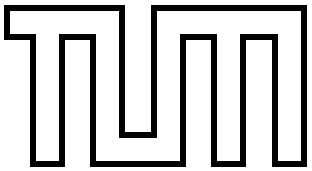


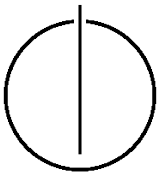
FAKULTÄT FÜR INFORMATIK

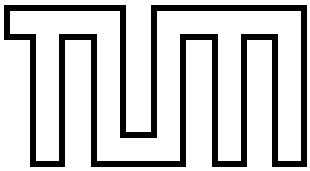
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatik

Spezifikation und Extraktion von Semantischen Mustern in Deutschen Gesetzen auf Basis von Linguistischen Eigenschaften unter der Verwendung von Apache Ruta

Patrick Ruoff





FAKULTÄT FÜR INFORMATIK

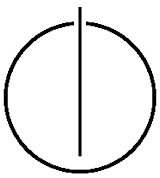
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatik

Spezifikation und Extraktion von Semantischen Mustern in Deutschen Gesetzen auf Basis von Linguistischen Eigenschaften unter der Verwendung von Apache Ruta

Specification and Extraction of Semantic Patterns in German Laws based on Linguistics Features using Apache Ruta

Autor:	Patrick Ruoff
Aufgabensteller:	Prof. Dr. Florian Matthes
Betreuer:	Bernhard Waltl, M. Sc.
Abgabedatum:	17. Oktober 2016



Ich versichere, dass ich diese Bachelor's Thesis selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 20. Oktober 2016 Patrick Ruoff

Danksagung

An erster Stelle bedanke mich bei meinem Betreuer Bernhard Waltl, der diese Bachelorarbeit erst möglich gemacht und mit großem Interesse und Engagement verfolgt hat. Mein besonderer Dank gilt zudem Konrad Heßler für die Einführung in die juristischen Details und die harmonische Zusammenarbeit. Außerdem bedanke ich mich beim Herrn Prof. Matthes dafür, dass ich diese Abschlussarbeit an dessen Lehrstuhl für Software Engineering betrieblicher Informationssysteme verfassen durfte.

Abstract

Die juristische Arbeit besteht zu einem großen Teil aus dem Durchsuchen von Normen, Urteilen und Gesetzen nach für den Einzelfall relevanten Informationen. Da dies ein sehr daten-, zeit- und wissensintensiver Prozess ist, gewinnen computergestütze Anwendungen in dieser Domäne immer mehr an Bedeutung. Infolgedessen entstand kürzlich auch die Data-Science Umgebung Lexia am Lehrstuhl Software Engineering betrieblicher Informationssysteme der Technischen Universität München (TUM). In Lexia ist die regelbasierte Text-Annotationssprache Ruta von Apache UIMA (Unstructured Information Management Architecture) integriert, um mit Mitteln der Sprachverarbeitung deutsche Gesetze zu annotieren.

In dieser Bachelorarbeit wird zunächst eine Einführung in einige Anwendungen der Textverarbeitung natürlicher Sprache gegeben. Anschließend wird in Zusammenarbeit mit Konrad Heßler der Versuch unternommen, die funktionale Klassifikation von Rechtsnormen zu automatisieren. Dazu werden syntaktische Regeln spezifiziert, die der Identifikation der Funktion von Normen dienen. Daraufhin werden diese Normen innerhalb von Lexia als Ruta-Skripte implemeniert, ausgeführt und getestet. Die resultierenden Annotationen werden untersucht, um eine erneute Überarbeitung der syntaktischen Regeln vorzunehmen, womit der Arbeitszyklus von neuem beginnt. Während des Arbeitsprozesses sind Probleme aufgetreten, die nicht gelöst werden konnten und zu gewissen Einschränkungen führten. In einer abschließenden Evaluation wurden unter anderen die Auswirkungen dieser Einschränkungen festgestellt. Die evaluierten Kennzahlen der resultierten Klassifikation legen nahe, dass sie für die praktische Anwendung im juristischen Alltag noch ungeeignet ist. Es wurde jedoch eine Grundlage für zukünftige Arbeiten geschaffen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AGL Anspruchsgrundlage

BGB Bürgerliches Gesetzbuch

FN False Negatives

FNR False Negatives Rate

FP False Positives

FPR False Positive Rate

GmbHG Gesetz betreffend die Gesellschaften mit beschränkter Haftung

LMU Ludwig-Maximilian-Universität München

NAGL Nicht-Anspruchsgrundlage

NER Named-Entitity-Recognition

NLP Natural Language Processing

NRM Nicht-Rechtsmacht

NRP Nicht-Rechtspflicht

POS Part-of-speech

RM Rechtsmacht

RP Rechtspflicht

Ruta Rule Based Text Annotation

SR Sonstige Rechtsvorschriften

STTS Stuttgart-Tübingen Tagset

SV Soll-Vorschrift

TP True Positives

TPR True Positive Rate

TUM Technische Universität München

UIMA Unstructured Information Management Architecture

WFP Weak False Positives

WFPR Weak False Positives Rate

WTP Weak True Positives

WTPR Weak True Positive Rate

1. Einleitung und Überblick

1.1. Motivation

Die Arbeit von Juristen besteht zu einem großen Teil aus dem Durchsuchen von Normen, Urteilen und Gesetzen nach für den Einzelfall relevanten Informationen. Dieser ohnehin daten-, zeit- und wissensintensive Prozess wird zusätzlich durch die stetig steigende Menge an Dokumenten, die berücksichtigt werden müssen, erschwert. Allein in der aktuellen Wahlperiode seit dem Jahr 2013 wurden bisher 323 Gesetze verabschiedet [9]. Die Informatik kann an dieser Stelle mit Mitteln des Natural Language Processing (NLP) Juristen bei ihrer Arbeit assistieren. So entstanden im Rahmen des neu erschlossenen Gebietes der Rechtsinformatik einige Systeme die beispielsweise die Kanzleiverwaltung und auch spezifischere Anwendungen, die Aufgaben, wie das Extrahieren der expliziten Informationen eines Textes, in hohem Maße unterstützen. In diesem Zuge wurde auch das Forschungsprojekt Lexalyze des Lehrstuhls für Software Engineering betrieblicher Informationssysteme der Technischen Universität München (TUM) gegründet. Im Rahmen dieses Projekts entstand auch die vorliegende Bachelorarbeit, deren Ziel es ist eine Anwendung zu schaffen, die die funktionale Klassifizierung von Rechtsnormen assistiert und umsetzt. Zu diesem Zweck wird auf Methoden des NLP zurückgegriffen, von denen einige im nächsten Abschnitt im Rahmen einer allgemeinen Einführung in das NLP erklärt werden. Im nächsten Kapitel wird anschließend das Ziel dieser Bachelorarbeit näher spezifiziert. Kapitel 3 beschreibt die Art und Weise, wie diese Ziele umgesetzt werden sollen bevor in Kapitel 4 das dabei verwendete Konzept erläutert wird. In Kapitel 5 wird anhand zweier Teilschritte die Umsetzung der Klassifikation und die dabei auftretenden Probleme beschrieben. Die Resultate werden dann in einer abschließenden Evaluation in Kapitel 6 bewertet. Ergebnis der Evaluation ist, dass die entwickelte Klassifikation nicht für eine praktische Anwendung tauglich ist. Am Ende der Bachelorarbeit ist der Inhalt in Kapitel 7 nochmals zusammengefasst und ein Ausblick für zukünftige Erweiterungen wird vorgenommen.

1.2. Natural Language Processing

NLP ist ein Forschungsgebiet das untersucht, wie natürliche Sprache von Computern verstanden und verarbeitet werden kann. Unter natürlicher Sprache wird dabei das Produkt menschlicher Kommunikation verstanden und beschränkt sich in dieser Bachelorarbeit auf die textuelle Form. Der Ursprung von NLP liegt in einer Vielzahl von Gebieten wie der Informatik, Linguistik, Elektrotechnik, Mathematik, Künstliche Intelligenz, Robotik und Psychologie. Andererseits eröffnet das NLP auch einige Disziplinen wie die maschinelle Übersetzung, Text Verarbeitung natürlicher Sprache, Spracherkennung, Zusammenfassungen etc. pp. Im Folgenden werden einige Beispiele für Anwendungen des NLP be-

schrieben. Diese sind vorrangig dem Gebiet der Text Verarbeitung natürlicher Sprache zuzuordnen.

Eine Eigenschaft der Programme der Verarbeitung natürlicher Sprache ist, dass sie zunächst unabhängig von der Anwendung sind und erst später auf die einzelnen Domänen zugeschnitten werden. So gibt es zahlreiche kleinere Teilbereiche, die unabhängig voneinander umgesetzt werden können. In den Applikationen werden dann die entsprechenden Programme hintereinander ausgeführt, sodass der Output des einen Programms der Input des darauf Folgenden ist etc. pp. Diese Struktur wird auch *Pipes and Filter Architecture* genannt, da die Programme in dieser Metapher als Filter den Eingabetext verarbeiten und dazwischen die Daten durch Pipelines in eine Richtung transportiert werden.

Sehr viele Anwendungen des NLP sind von Sprache zu Sprache verschieden. So kann beispielsweise die Segmentierung nach Morphen im Englischen sehr einfach umgesetzt werden, während sie im Türkischen gar nicht umsetzbar ist [16]. Die Anwendungen in dieser Bachelorarbeit beziehen sich stets auf das Deutsche.

