">
            </xsl:when>
            <xsl:otherwise>
                <xsl:element name="p">
                    <xsl:apply-templates/>
                </xsl:element>
            </xsl:otherwise>
        </xsl:choose>

    </xsl:template>



    <xsl:template match="list">
        <xsl:element name="list">
            <xsl:attribute name="id">
                <xsl:value-of select="'list'"/>
            </xsl:attribute>
            <xsl:apply-templates/>
        </xsl:element>
    </xsl:template>

    <xsl:template match="list-item">
        <xsl:element name="list-item">
            <xsl:apply-templates/>
        </xsl:element>
    </xsl:template>



    <xsl:template match="fig">

            <xsl:choose>
                <xsl:when test="contains(./@id,'Eq')">
                    <xsl:element name="formula">
            <xsl:attribute name="id">
                <xsl:value-of select="'formula'"/>
            </xsl:attribute>
```

```
                    <xsl:value-of select="'Formeldarstellung '"/>
                    </xsl:element>
                </xsl:when>
                <xsl:otherwise>
                    <xsl:element name="fig">
            <xsl:attribute name="id">
                <xsl:value-of select="@id"/>
            </xsl:attribute>
                    <xsl:value-of select="'Abbildung '"/>
                    <xsl:apply-templates/>
                    </xsl:element>
                </xsl:otherwise>
            </xsl:choose>


    </xsl:template>

    <xsl:template match="caption">
        <xsl:element name="caption">
            <xsl:attribute name="id">
                <xsl:value-of select="'caption'"/>
            </xsl:attribute>
            <xsl:apply-templates/>
        </xsl:element>
    </xsl:template>

    <xsl:template match="graphic">
        <xsl:element name="graphic">
            <xsl:attribute name="href">
                <xsl:value-of select="@x-link:href"  xmlns:x-
link="http://www.w3.org/1999/xlink"/>
            </xsl:attribute>
        </xsl:element>
    </xsl:template>

    <xsl:template match="label">
        <xsl:apply-templates/>
    </xsl:template>




    <xsl:template match="back">
        <xsl:choose>
            <xsl:when test="count(//ack) > '0'">
                <xsl:element name="ack">
                    <xsl:attribute name="id">
                        <xsl:value-of select="'acknowledgement'"/>
                    </xsl:attribute>
                    <xsl:element name="title">
                        <xsl:value-of select="//back/ack/title"/>
                    </xsl:element>
                    <xsl:element name="p">
```

```
                        <xsl:value-of select="//back/ack/p"/>
                    </xsl:element>
                </xsl:element>
            </xsl:when>
        </xsl:choose>
        <xsl:for-each select="//back/ref-list/ref-list/ref">
            <xsl:element name="Literaturhinweis">
                <xsl:attribute name="publication-type">
                    <xsl:value-of select="mixed-citation/@publication-
type" />
                </xsl:attribute>
                <xsl:choose>
                    <xsl:when test="mixed-citation/@publication-type =
'other'">
                        <xsl:attribute name="mixed-citation">
                            <xsl:value-of select="mixed-citation" />
                        </xsl:attribute>
                    </xsl:when>
                    <xsl:otherwise>
                        <xsl:attribute name="author">
                            <xsl:value-of select="concat(mixed-
citation/person-group/name/surname,', ',mixed-citation/person-
group/name/given-names)" />
                        </xsl:attribute>
                        <xsl:attribute name="Titel">
                            <xsl:value-of select="mixed-citation/source"
/>
                        </xsl:attribute>
                        <xsl:attribute name="Jahr">
                            <xsl:value-of select="mixed-citation/year" />
                        </xsl:attribute>
                        <xsl:attribute name="Publizierungsort">
                            <xsl:value-of select="mixed-
citation/publisher-loc" />
                        </xsl:attribute>
                        <xsl:attribute name="Publizierungsname">
                            <xsl:value-of select="mixed-
citation/publisher-name" />
                        </xsl:attribute>
                    </xsl:otherwise>
                </xsl:choose>
            </xsl:element>
        </xsl:for-each>


    </xsl:template>




    <xsl:template match="text()|@*">
        <xsl:value-of select="."/>
    </xsl:template>
```

```
    <xsl:template match="table-wrap">
        <xsl:choose>
            <xsl:when test="count(./table/tbody/tr/td/inline-formula) >
'0'">
                <xsl:apply-templates/>
            </xsl:when>
            <xsl:otherwise>