Die in diesem Abschnitt vorgestellten Verfahren werden teilweise an anderer Stelle wieder aufgegriffen. Der Abschnitt dient jedoch hauptsächlich dazu in das NLP im Allgemeinen einzuführen. Es werden die vier Bereiche der lexikalischen Analyse, der morphologischen Analyse, der syntaktischen Analyse und der semantischen Analyse unterschieden.

1.2.1. Lexikalische Analyse

Für die linguistische Untersuchung eines Textes ist es zunächst notwendig, ihn in seine einzelnen linguistischen Segmente aufzuteilen. Dieser Prozess wird auch Lexikalische Analyse oder Segmentierung bezeichnet. Er stellt in den meisten Anwendungen die Basis für weitere Analyseschritte dar. Die aus der Lexikalischen Analyse resultierenden Komponenten werden Segmente genannt. Dieser Begriff wird in verschiedenen Anwendungen unterschiedlich definiert. Ein Segment kann beispielsweise eine Silbe, ein Wort, einen Satz oder auch einen Absatz darstellen. Im Folgenden wird auf die ersten drei Varianten eingegangen. Um eine Segmentierung an die Strukturen der jeweils zu behandelnden Texte anpassen zu können, besteht dieser aus mehreren einzelnen aufeinander aufbauenden Modulen, die unterschiedliche, zum Teil aufeinander aufbauende Arbeitsschritte erledigen.

Segmentierung nach Sätzen

Einen Text in seine Sätze zu unterteilen ist ein fundamentaler Schritt des Strukturierungsprozesses natürlicher Sprache. Es gibt verschiedene Vorgehensweisen, die verschiedene Eigenschaften und Schwachstellen haben. Der naivste Ansatz ist, alle Zeichenfolgen zwischen zwei Punkten als Satz zu deklarieren. Diese Methode schlägt allerdings fehl, wenn der gegebene Text Abkürzungen mit folgendem Punkt oder andere Formate, wie beispielsweise Ordinalzahlen oder Datumsangaben, die Punkte verwenden, enthält. Eine erweiterte Strategie, die diesen Fehlern vorbeugt ist [17] entnommen und lautet wie folgt.

1. Wenn ein Punkt gefunden wird, markiert er das Ende eines Satzes.
2. Wenn das vor dem Punkt platzierte Segment in der Liste der Abkürzungen steht, markiert er nicht das Ende des Satzes.

3. Wenn das nächste Wort mit einem Großbuchstaben beginnt, markiert der Punkt das Ende des Satzes

Die verbleibenden Fehler dieser Vorgehensweise sind vor allem Nennungen von Namen im Format „F. Matthes“. Da diese Schreibweise im Deutschen allerdings kaum Gebrauch findet, gilt dieser Algorithmus als eine sehr simple und effiziente Art der Segmentierung [17]. In der Praktischen Umsetzung dieser Bachelorarbeit wird in Abschnitt 5.1 eine an die Domäne angepasste Definition des Satzes getroffen. Demnach werden Sätze nicht nur durch Punkte sondern auch durch Semikolon und Doppelpunkte voneinander getrennt.

Segmentierung nach Wörtern

Eine weitere grundlegende Segmentierungsstufe ist diejenige nach Wörtern. Sie wird auch Tokenisierung genannt und geht nahezu allen Analysen auf höheren syntaktischen und semantischen Ebenen voraus. Auch für die Umsetzung der funktionalen Klassifikation ist diese Form der Segmentierung unerlässlich. Nachfolgend werden alle Phasen der Tokenisierung im Einzelnen dargestellt. Die Beschreibung dieser Phasen lehnt sich an die Ausführungen in [18].

Am Anfang des Prozesses steht die sogenannte White-Space-Segmentierung, wodurch Tabulatoren, Zeilenumbrüche und Leerzeichen aus dem Text entfernt werden. Die Ausgabe nach diesem Schritt ist, wie nach allen Schritten, eine Liste aller bisher als Segment identifizierten Zeichenfolgen. Der nächste Arbeitsschritt handhabt die Trennung von Schrägstrichschreibungen, wie „und/oder“. Die Ausnahme bilden Zahlen, die mit einem Schrägstrich getrennt sind, wie „2016/17“, und Einheitsangaben, die mindestens zu einem Teil nur aus einem Buchstaben bestehen, wie „l/m“. Daraufhin werden die Satzzeichen, die vor einem Leerzeichen stehen, als eigenes Segment deklariert. Dabei werden Trennungszeichen am Zeilenende besonders betrachtet. Es wird unterschieden zwischen Phrasen mit Ergänzungsstrich, wie „Haupt- und Nebensatz“, und solchen mit Bindestrich, wie in „Preis-Leistungs-Verhältnis“. Erstere werden in mehrere Token unterteilt, zweitere bilden ein einzelnes. Der nächste Arbeitsschritt besteht darin, von Leerzeichen getrennte Zahlen, wie „100 000“, zu einem Token zusammenzufassen. Ist der Segmentierung nach Wörtern noch keine Segmentierung nach Sätzen vorangegangen, so muss man diese letztlich noch durchführen, um im Text stehende Punkte am Wortende richtig einzuordnen. Punkte können z. B. Ordinalzahlen, Abkürzungen oder ein Satzende markieren.

Segmentierung nach Silben

Die feinste in diesem Abschnitt behandelte Segmentierung extrahiert aus den gewonnenen Wörtern zusätzlich die Silben. Diesen Vorgang nennt man auch Worttrennung. In aktuellen Anwendungen wird diese Art der Segmentierung größtenteils mit Hilfe von Wörterbüchern mit Informationen zur Silbentrennung umgesetzt. Die früher verwendeten algorithmischen Methoden stoßen schnell an ihre Grenzen, da die Silbe in der Sprachwissenschaft als kleinste Lautgruppe im natürlichen Sprechfluss definiert ist und nicht stets formal definierbaren Regeln folgt.

Ein typisches direktes Anwendungsgebiet der Segmentierung nach Silben sind Texteditoren. Wörter, die über die Zeile hinausreichen würden, können zwischen zwei passenden Silben getrennt werden anstatt sie in der darauffolgenden Zeile zu positionieren.

1.2.2. Morphologische Analyse

Die Morphologie untersucht als Teildisziplin der Linguistik die Gesetzmäßigkeiten der Strukturen von Wörtern. Im Zentrum der Betrachtung stehen dabei häufig die sogenannten Morphe, die kleinste sinnbehaftete Einheit in der Sprachwissenschaft. Sie werden in Klassen der Morpheme kategorisiert und aus ihnen kann man Genus, Kasus, Numerus und Person des Wortes ableiten. Ein Wort kann immer dann in Morphe zerlegt werden, wenn diese Morphe in der selben Form und Bedeutung auch in anderen Wörtern vorkommen [14]. Zum Beispiel wird das Wort „geht“ in die Morphe „geh-“ und „-t“ unterteilt. „-geh“ kommt als Wortstamm auch mit der selben Bedeutung in „be-geh-bar“ und „-t“ als Endung auch in „spiel-t“ vor. Im Folgenden wird in das Part-of-speech Tagging und das Stemming eingeführt. Für beide ist die Zerlegung der Wörter in ihre Morphe nicht immer relevant.

Part-of-speech Tagging

Im Part-of-speech (POS) Tagging wird den Wörtern eines Textes die Wortart zugeordnet. Die im Deutschunterricht behandelte händische Einordnung der Wortarten in Nomen, Verben, Adjektive usw. ist bei genauerer Betrachtung nicht vollständig. Jede dieser Hauptwortarten kann weiter subklassifiziert werden. Im Stuttgart-Tübingen Tagset (STTS) werden beispielsweise 54 Tags, also Wortarten, unterschieden [12]. Diese Tags bestehen jeweils aus einer Haupt- und einer oder mehreren Subklassen. Neben dem STTS gibt es noch viele weitere Tagsets allerdings gilt es im Deutschen als Standard [19]. Das POS Tagging kann nicht als simples Lesen von Wortarten aus einem Wörterbuch umgesetzt werden, da ein Wörter in vielen verschiedenen Wortarten auftreten. Das Wort „einen“ wird beispielsweise sowohl als Artikel als auch als Verb verwendet. Im STTS wird es einem der drei Tags *ART*, für Artikel, *VVINF*, für infinites Verb, und *VVFIN*, für finites Verb, zugeordnet. Um die Wortart dennoch mit hoher Wahrscheinlichkeit richtig zuzuordnen wurden einige regelbasierte und stochastische Verfahren der Kategorisierung entwickelt. In einem stochastischen Verfahren werden zum Beispiel zunächst alle möglichen Tag-Folgen für eine Folge von Wörtern ermittelt. Daraufhin werden die Auftrittswahrscheinlichkeiten der Tag-Folgen anhand gegebener Daten verglichen und die wahrscheinlichste ausgewählt.

Stemming

In der Computerlinguistik wird mit Wörtern oftmals nicht in ihrer ursprünglichen Form, sondern als deren extrahierter Wortstamm, gearbeitet. Der Vorgang der Bildung des Wortstamms trägt die Bezeichnung Stemming. Es ist das Verfahren, das die Wörter „ankommt“ und „angekommen“ auf den gemeinsamen Wortstamm „ankomm“ zurückführt. Es sei angemerkt, dass das primäre Ziel des Stemming nicht darin liegt, die exakte Grundform „ankommen“ zu finden, was Gegenstand der Grundformreduktion ist. Mit Stemming werden lediglich gleiche morphologische Varianten eines Wortes auf den selben Ausdruck zurückgeführt, was für die meisten Anwendungen ausreicht [11].

Stemming wird beispielsweise in Suchmaschinen genutzt, um die Suche zu vereinfachen und zu beschleunigen, weil dafür nicht die grammatikalischen Eigenschaften eines Wortes entscheidend sind, sondern dessen Bedeutung. Durch Stemming können verschie-

dene Wortformen zu dem selben Term zusammengefasst werden. Somit werden Verweisstrukturen verkleinert, was eine Suche beschleunigt. Das Verwalten mehrerer Wortformen unter einem Term hat außerdem eine inhaltliche Verallgemeinerung der Wörter zur Folge. Eine Suche wird daher mehr Übereinstimmungen finden können.

Einfache Algorithmen des Stemming verwenden Lookup-Tabellen, um einem Wort dessen Wortstamm zuzuordnen. So erfolgt die Zuordnung schnell und etwaige Ausnahmen, die regelbasierte Algorithmen komplizierter machen, sind einfach zu handhaben. Beispielsweise sind Konstrukte wie „kommen (...) an“ in regelbasierten Verfahren äußerst schwer umzusetzen. Andererseits kann bei unbekannten und neuen Wörtern kein Wortstamm in der Lookup-Tabelle gefunden werden und somit keine Zuweisung erfolgen.

Ein regelbasierter Algorithmus ist der im Englischen geläufige Porter-Stemmer, zu dem es auch eine Adaption für das Deutsche gibt [11]. Im Rahmen dieser Adaption wird zunächst der Eingabetext verändert, indem Umlaute durch die jeweiligen Vokale und „ß“ durch „ss“ ersetzt werden. Danach werden bestimmte Wortendungen unter festgesetzten Bedingungen am Ende des Wortes abgeschnitten [10]. So erhält man vor allem im Englischen sehr akkurate Ergebnisse, weil es dort die oben erwähnten Konstrukte nicht gibt.

1.2.3. Syntaktische Analyse

In der syntaktischen Analyse wird die Struktur von Sätzen näher betrachtet. Es gibt verschiedene Modelle, die einer syntaktischen Analyse zugrunde liegen können. Die Modelle unterscheiden sich vor allem hinsichtlich des Detailgrads der grammatikalischen Einordnung und in der Herangehensweise, die Satzstruktur aufzubrechen. Manche dieser Modelle, wie das im Folgenden beschriebene Chunking, betrachten nur die grobe Struktur des Satzes, wohingegen andere eine tiefere grammatikalische Analyse vornehmen und beispielsweise Subjekte von Prädikatobjekten unterscheiden.

Den meisten syntaktischen Analyseverfahren geht eine eine Segmentierung nach Sätzen, Wörtern und Morphen, sowie ein POS-Tagging voran. Im folgenden werden die Begriffe der Dependenzgrammatik, Konstituentengrammatik und Chunking voneinander abgegrenzt und erläutert.

Dependenzgrammatik

Die Dependenzgrammatik basiert auf der Annahme, dass die syntaktische Struktur eines Satzes durch binäre, asymmetrische Dependenzrelationen zwischen Wörtern darstellbar ist. Jede dieser Dependenzrelationen beschreibt eine Abhängigkeit eines sog. Depentent, dem regierten Wort, von einem Regent, dem regierenden Wort. Außerdem wird die Art der Beziehung der beiden Wörter, der Dependenz Typ festgelegt.

Weiterhin gilt für die Dependenzgrammatik, wie für die anderen syntaktischen Modelle auch, die Grundannahme, dass jedem grammatikalisch korrekten Satz eine eindeutige, dem Modell zugrunde liegende Struktur zugewiesen werden kann.

Wie im Beispiel aus Abbildung 1.1 dargestellt ist, besteht eine syntaktische Beziehung des Wortes ”Klausmit dem Wort ßchreibt”. Dadurch wird angezeigt, dass der Dependent ”Klausïn seiner Funktion als Subjekt (`SUBJ`) von dem Regent ßchreibtäbhängt. Auf diese Weise wird über dem gesamten Satz durch die Dependenzrelationen ein Dependenzbaum gespannt, dessen Wurzel mit `ROOT` markiert wird.

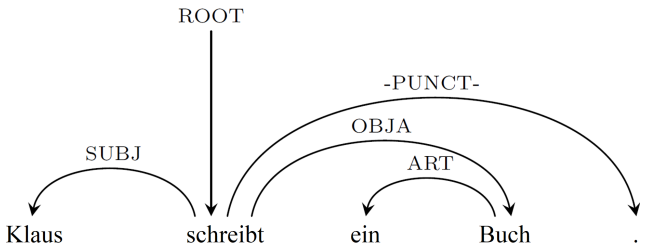


Abbildung 1.1.: Ein Satz mit Darstellung seiner Dependenzrelationen
Quelle: Selbst erstellt

Die automatisierte Zuweisung eines Dependenzbaumes zu einem Eingabesatz erfolgt durch Dependenz Parser, die im Allgemeinen zwei verschiedenen Ansätzen folgen. Grammatikbasierte Dependenz Parser verwenden feste Regeln und führen Nichtterminale ein, um den korrekten Dependenzbaum zu erstellen. Verschiedene Implementierungen sind näher in Kapitel 3 und 4 von [6] erläutert. Auf der anderen Seite stehen datengetriebenen Ansätze, die für die Erstellung des Dependenzbaumes Methoden des maschinellen Lernens verwenden. In Kapitel 5 aus [6] werden hierfür zwei Ansätze erläutert.

Kontituentengrammatik

Im Gegensatz zur Dependenzgrammatik basiert die Konstituentengrammatik auf Teil-Ganzes-Beziehungen zwischen Phrasen des Satzes. Als Phrase wird im Folgenden eine beliebige lange Folge von Wörtern und Satzzeichen, die Teil eines einzelnen Satzes sind, bezeichnet. Das der Konstituentengrammatik zugrunde liegende Prinzip besteht darin, dass zuerst jedem Wort ein Nichtterminal zugeordnet wird, das dessen Wortart beschreibt. Daraufhin werden immer zwei Nichtterminale zusammengefasst, sodass die entstehende Phrase ein Nichtterminal auf einer höheren Ebene bildet. Dieser Schritt wird so lange wiederholt, bis der gesamte Satz von einem Nichtterminal, der Wurzel `S`, umfasst wird. Die Nichtterminale werden auch als Konstituenten (Bestandteil) bezeichnet.

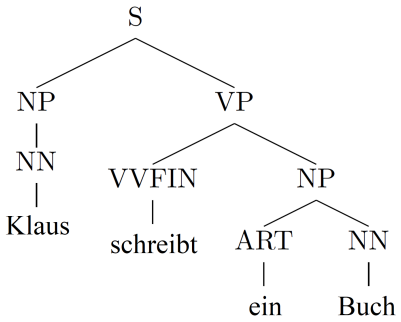


Abbildung 1.2.: Ein Satz mit Darstellung seiner Konstituenten
Quelle: Selbst erstellt

Wendet man dieses Verfahren auf den vorherigen Beispielsatz an, so erhält man die Struktur, wie in Abbildung 1.2 dargestellt. Hier werden die Konstituenten `ART` (Artikel) und `NN` (Nomen) der Wörter ëinünd "Buchßu einer Nominalphrase (`NP`) zusammenge-

fasst, wodurch ihre Beziehung markiert wird. Diese Phrase wird daraufhin mit dem finiten Verb ßchreibtßu einer Verbalphrase (VB) vereint. Im letzten Schritt wird die Verbalphrase mit dem Wort "Klaus"vereint, dem zunächst aufgrund der Wortart das Nichttermnal NN zugeordnet wurde und daraufhin ebenfalls NP. Das resultierende Nichtterminal S umfasst den ganzen Satz.

Analog zur Dependenzgrammatik gibt es auch bei der Konstituentengrammatik Parser, die einem datengetriebenen oder einen grammatikbasierten Ansatz folgen. Der grammatikbasierte Ansatz wird jedoch häufiger gewählt, weil die Konstituentenbäume intuitiv als Grammatik ausgedrückt werden können. Ein Beispiel hierfür ist der CKY-Algorithmus, der in Absatz 3.4.2 von [2]ausführlich beschrieben ist.

Chunking

Als dritte Form der syntaktischen Analyse wird im Folgenden kurz auf das Chunking eingegangen. Chunking ist eine partielle syntaktische Analyse und berechnet somit nicht die volle Dependenz und Konstituenz eines Satzes. Es betrachtet ebenfalls Wortsequenzen und versucht strukturelle Beziehungen in und zwischen ihnen zu identifizieren, um Phrasen gewisse Typen, wie Nomenphrasen, Verbphrase, Adjektivphrasen etc. zuzuweisen. Jedoch können bestimmte Teilstrukturen dabei nebengeordnet und in ihrer syntaktischen Funktion unbestimmt bleiben [7].