                <xsl:element name="table">
                    <xsl:attribute name="id">
                        <xsl:value-of select="@id"/>
                    </xsl:attribute>
                    <xsl:value-of select="'Tabelle '" />
                    <xsl:apply-templates/>
                </xsl:element>
            </xsl:otherwise>
        </xsl:choose>

    </xsl:template>

    <xsl:template match="table">
        <xsl:apply-templates/>
    </xsl:template>

    <xsl:template match="tbody">
        <xsl:apply-templates/>
    </xsl:template>

    <xsl:template match="tr">


        <xsl:choose>
            <xsl:when test="count(./td/inline-formula) > '0'">
                <xsl:element name="formula">
                    <xsl:attribute name="id">
                        <xsl:value-of select="'formula'"/>
                    </xsl:attribute>
                    <xsl:value-of select="./td/bold" />
                    <xsl:value-of select="' '" />
                    <xsl:value-of select="./td/xref" />
                    <xsl:value-of select="' '" />
                    <xsl:value-of select="'Formeldarstellung'" />
                    <xsl:element name="graphic">
            <xsl:attribute name="href">
                <xsl:value-of select="./td/inline-
formula/alternatives/inline-graphic/@x-link:href"  xmlns:x-
link="http://www.w3.org/1999/xlink"/>
            </xsl:attribute>
        </xsl:element>
                </xsl:element>
```

```
            </xsl:when>
            <xsl:otherwise>
                <xsl:apply-templates/>
            </xsl:otherwise>
        </xsl:choose>

    </xsl:template>
<xsl:template match="tex-math">
</xsl:template>

    <xsl:template match="td">
        <xsl:element name="td">
        <xsl:apply-templates/>
        </xsl:element>
    </xsl:template>


    <xsl:template match="disp-formula">
        <xsl:element name="formula">
            <xsl:attribute name="id">
                <xsl:value-of select="'formula'"/>
            </xsl:attribute>
            <xsl:value-of select="'Formeldarstellung '" />
            <xsl:value-of select="'GL. '" />

        <xsl:apply-templates/>
        </xsl:element>
    </xsl:template>



</xsl:stylesheet>
```

Listing 20: XSLT-Skript zur Umformatierung der XML-Dateien

C Python-Skripte zur Verbesserung der Lemmaannotation

```
import os

lemma_final=""
lemma_final2=""
lemma_front=[]
lemma_list = set()
lemma_list_final=[]
i=0
tf ="/[PATH_TO_EXPORT_FOLDER].txt"

#Lese die Dateien ein
with open('/[PATH_TO_ANNIS_EXPORT].txt') as fu:
```

```
    for l in fu.readlines():

#Das Output-Format von ANNIS wird hier so verändert, dass es später
leichter in Python eingelesen werden kann: Am Ende dieses Prozesses steht
immer nur das Wort in einer Zeile
        try:
            lemma_front = l.split('[')
            lemma_final = lemma_front[1].split(']')
            #lemma_front = l.split()
            lemma_final2 = lemma_final[0]
            lemma_list.add(lemma_final2)
            i=i+1

        except(IndexError):
            print('Catch Error Text')

#Sortiere die Liste
lemma_list_final = sorted(lemma_list)

#Schreibe die Liste in eine neue Datei
with open(tf, mode="wb") as outfile:
    for x in lemma_list_final:
        outfile.write(x.encode())
        outfile.write(b"\n")

print(i)
```

Listing 21: Umformatierung der ANNIS-Exporte

```
import os
from nltk.metrics import edit_distance
import duden

THRESHOLD = 3  # maximale edit-distance

# Aufbau von list_of_unknown und list_of_known: eine Zeile pro Lemma
# list_of_known enthält alle Lemmata, die der Treetagger erkannt hat
# list_of_unknown alle Wortformen, die nicht erkannt wurden. In unseren
Daten stand an diesen Stellen der Tag im Lemma Feld "<unknown>".