Abney sieht Chunking aus einer psycholinguistischen Sicht [1]:

> „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]"

Das Chunking ist also nur eine grobe Einteilung einzelner grammatikalischer Phrasen eines Satzes. Für manche Anwendungen ist das allerdings ausreichend. Vorteile des Chunking gegenüber einer Analyse hinsichtlich der Dependenz oder Konstituenz sind die Geschwindigkeit der Algorithmen und die verringerte Komplexität der Chunks. Außerdem sind die Verfahren sehr robust, weil nicht die der ganze Text eingeordnet werden muss, womit schwierige Spezialfälle umgangen werden [3].

1.2.4. Semantische Analyse

Die höchste Form der textuellen Analyse ist die semantische Analyse. Ihr Ziel ist es, anhand eines expliziten Eingabetextes, dessen implizite Bedeutung zu extrahieren. Diese Bedeutung ist dabei unabhängig von der verwendeten Sprache. Während die bisher vorgestellten Ebenen der Verarbeitung natürlicher Sprache meist hinreichend gelöst wurden, sind einige Herausforderungen der semantischen Analyse noch immer Gegenstand aktueller Forschung.

Named Entity Recognition

Unter Named Entity Recognition versteht man, wie der Name bereits erkennen lässt, den Vorgang der Erkennung und Klassifizierung von benannten Entitäten eines Textes. Bei-

spiele für benannte Entitäten sind einerseits konkrete Dinge der Wirklichkeit, wie Personen, Orte und Organisationen, und andererseits auch Währungs-, Maß- und Zeitangaben. Die Herausforderung besteht unter anderem darin, unterschiedliche Schreibweisen der selben Entität zu identifizieren. In Text natürlicher Sprache ist mit „U.S.A." beispielsweise das selbe gemeint wie mit „Vereinigte Staaten" oder teilweise auch nur „Staaten". Noch signifikantere Unterschiede gibt es zum Beispiel bei den Zeitangaben. „Zehn Minuten Fünf" beschreibt die selbe Uhrzeit wie „17:10 Uhr". Es gibt sowohl grammatikbasierte als auch datengetriebene Ansätze zur Umsetzung der NER, wobei erstere typischerweise einen größeren Teil der benannten Entitäten erkennen (True Positive Rate). Diese Ansätze markieren allerdings auch mehr Wörter als benannte Entität, die gar keine sind (False Positive Rate) [15].

Semantic Role Labeling

Das Semantic Role Labeling beschäftigt sich mit der Erkennung und Einordnung der semantischen Argumente des Prädikats eines Satzes. Beispielsweise hat das Verb *„kaufen"* typischerweise drei Argumente. Es wird gefragt **wer** etwas kauft, **was** gekauft wird und **von wem** es gekauft wird. Zusätzlich kann noch nach anderen Kriterien wie dem Zeitpunkt oder dem Ort der Handlung gefragt werden. Die Antworten auf die Fragen können dabei verschiedenste Formate annehmen, wodurch die Umsetzung erschwert wird. Jurafsky & Martin erläutert dies am Beispiel eines Unternehmens XYZ, das Aktien erwirbt. In den folgenden Sätzen nimmt XYZ stets die selbe semantische und eine unterschiedliche syntaktische Rolle ein, wobei jedes Mal die selbe Handlung beschrieben wird [8].

- XYZ hat die Aktie gekauft.
- Sie verkauften die Aktie an XYZ.
- Die Aktie wurde von XYZ gekauft.
- Der Kauf der Aktie durch XYZ...
- Der Aktienkauf von XYZ...

Das Semantic Role Labeling ist essentiell, um die Aussage eines Satzes für Computer verständlich zu machen.

2. Problembeschreibung

In diesem Kapitel wird der Arbeitsauftrag dieser Bachelorarbeit näher spezifiziert. In dem Zuge wird die im Titel verwendete allgemeine Formulierung näher präzisiert.

Diese Bachelorarbeit beschäftigt sich im Kern mit der Spezifikation und Extraktion semantischer Muster in Deutschen Gesetzen. Dazu werden linguistische Eigenschaften betrachtet und mit Hilfe der regelbasierten Textannotationssprache Apache Ruta umgesetzt. Die betrachteten semantischen Muster stellen funktionale Klassen von rechtlichen Normen dar. Ziel der Arbeit ist es, die Erkennung verschiedener funktionaler Klassen zu automatisieren. Einige wesentlichen Informationen zu dieser Klassifikation sind in Abschnitt 2.1 festgehalten. Welche Klassen behandelt werden, wird im darauffolgenden Abschnitt 2.2 näher beschrieben. Es werden zunächst syntaktische Regeln, die die jeweiligen Klassen ausmachen, aus linguistischen Überlegungen heraus spezifiziert. Diese Regeln werden anschließend in Apache Ruta implementiert und in der Data-Science Umgebung Lexia ausgeführt, um zunächst deren korrekte Umsetzung festzustellen und weiterhin deren semantische Trefferquote und Genauigkeit zu evaluieren. Eine Einführung in Lexia und dem dort verwendeten Apache Ruta ist in Abschnitt 2.3 zu finden.

2.1. Funktionale Klassifikation von rechtlichen Normen

Ein Großteil der Informationen in diesem Abschnitt sind aus Unterlagen und Erklärungen von Konrad Heßler, Wissenschaftlicher Mitarbeiter am Lehrstuhl für Bürgerliches Recht, Handels- und Gesellschaftsrecht, Privatrechtstheorie der Ludwig-Maximilians-Universität München (LMU) zusammengefasst, die keine weitere Quellenangabe ermöglichen.

Normen können anhand ihrer Funktion unterschiedlichen funktionalen Klassen zugeordnet werden. Diese jeweilige Funktion ergibt sich dabei nicht zwingend unmittelbar aus dem Gesetzeswortlaut. Es gibt Normen deren Funktion erst durch ihre Rolle im Gefüge der Regelungen ersichtlich wird, weswegen auch andere Normen zur Identifizierung ihrer Funktion betrachtet werden müssen. Es gibt mehrere Ansätze Normen nach ihrer Funktionalität zu klassifizieren. Es kann zum Beispiel zwischen einer äußeren und einer inneren Klassifikation unterschieden werden. Bei der äußeren, auf linguistischen Eigenschaften basierten Klassifikation, ergibt sich die Zuordnung unmittelbar aus dem Inhalt der Norm. Wohingegen für eine innere Klassifikation zusätzlich die Funktion der Norm im Regelungsgefüge betrachtet wird. Auch die Rolle der Norm in der Anwendungspraxis ist dabei von Bedeutung.

Gegenstand dieser Arbeit ist die äußere funktionale Klassifikation. Durch die Entwicklung von Ruta-Skripten kann die Syntax der Normen analysiert und ein beschränkter Schluss auf deren Semantik gezogen werden. Es ist allerdings stets zu beachten, dass eine solche Klassifikation gewisse Normen, deren Funktionalität nicht ausschließlich durch den Wortlaut sondern zusätzlich durch die Rolle im Regelungsgefüge gebildet wird, nicht

immer korrekt einordnen kann.

2.2. Funktionale Klassen von Normen

Die funktionalen Klassen, die in dieser Bachelorarbeit extrahiert und implementiert werden, werden im Folgenden beschrieben. Wichtig sind dabei zwei juristische Begriffe. Zum einen der des *Anspruchs*, der in §194 Abs. 1 des BGB legaldefiniert ist. Ein Anspruch ist demnach „das Recht, von einem anderen ein Tun oder Unterlassen zu verlangen“. Zum anderen ist der Begriff des *Rechtssubjekts* zentral. Ein Rechtssubjekt nach [4] ist, wer fähig ist, Rechte und Pflichten zu tragen.

Rechtsmacht Die Klasse all jener Normen, die einem Rechtssubjekt das Recht zu einem Tun oder Unterlassen einräumen, wird im Folgenden Rechtsmacht (RM) genannt.

Nicht-Rechtsmacht Die Normen, die negativ feststellen, dass eine bestimmte Rechtsmacht nicht oder nur in begrenztem Umfang besteht, haben die Funktionalität Nicht-Rechtsmacht (NRM)

Rechtspflicht Der funktionalen Klasse Rechtspflicht (RP) gehören die Normen an, die einem Rechtssubjekt eine Pflicht zu einem Tun oder einem Unterlassen auferlegen.

Nicht-Rechtspflicht Als Negation der Rechtspflicht, beschreibt die Klasse Nicht-Rechtspflicht (NRP) die Normen, die feststellen, dass einem Rechtssubjekt eine Pflicht zu einem Tun oder einem Unterlassen nicht oder in begrenztem Umfang auferlegt wird.

Anspruchsgrundlage Eine Anspruchsgrundlage (AGL) ist eine Norm, die einem Rechtssubjekt, dem Anspruchsinhaber, die Rechtsmacht gibt, von einem anderen Rechtssubjekt, dem Anspruchsgegner, ein Tun oder Unterlassen zu verlangen. Es sind also lle diejenigen Normen, die einen Anspruch von einem Rechtssubjekt an ein anderes beschreiben und bildet somit eine Unterklasse der Rechtsmacht und Rechtspflicht.

Nicht-Anspruchsgrundlage Jeder Rechtssatz, der die Feststellung enthält, dass ein Anspruch nicht oder in begrenztem Umfang besteht, ist Element der Klasse Nicht-Anspruchsgrundlage (NAGL). Sie ist eine Unterklasse zu Nicht-Rechtsmacht und Nicht-Rechtspflicht.

Soll-Vorschrift Wird in einer Norm einem Rechtssubjekt ein bestimmtes Verhalten in einer Weise nahegelegt, die hinter einer Rechtspflicht zurückbleibt, so gehört diese Norm der Klasse Soll-Vorschrift (SV) an.

Während die Begriffe der Anspruchsgrundlage und der Soll-Vorschrift im Allgemeinen einheitlich verstanden werden, sind die Begriffe der Rechtspflicht, Rechtsmacht und deren Negationen in anderer Literatur möglicherweise abweichend definiert. Sie wurden vom Domänenexperten Konrad Heßler auf diese Art beschrieben.

Wurde einer Norm die funktionale Klasse zugeordnet, so ist es eine weitere Aufgabe, einzelne Bestandteile, die zu der Klasse gehören, zu identifizieren. Eine Anspruchsgrundlage enthält beispielsweise einen *Anspruchsinhaber*, einen *Anspruchsgegner*, den *Anspruchsinhalt* und potenzielle *Tatbestandsvoraussetzungen*. In Kapitel 5 wird der Versuch unternommen auch diese und weitere Phrasen in den Normen zu erkennen.

2.3. Lexia

Die Umsetzung der funktionalen Klassifikation erfolgt in dieser Arbeit auf der Data-Science-Umgebung Lexia. Lexia entstand als Web-Anwendung im Zuge des Forschungsprojekts Lexalyze vom Lehrstuhl für Software Engineering betrieblicher Informationssysteme der TUM. In Lexia ist Apache UIMA (Unstructered Information Management Architecture) integriert. Diese NLP-Architektur stellt die Ruta (Rule-based Text Annotation) Engine zur Verfügung. Ruta besitzt eine imperative Text-Annotationssprache, die eine Folge von Annotationen mit verschiedenen Bedingungen auswertet und im Falle einer Übereinstimmung eine Menge von Aktionen ausführt [5]. In Lexia wird das Framework DKPro Core der Technischen Universität Darmstadt bereitgestellt. Dadurch kann auch verschiedenste NLP-Komponenten, wie einem Segmenter oder POS-Tagger zugegriffen werden. Lexia stellt außerdem einen Text-Editor zum Erstellen der Ruta Skripte bereit. Gesetze können als XML-Dateien importiert werden, um auf ihnen die Skripte auszuführen. Abbildung 2.1 zeigt die Ansicht eines annotierten Gesetzes in Lexia.

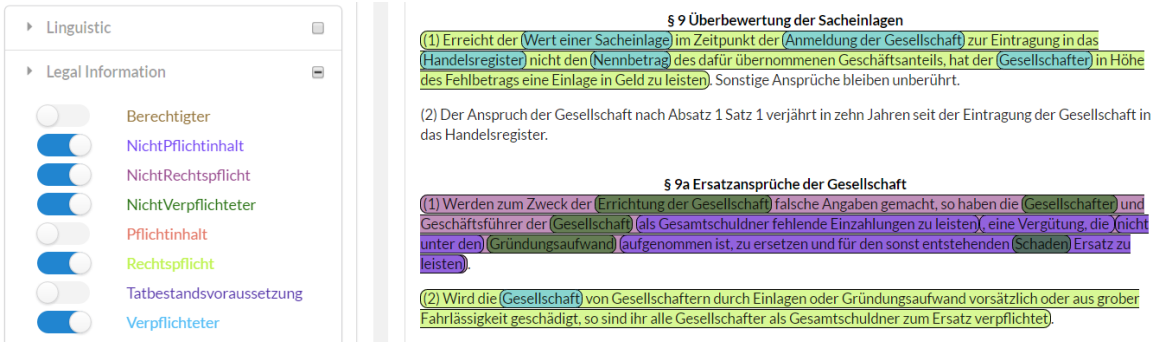


Abbildung 2.1.: Screenshot der Webansicht von Lexia im Browser
Quelle: Selbst erstellt

3. Methode

In diesem Kapitel wird die Arbeitsweise zur Erstellung der funktionalen Klassifikation erläutert, die im Rahmen dieser Bachelorarbeit verwendet wird. Sie lehnt sich an die Explorativen Datenanalyse wie sie von John W. Tukey beschrieben wurde [13]. Es wird ein Zyklus von drei Positionen gebildet, der solange wiederholt wird, bis ein gewisses Ergebnis erzielt wird.

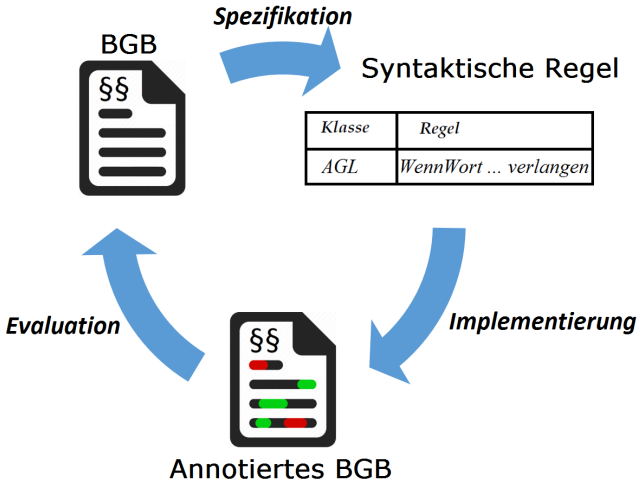


Abbildung 3.1.: Schemenhafte Darstellung des Arbeitsprozesses
Quelle: Selbst erstellt

Der Arbeitsprozess ist in Abbildung 3.1 dargestellt und beginnt bei dem Artefakt des Bürgerlichen Gesetzbuches (BGB). Es werden darin die Normen betrachtet, denen eine der in Abschnitt 2.2 beschriebenen Funktionen zugeordnet werden kann. Anhand wiederkehrender linguistischer Eigenschaften, die für die jeweilige funktionale Klasse bezeichnend scheinen, werden syntaktische Regeln für diese Klasse spezifiziert. Beispielsweise ist die Formulierung „*Rechtssubjekt1* kann von *Rechtssubjekt2* ... verlangen“ bezeichnend für eine Anspruchsgrundlage. Derartige Formulierungen können sowohl direkt aus den Normen gelesen werden als auch intuitiv für die jeweiligen funktionalen Klassen erschlossen werden. Der Gesetzgeber hat den Anspruch, dass rechtliche Normen von jedem Bürger ohne intensives Vorstudium verstanden werden können sollen. Aus diesem Grund sollten Normen in einer einfachen, natürlichen Sprache formuliert werden. Wiederkehrende Satzkonstrukte in Gesetzen erfüllen einerseits diesen Anspruch und erleichtern andererseits deren automatisierte Klassifikation.

Es resultiert das zweite Artefakt, syntaktische Regeln für die Identifizierung der Funktion einer Norm. Diese werden im Anschluss als Ruta-Skript in Lexia implementiert. Zum Testen wurden Beispielnormen erstellt und in Lexia importiert, um sicherzustellen, dass die Regeln korrekt umgesetzt wurden. Anschließend werden die Skripte auf dem BGB

ausgeführt. Die resultierenden Annotationen sind die Grundlage für die Evaluation der Regeln. Anhand von ihnen werden die erstellten Regeln erneut hinsichtlich ihrer semantischen Zutrefflichkeit und Vollständigkeit hinterfragt, womit der Arbeitszyklus von vorne beginnt.

Sind nach einer Evaluation keine weiteren Änderungen mehr nötig oder denkbar, so wird der Zyklus unterbrochen. Die erhaltenen Regeln und Skripte werden dann in einer abschließenden Evaluation auf dem Gesetz betreffend die Gesellschaften mit beschränkter Haftung (GmbHG) bewertet.

Über den gesamten Prozess wird die Extraktion und Spezifikation vom Domänenexperten Konrad Heßler vollzogen, der Mitarbeiter am Lehrstuhl für Bürgerliches Recht, Handels- und Gesellschaftsrecht, Privatrechtstheorie der juristischen Fakultät der LMU ist. Der Autor, Patrick Ruoff, setzt die Implementierung der syntaktischen Regeln, das anschließende Testen der Ruta-Skripte und die abschließende Evaluation und Dokumentation um. Beide stehen dabei in engen Kontakt.

4. Konzept & Design

4.1. Konzept der semantischen Muster

Um die Regeln zur Erkennung der semantischen Muster von funktionalen Klassen von Normen eindeutig beschreiben zu können, wurde eine eigene Notation festgelegt. Ein Auszug der Notation ist in Tabelle 4.1 dargestellt.