#Lese die Daten ein
with open('/[PATH_TO_FILE_WITH_UNKNOWN_WORDS].txt') as fu,
open('/[PATH_TO_FILE_WITH_KNOWN_WORDS].txt') as fk:
    lemmas = {l.strip() for l in fk.readlines()}
    unknowns = {u.strip() for u in fu.readlines()}

#Versuche die unbekannten Wörter mit bekannten Lemmata zu matchen.
lemma_to_unknown = {l: [] for l in lemmas}
c = 0
done = set()
fail = set()
for lemma in lemmas:
    if not c % 100:
        print(c, 'of', len(unknowns), 'unknowns placed')
    for u in unknowns:
```

```
        #Wenn Wort kürzer oder gleich 3 Buchstaben, mache nichts
        if len(u) < 3 or u in done:
            continue
        #Wenn Lemma und Wort mit unbekannten Lemma gleich beginnen und
der Wortabstand zwischen den Wort kleinergleich 3 ist, matche das Wort
zum Lemma
        if u.startswith(lemma) and edit_distance(lemma, u) <= THRESHOLD:
            lemma_to_unknown[lemma].append(u)
            c += 1
            done.add(u)
            continue

        #Versuche über eine Anwendungsschnittstelle (API) des deutschen
Duden das Lemma zu ermitteln
        try:
            w = duden.get(u)
            print(w.name)
            lemma_to_unknown[lemma].append(w.name)

        except(AttributeError):
            fail.add(u)

#Schreibe alle Wörter in eine Textdatei, die noch immer kein bekanntes
Lemma haben
fail.sort()


#Schreibe die output-Dateien
with open('[EXPORT_PATH_TO_FILE_WITH_MAPPED_WORDS].txt', 'w') as f_out:
    f_out.write(os.linesep.join(':'.join([lemma,
','.join(sorted(lemma_to_unknown[lemma]))]) for lemma in lemmas))

with open('/[EXPORT_PATH_TO_FILE_WITH_MAPPED_WORDS_FAILED].txt',
mode="wb") as outfile:
    for x in fail:
        outfile.write(x.encode())
        outfile.write(b"\n")
```

Listing 22: Fix-Unknown-Skript

```
import os
from nltk.metrics import edit_distance

UNKNOWN = '<unknown>'


# unknowns -> Menge aller Lemmata, die nach der Anwendung des Mappings
noch unbekannt sind (die verbleibenden unknowns im Korpus haben immer
noch das "<unknown>"-Präfix)

#Lade die Datei der restlichen unbekannten Lemmata und der bekannten
Lemmata
with open('/[PATH_TO_FILE_WITH_STILL_UNKNOWN_WORDS].txt') as fu,
open('/[PATH_TO_FILE_WITH_KNOWN_WORDS].txt') as fk:
```

```
    unknowns = {u.strip() for u in fu.readlines()}
    proto_to_forms = {}
    done = set()
    c = 0
    for u in unknowns:
        for pot_prot in unknowns:
            if u != pot_prot and pot_prot not in done:
                # startswith-Kriterium (Wenn Lemma und Wort mit
unbekannten Lemma gleich beginnen und der Wortabstand zwischen den Wort
kleinergleich 3 ist) UND zweites Kriterium: weniger als 1/4 der längeren
Form darf abweichen (final)
                if u.startswith(pot_prot) and edit_distance(u, pot_prot)
/ len(u) < .25:
                    if pot_prot not in proto_to_forms:
                        proto_to_forms[pot_prot] = set()
                    proto_to_forms[pot_prot].add(u)
                    done.add(u)
                    break
        c += 1
        if not c % 100:
            print('{}/{}'.format(c, len(unknowns)))

    # Erstelle eine weitere Mapping-Datei (Die Zeilen sehen so aus:
Lemma: Form 1, Form 2, ...)
    with open('/[EXPORT_PATH_TO_SECOUND_FILE_WITH_MAPPED_WORDS].txt',
'w') as f:
        lines = [':'.join([prototype,
','.join(sorted(proto_to_forms[prototype]))])
                 for prototype in sorted(proto_to_forms.keys())]
        f.write(os.linesep.join(lines))

    #Man erhält zwei Mapping-Dateien, die verschiedene, flektierte (oder
anderweitig veränderte) Formen, z. B. koordiniert mit / oder -)
zusammenfasst und eine Form als Repräsentationsform wählt (nicht zwingend
Verb im Inf. oder Nomen im Nom. Sg. etc.)
```

Listing 23: Fix remaining unknown