Tabelle 4.1.: Auszug der Notation für die Regeln zur Erkennung funktionaler Klassen

Notation	**Bedeutung**
Text	Das Wort Text in Groß- und Kleinschreibung
Text 1/Text 2/.../Text n	Genau eines der Elemente muss vorliegen
[Text]	Platzhalter einer grammatikalischen Funktion bzw. Regel
(Text)	Der Teil zwischen den Klammern ist optional
...	Beliebig viele (auch keine) Zeichen, aber ohne Punkt, Semikolon oder Doppelpunkt dazwischen
Text \	Regel ist nicht erfüllt, wenn der zwischen \und \liegende Text an dieser Stelle steht.
Text	Der hinterlegte Teil der Regel soll mit einem Tag versehen werden. Den unterschiedlichen Farben ist ein Tag zugeordnet
-I *A / B / C* *I-*	Die zwischen *-I I-* stehenden Elemente können in beliebiger Reihenfolge stehen aber müssen mindestens ein Mal vorkommen. Sonstiger Text zwischen den einzelnen Elementen wird als SONST bezeichnet

Die in diesem Auszug nicht aufgeführten Elemente der Notation finden kaum Anwendung oder sind für die Beispiele, die im Folgenden gegeben werden nicht relevant. Bei Bedarf wurde die Notation im Zuge der Spezifikation der Muster stets erweitert und angepasst.

4.2. Hierarchie der Skripte

In diesem Abschnitt werden die Beziehungen der implementierten Ruta-Skripte beschrieben. Im Anhang (A) sind davon ausgewählte Skripte hinterlegt. Die Ruta-Skripte werden aus Gründen der Übersicht und verbesserten Handhabung, wie in Abbildung 4.1 dargestellt, in vier Schichten unterteilt. Das Skript der ersten Schicht besteht aus sehr simplen Hilfsregeln und heißt EinfacheHilfregeln.ruta. Das Sktipt der zweiten Schicht trägt den Namen KomplexeHilfsregeln.ruta und enthält die restlichen, Hilfsregeln. Auf-

grund eines nicht gelösten Problems bei der Implementierung, das in Abschnitt 5.3.2 beschrieben wird, muss das Skript für die Anspruchsgrundlage auf der dritten Ebene zweigeteilt werden, wo auch die Soll-Vorschriften eingeordnet werden. Da insgesamt zehn Regeln für die Anspruchsgrundlage erstellt werden, wurden die entstandenen Skripte Anspruchsgrundlage1-6.ruta und Anspruchsgrundlage7-10.ruta genannt. In der vierten und höchsten Ebene sind die Ruta-Skripte für die Rechtspflicht und Rechtsmacht. Sie greifen beide auf Anspruchsgrundlage7-10.ruta zu, weil dort die nocheinmal einzelne Hilfsannotationen definiert werden, die ausschließlich in den drei Skripten gebraucht werden.

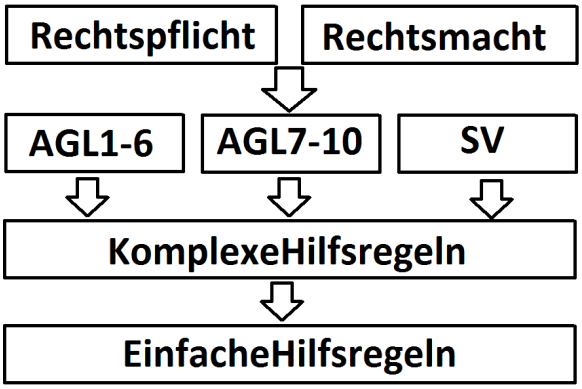


Abbildung 4.1.: Die Beziehungen unter den Ruta-Skripten
Quelle: Selbst erstellt

5. Umsetzung & Implementierung

In diesem Kapitel ist festegehalten, wie die syntaktischen Regeln spezifiziert wurden und sich die dazugehörigen Ruta Skripte sich im Arbeitsprozess entwickelt haben. Nach einigen allgemeinen Anmerkungen wird zunächst ein repräsentatives Beispiel einer Regel vom Beginn des Prozesses gegeben. Im Anschluss wird in Abschnitt 5.2 die finale Form der selben Regel näher erläutert. Im darauffolgenden Abschnitt wird ein Ausblick auf die Weiterentwicklung der Muster im Zuge erweiterter technischer Möglichkeiten gegeben. Abschließend werden in Abschnitt 5.3 die Grenzen der Implementierung näher beleuchtet.

5.1. Erster Versuch

Die Spezifikation der funktionalen Klassen wurde anhand der Methode aus Kapitel 3 umgesetzt. Zu Beginn wurden von Konrad Heßler, einem Experten in der Domäne der funktionalen Klassifizierung rechtlicher Normen, am BGB erste syntaktische Regeln für die betrachteten Klassen extrahiert. Diese Regeln waren in ihrer Definition recht intuitiv gehalten und gaben stellten die grundlegende Idee dar, wie die Funktion der Norm erkannt werden sollte. Sie bestanden vor allem aus Folgen von konkreten Wörtern und Platzhaltern für bestimmte Wortarten. Getrennt wurden diese Phrasen zumeist durch Wildcards wie sie als „...“ in Tabelle 4.1 definiert sind.

Bei allen syntaktischen Regeln wurde festgelegt, dass diese stets einen ganzen Satz umfassen sollen. Als Satz wird dabei jede Zeichenfolge betrachtet, die durch einen Punkt, einen Doppelpunkt oder ein Semikolon von einem anderen Satz getrennt ist. Die einzige Ausnahme bilden Punkte, die in Form der Abkürzung „Abs.“ vorkommen. Die genannten Satzzeichen zur Trennung von Sätzen werden im Folgenden als Satzendung bezeichnet.

Das folgende Beispiel zeigt beispielhaft eine der ersten aus dem BGB extrahierten Regeln.

Regelbeispiel 5.1

Klasse	***Regel***
Anspruchsgrundlage	*... , {wenn/sofern/sobald/soweit/insoweit/falls}* *... , {kann/können/darf/dürfen/ist gestattet}* *... [Substantiv]* *... (verlangen/fordern)* *... {./;/:}*

Ein Satz der von dieser Norm als Anspruchsgrundlage erkannt wird, enthält die folgenden Phrasen in der genannten Reihenfolge.

- Ein Komma gefolgt von einem der Wörter „wenn“, „sofern“, „sobald“, „soweit“, „insoweit“ oder „falls“

- Ein Komma gefolgt von einer der Phrasen „kann“, „können“, „darf“, „dürfen“ oder „ist gestattet“
- Ein Substantiv
- Eines der Wörter „verlangen“ oder „fordern“
- Eine Satzendung

Zwischen diesen Phrasen dürfen wegen „...“ beliebig viele weitere Zeichen außer Satzendungen stehen. Auf die Angabe von „. . . “ am Anfang und „. . . {./;/:}“ am Ende der Regeln wird im Folgenden verzichtet, da sie bei jeder Regel auf Ebene der funktionalen Klassen gleich ist. Um diese Regel zu implementieren sind einige Schritte notwendig die im folgenden erläutert werden.

Weil die Wörter der ersten beiden Punkte der Aufzählung in vielen Regeln und wiederholt vorkommen, wurde für sie und andere wiederkehrende Wortgruppen jeweils Hilfsregeln entworfen.

Regelbeispiel 5.2

Klasse	***Regel***
WennWort	{*wenn/sofern/sobald/soweit/insoweit/falls*}

Regelbeispiel 5.3

Klasse	***Regel***
KannWort	{*kann/können/darf/dürfen/ist gestattet*}

Die Regelbeispiele 5.2 und 5.3 sind zwei dieser Hilfregeln. Ihre Implementierung In Apache Ruta gestaltet sich nach folgendem Muster.

Codebeispiel 5.4

```
DECLARE WennWort;
W{REGEXP("Wenn|wenn")->WennWort};
W{REGEXP("Sofern|sofern")->WennWort};
W{REGEXP("Sobald|sobald")->WennWort};
W{REGEXP("Soweit|soweit")->WennWort};
W{REGEXP("Insoweit|insoweit")->WennWort};
W{REGEXP("Falls|falls")->WennWort};
```

In der ersten Zeile wird mit dem DECLARE-Befehl die Annotation *WennWort* eingeführt. In Ruta wird nach jeder Regel ein Semikolon gesetzt, um ihr Ende zu kennzeichnen. In der zweiten Zeile wird für jedes Wort (`W`) des Eingabetextes durch einen regulären Ausdruck (REGEXP) verglichen, ob es das Wort „Wenn“ oder „wenn“ ist, weil nach der Notation sowohl groß- als auch kleingeschriebene Wörter gesucht werden. Ist das der Fall, so wird es als *WennWort* annotiert. Das selbe wird in der dritten Zeile für das Wort „sofern“ fortgesetzt etc. pp. Analog wurde auch das *KannWort* und andere einfache Hilfsregeln implementiert, die ausschließlich nach konkreten Wörter suchen.

Um das „...“ der Notation, also eine beliebig lange Zeichenkette, die keine Satzendung enthält zu implementieren, wurde zu Beginn das Wildcard-Zeichen `#` von Ruta benutzt,

das für eine beliebige Menge von Zeichen steht, bis das darauffolgende Regelelement gefunden wird. Nach ersten Versuchen kam es jedoch zu erheblichen Verschlechterungen in der Laufzeit bei Verwendung von zwei oder mehr Wildcards pro Regel. Aus dem Grund wurde die Hilfsregel *WCOMMA* eingeführt. Sie ist definiert als alle Wörter und Zeichen ausgenommen der Satzendungen. Der überwiegende Großteil ihrer Elemente sind also Wörter und Kommas. Außerdem gehören ihr auch andere Satz- und Sonderzeichen an.

Codebeispiel 5.5 `DECLARE WCOMMA;`

```
Token{
AND(NOT(IS(PERIOD)),NOT(IS(SEMICOLON)),NOT(IS(COLON))
-> WCOMMA};

// Ausnahme bei "Abs."
W{REGEXP("Abs")} PERIOD{-> MARK(WCOMMA)};
```

Codebeipiel 5.5 zeigt die Implementierung von *WCOMMA*. Die Annotation *Token*, die in der zweiten Zeile verwendet wird, wurde durch eine importierte Tokenisierung erzeugt. In der dritten Zeile wird eine klassische logische Abfrage gemacht. `IS(PERIOD)` wird wahr, wenn das gegebene Token von Typ *PERIOD* (zu deutsch Punkt) ist. Analog funktionieren `IS(SEMICOLON` für das Semikolon und `IS(COLON)` für den Doppelpunkt. In der letzten Zeile wird die Ausnahme umgesetzt, dass Punkte nach der Abkürzung „Abs“ nicht als Satzendung gelten.

Somit entsteht die Implementierung der ersten Regel zur Erkennung einer Anspruchsgrundlage.

Codebeispiel 5.6 `DECLARE Anspruchsgrundlage;`

```
WCOMMA* @COMMA WennWort
WCOMMA*? COMMA  KannWort
WCOMMA*? pos.N
WCOMMA*? W{REGEXP("verlangen"|"fordern")}
WCOMMA*)
{AND(-PARTOF(Anspruchsgrundlage))->MARK(Anspruchsgrundlage)};
```

Diese Implementierung erinnert stark an die Darstellung der Regel durch die getroffene Annotitaon. Das resultiert aus der intuitiven, regelbasierten Struktur der Ruta Skripte. Der `*`-Operator aus Zeile zwei zeigt an, dass an der Stelle in der Norm beliebig viele Elemente der Hilfsegel *WCOMMA* stehen dürfen. Mit dem `@`-Operator wird dem Compiler gezeigt, dass die Erkennung dieser Regel bei dem ersten Komma (`COMMA`) der Regel beginnen soll. So wird Zeit bei der Ausführung der Regel gespart, weil sonst jedes Wort als Beginn der Regel behandelt wird und mit dem Zusatz nur die Kommas. Ab der dritten Zeile wird anstatt dem `*`- der `*?`-Operator nach dem *WCOMMA* gesetzt. Er steht ebenfalls dafür, dass an der Stelle eine beliebige Anzahl an *WCOMMA* stehen kann mit der Erweiterung, dass die Regel nur terminiert, wenn das nächste Regelelement gefunden wird, bevor die Folge von *WCOMMA* abbricht. In der vierten Zeile wird auf ein importieres Part-of-speech Tagging zugegriffen. `pos.N` steht für die Subjektive eines Textes. Die Regel endet mit der Markierung als *Anspruchsgrundlage*, sofern der Text nicht bereits eine *Anspruchsgrundlage* enthält. Dieser Zusatz bewirkt, dass mit dem abschließenden `WCOMMA*` nur die

längste Kette von *WCOMMA*-Annotationen mit in die *Anspruchsgrundlage* aufgenommen wird und nicht alle möglichen Teilketten. Es wird also alles bis zur nächsten Satzendung annotiert.

Durch diese Beispiele sollte die Vorgehensweise bei der Umsetzung und Implementierung vermittelt werden. Im nächsten Abschnitt wird die finale Fassung der selben Beispielregel 5.1 gegeben und ihre Implementierung gezeigt.

5.2. Finale Form

Nach einigen Durchläufen des in Kapitel 3 beschriebenen Arbeitszyklus, entstanden für alle betrachteten funktionalen Klassen finale Regeln für deren Erkennung.

Zusätzlich zur Identifizierung der funktionalen Klasse einer Norm wurde auch der Versuch unternommen, in den Normen die Rechtssubjekte in ihren verschiedenen Rollen, die Inhalte des Anspruchs, der Soll-Vorschrift, der Rechtspflicht oder der Rechtsmacht und potentielle Tatbestandsvoraussetzungen zu erkennen. Die Regeln wurden mit dem Ziel spezifiziert, dass auch diese Phrasen in den Normen gefunden werden können. Es wurde eine farbliche Notation eingeführt, um die Stellen zu kennzeichnen, an denen eine der genannten Annotationen stehen kann.

Für die Nicht-Anspruchsgrundlage mussten keine eigenen Regeln erstellt werden, denn sie wird, wie auch die anderen negativen Klassen, im Zuge ihres positiven Äquivalents gesucht. Enthält eine Anspruchsgrundlage, Rechtspflicht oder Rechtsmacht eine bestimmte Art von vordefinierter *Negation* im Bereich der als *SONST* gekennzeichnet ist, so wird die erkannte Norm als die jeweilige negative Klasse behandelt. Mit dieser und einigen anderen Erweiterungen entstand die finale Form der Regel aus dem ursprünglichen Regelbeispiel 5.1.

Regelbeispiel 5.7

Klasse	***Regel & Annotationen***
Anspruchsgrundlage *Falls SONST eine Negation enthält:* *Nicht-Anspruchsgrundlage*	*{[Wenn-Wort]/[Wenn-Wort2]} ... ,* *,([DannWort]) [KannWort]* *-I* *[Anspruchsinhaber N] /* *{verlangen/fordern} /* *({gegenüber/von/vom} [Anspruchsgegner D]) /* *([WennWort] ... <Ende des Halbsatzes>) /* *([AusnahmePhrase]) /* *(SONST)* *I-*

Das Regelbeispiel 5.7 enthält einige kleinere Erweiterungen zur Strukturellen Erkennung. Zum Beispiel kann nun zwischen dem Komma und dem *KannWort* optional ein *DannWort* sein und der erste Teil der Regel wird als *Tatbestandsvoraussetzung* markiert. Das wird durch die grüne Markierung angezeigt. Welche Farbe für welche Markierung steht, ist in Tabelle 5.1 dargestellt. Daneben gab es auch größere Änderungen, wie die

Tabelle 5.1.: Zuordnung von Farben zu den Rechtsbegriffen

Tag
Tatbestandsvoraussetzung
Anspruchsinhalt
Anspruchsinhaber
Anspruchsgegner

Überarbeitung der Erkennung von Substantiven. Mit der Hilfsregel *AnspruchsinhaberN* sollten alle potentiellen Rechtssubjekte, die als Anspruchsinhaber in Frage kommen und im Nominativ stehen, erkannt werden. Analog dazu wird jedes Rechtssubjekt als *AnspruchsgegnerD* markiert, das ein möglicher Anspruchsgegner ist, der im Dativ steht. Diese beschriebenen Hilfregeln werden auf der zweiten Ebene der Ruta-Skripte (Abschnitt 4.2) umgesetzt.

Die signifikanteste Veränderung zu den ersten Regeln ist der Teil, der als *-I I-* gekennzeichnet ist. Nach der definierten Notation muss in dem Bereich zwischen dem *KannWort* und der Satzendung ein *AnspruchsinhaberN* und eines der Wörter „verlangen“ oder „fordern“ stehen, wobei der *AnspruchsinhaberN* als *Anspruchsinhaber* markiert wird. Dieser soeben genannte Bereich ist grundlegend für die weitere Klassifikation. In ihm kann optional auch die Aneinanderreihung eines der Wörter „gegenüber“, „von“ oder „vom“ gefolgt von einem *AnspruchsgegnerD* sein. In dem Fall wird der Anspruchsgegner als solcher annotiert. Ebenfalls kann der Bereich die Phrase von einem *WennWort* bis zum Ende des Halbsatzes stehen, was als *Tatbestandsvoraussetzung* annotiert wird. Ein Halbsatz ist in diesem Sinne das Ende des Hauptsatzes, der durch Nebensätze verlängert werden kann. Außerdem können vordefinierte *AusnahmePhrasen* in dem genannten Bereich sein, die dann als *Tatbestandsvoraussetzung* markiert werden. Ist in dem Bereich noch mehr Text als die genannten Phrasen, so fallen diese Stellen in die Kategorie *SONST*. Die Phrasen von *SONST* werden als *Anspruchsinhalt* markiert. Enthält eine der Phrasen von *SONST* eine vordefinierte Form der *Negation*, so wird die Norm anstatt als *Anspruchsgrundlage* als *Nicht-Anspruchsgrundlage* markiert. Ist das der Fall, so werden weiterhin abgesehen von den *Tatbestandsvoraussetzungen* die getroffenen Annotationen negiert. Der *Anspruchsinhaber* wird stattdessen als *NichtAnspruchsinhaber*, der *Anspruchsinhalt* als *NichtAnspruchsinhalt* und der *Anspruchsgegner* als *NichtAnspruchsgegner* markiert.

Um die folgende Implementierung des Regelbeispiels nachvollziehen zu können, benötigt man grundlegende Kenntnisse in Apache Ruta. Durch Kommentare werden die einzelnen Glieder der Regel erklärt.