```
import os
import glob
import csv
import duden

newDict={}

#Öffne die beiden erstellten Wörterbücher
with open('/[PATH_TO_SECOUND_FILE_WITH_MAPPED_WORDS].txt', 'r') as fur:
    for line in fur.read().split('\n'):
        lemma1 = line.split(':')
        basic = lemma1[0]
        newDict[basic] = basic
        add = lemma1[1].split(',')

        for i in add:
```

```
            newDict[i]=basic

with open('/[ PATH_TO_FILE_WITH_MAPPED_WORDS].txt', 'r') as fm:
    for line in fm.read().split('\n'):
        lemma1 = line.split(':')
        basic = lemma1[0]
        newDict[basic] = basic
        add = lemma1[1].split(',')

        for i in add:
            newDict[i] = basic


#Lege den Dateipfad fest
path = '/[PATH_TO_TAGGED_FILES]/'
ueberspringen=False

#Öffne Dateien
for filename in glob.glob(os.path.join(path, '*.tab')):

    filename_new = filename[0:-30] + ".tab"
    with open(filename) as fs:
        reader=csv.reader(fs, dialect='excel', delimiter='\t')
        for row in reader:
            #Workaround, da die XML-Dateien nicht komplett richtig nach
Konvention geschrieben wurden.
            if ueberspringen==True:
                ueberspringen = False
                continue

            #Ausbesserung von Sonderzeichen, welche kryptisch im Text
auftauchen
            if(row[0]==('&lt')):
                row[0]='<'
                row[1] = '$('
                row[2] = '<'
                row.extend('<')
                ueberspringen=True

            #Ausbesserung von Sonderzeichen, welche kryptisch im Text
auftauchen
            if (row[0] == '&gt'):
                row[0] = '>'
                row[1] = '$('
                row[2] = '>'
                row.extend('>')
                ueberspringen = True

            #Ausbesserung von Sonderzeichen, welche kryptisch im Text
auftauchen
            if (row[0] == '&amp'):
                row[0] = '&'
                row[1] = '$('
                row[2] = '&'
```

```
                row.extend('&')
                uebersprungen = True

            #Ausbesserung von Sonderzeichen, welche kryptisch im Text
auftauchen
            if (row[0] == '&quot'):
                row[0] = '"'
                row[1] = '$('
                row[2] = '"'
                row.extend('"')
                uebersprungen = True

            #Ausbesserung von Sonderzeichen, welche kryptisch im Text
auftauchen
            if (row[0] == '„'):
                row[0] = '"'
                row[1] = '§(('
                row[2] = '"'
                row.extend('"')

            #Ausbesserung von Sonderzeichen, welche kryptisch im Text
auftauchen
            if (row[0] == '‟'):
                row[0] = '"'
                row[1] = '§('
                row[2] = '"'
                row.extend('"')

            #Ab hier werden die <unknown>-Lemma bearbeitet.
            #If-Abfrage notwendig, da nur der Zeitschriftentext und keine
XML-Elemente betrachtet werden sollen. Der Zeitschriftentext steht in
drei tab-getrennten Spalten: tok row[0], pos row[1], lemma row[2].  XML-
Elemente sind nur eine Spalte lang.
            if len(row)==3:
                #Wenn ein Lemma bekannt ist, kopiere das Lemma in
lemma_cor
                if (row[2] != '<unknown>'):
                    row.append(str(row[2]))

                #Gehe in dieses IF, wenn das Lemma unbekannt
                if (row[2] == '<unknown>'):
                    #Suche nach dem Lemma in den geöffneten Wörterbüchern
                    rowAppendDic= str(newDict.get(row[0]))
                    #Wenn das Wort kürzer oder gleich 3 Buchstaben,
übernehme das Wort als Lemma. (So können Chemische Formeln oder
Produktbezeichnungen übernommen werden)
                    if (len(row[0])<=3):
                        row.append(row[0])

                    #Falls auch im Wörterbuch nichts gefunden, setze für
lemma_cor ebenfalls <unknown>
                    if (rowAppendDic=="None"):
                        row.append('<unknown>')
```

```
                    #In allen anderen Fällen, setze für lemma_cor den
gefundenen Eintrag aus dem Wörterbuch
                    else:
                        row.append(rowAppendDic)

            #Schreibe die Daten wieder in eine Datei
            with open(filename_new, 'a') as f:
                f.write('\t'.join(row[0:]) + '\n')
```

Listing 24: Einfügen der verbesserten Lemmaannotation