Codebeispiel 5.8

```
// Erkennung der ganzen Norm als
// mögliche Anspruchsgrundlage
(WCOMMA* W{OR(IS(WennWort), IS(WennWort2))}
#{AND(-CONTAINS(COLON),-CONTAINS(SEMICOLON),-CONTAINS(PERIOD))}
@COMMA DannWort? KKDD WCOMMA*)
{AND(-PARTOF(Anspruchsgrundlage))->MARK(Anspruchsgrundlage)};

// Regeln innerhalb des BLOCK-Statements werden nur innerhalb
```

```
// der Anspruchsgrundlage-Annotation ausgeführt
BLOCK(ForEach) Anspruchsgrundlage{} {

//Markieren des Bereichs "-I  I-" als AnspruchsgrundlageEnde
W{OR(IS(WennWort), IS(WennWort2))}
#{AND(-CONTAINS(COLON),-CONTAINS(SEMICOLON),-CONTAINS(PERIOD))}
@COMMA DannWort? KKDD WCOMMA*{->AnspruchsgrundlageEnde};

// Checken der Bedingungen
AnspruchsgrundlageEnde{AND(-CONTAINS(AnspruchsInhaberN),
-CONTAINS(FordernVerlangen))-> UNMARK(AnspruchsgrundlageEnde)};

// Regeln innerhalb des BLOCK-Statements werden nur innerhalb
// der AnspruchsgrundlageEnde-Annotation ausgeführt;
// Wird übersprungen, falls vorherige Bedingungen nicht erfüllt
BLOCK(ForEach) AnspruchsgrundlageEnde{} {

// XXToken markiert alle Wörter, die als XX markiert sind
// Wenn_ = "WennWort ... <Ende des Halbsatzes>"
// Der Abschnitt sortiert Annoationen aus SONST aus
WCOMMA*{AND(-CONTAINS(AnspruchsinhaberNToken),
-CONTAINS(FordernVerlangen),
-CONTAINS(GegenAnspruchsinhaberDToken),
-CONTAINS(Wenn_Token),
-CONTAINS(AusnahmePhraseToken),
-PARTOFNEQ(Sonst)) // nur die größte zusammenhängende Phrase
->MARK(Sonst)};

// Boolsche Variable ContainsNegationInSonst
ASSIGN(ContainsNegationInSonst, false);

// Wenn Sonst eine Negation enthält, wird die Variable true
Sonst{CONTAINS(Negation) ->
ASSIGN(ContainsNegationInSonst,true)};

//Sonstige Annotationen werden hinzugefügt
Sonst{CONTAINS(W),-ContainsNegationInSonst
->Anspruchsinhalt};
Sonst{CONTAINS(W),ContainsNegationInSonst
->NichtAnspruchsinhalt};
Wenn_{-> Tatbestandsvoraussetzung};
AusnahmePhrase{-> Tatbestandsvoraussetzung};

} //Ende des BLOCKs AnspruchsgrundlageEnde

// Speichert, ob die Norm postitiv oder negativ ist
```

```
AnspruchsgrundlageEnde{ContainsNegationInSonst
->UNMARK(AnspruchsgrundlageEnde),
MARK(NichtAnspruchsgrundlageEnde)};

// Fügt die erste Tatbestandsvoraussetzung dazu
(W{OR(IS(WennWort),IS(WennWort2))} W*)
{PARTOF(Anspruchsgrundlage)->Tatbestandsvoraussetzung}
COMMA;

} //Ende des BLOCKs Anspruchsgrundlage

// Wenn Negation in Sonst -> NichtAnspruchsgrundlageEnde
Anspruchsgrundlage{CONTAINS(NichtAnspruchsgrundlageEnde)
-> MARK(NichtAnspruchsgrundlage)};

// Falls Bedingungen nicht erfüllt oder Negation in Sonst
// handelt es sich nicht um eine AGL oder NAGL
Anspruchsgrundlage{-CONTAINS(AnspruchsgrundlageEnde)
->UNMARK(Anspruchsgrundlage)};
```

Insgesamt entstanden in dieser Form zehn Regeln für die Anspruchsgrundlage, vier für die Rechtspflicht, sechs für die Rechtsmacht und zwei für die Soll-Vorschriften. Die Regeln haben im Wesentlichen alle die selbe Struktur, wie sie in Regelbeispiel 5.7 dargestellt ist. Jede Implementierung wurde auf Beispielnormen in Lexia getestet.

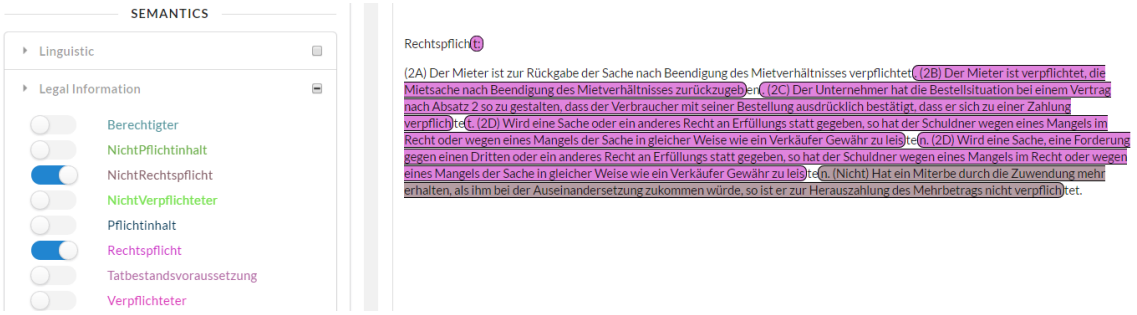


Abbildung 5.1.: Screenshot der Annotationen der Rechtspflicht in Lexia
Quelle: Selbst erstellt

In Abbildung 5.1 sind die Annotationen zu sehen, die durch das Rechtspflicht-Skript auf den Beispielnormen erzeugt werden. Die Annotation in der ersten Zeile wird in Lexia lediglich fehlerhaft dargestellt und bezieht sich auf den ersten vollständigen Satz.

5.3. Grenzen der Implementierung

Manche Hindernisse, die während des Arbeitsprozesses auftraten, konnten nicht ohne negativen Auswirkungen überwunden oder umgangen werden und sind in diesem Abschnitt erläutert.

5.3.1. Erkennung des Kasus von Subjekten

Die korrekte Erkennung des Kasus der Subjekte eines Textes ist äußerst wichtig, um die Einordnung von Annotationen, wie *AnspruchsinhaberN* und die der anderen verwendeten Rechtssubjekte, richtig ausführen zu können. Nur so können diese Annotationen sinnvoll vorgenommen werden ohne übermäßig viele Fehler zu machen. Für die Implementierung dieser Regeln konnte im Rahmen der technischen Möglichkeiten jedoch nicht auf den Kasus eines Subjekts zugegriffen werden. Somit wurde somit ein Kompromiss umgesetzt der lediglich einen Großteil der im Dativ stehenden Subjekte auch dem Nominativ zuordnet und anders herum.

5.3.2. Teilung der Anspruchsgrundlage

Das signifikanteste Problem, das nicht behoben werden konnte, trat bei der Implementierung der zehn Regeln der Anspruchsgrundlage auf. Die Ruta Skripte werden aus Gründen der Übersicht und verbesserten Handhabung in in vier Schichten unterteilt, wie es in Abschnitt 4.2 beschrieben wird. Beim Versuch die zehn Regeln der Anspruchsgrundlage in einem Skript in Lexia auszuführen, wird ein Fehler gemeldet, dass eine Annotation nicht aufgelöst werden kann, als wäre die Annotation nicht definiert. Bildet man zwei einzelne Ruta-Skripte für die Anspruchsgrundlage und teilt auf ihnen die Regeln auf, die im Groben die selben Hilfsregeln verwenden, so tritt dieser Fehler nicht mehr auf, ohne das an der Struktur oder dem Inhalt der Regeln etwas verändert wurde. Dieses Verhalten legt die Vermutung nahe, dass mit dem großen Skript eine Ruta-interne Speichergrenze überschritten wird, weil die geteilten Skripte für sich weniger Regeln und Annotationen definieren. Nach einer Recherche konnte dazu allerdings nichts näheres gefunden werden. Aufgrund dieser Gegebenheit entstanden zwei separate Skripte mit vier und sechs Regeln für die Anspruchsgrundlage. Auch jeder Versuch diese beiden Skripte nacheinander auszuführen scheiterte. Es ist also nicht gelungen in einem Dokument alle zehn Regeln der Anspruchsgrundlage auszuführen.

Dieses Problem nimmt noch größere Ausmaße an, weil die Anspruchsgrundlage als Unterklasse der Rechtspflicht und Rechtsmacht auch auf deren Regeln in der vierten Ebene Einfluss nimmt. Wird eine Norm als Anspruchsgrundlage erkannt, so ist es automatisch auch eine Rechtspflicht und eine Rechtsmacht. Da man jedoch nicht auf das Ergebnis aller Regeln der Anspruchsgrundlage zugreifen kann, können die Regeln der Rechtsmacht und Rechtspflicht auch nicht vollständig umgesetzt werden.

5.3.3. Erkennung von Nebensätzen und Aufzählungen

Ein weiteres Problem stellt die Implementierung der Regel `[WennWort] ... <Ende des Halbsatzes>` dar, die für jede der 22 Regeln zur Erkennung der funktionalen Klassen verwendet wird. Ein Halbsatz ist in diesem Sinne das Ende des Hauptsatzes, der durch Nebensätze verlängert werden kann. Der Sinn dahinter ist, dass Phrasen, die Tatbestandsvoraussetzungen darstellen, oft mit Konjunktionen (*WennWort*) beginnen. Die Schwierigkeit besteht darin, das Ende dieser Phrasen richtig zu setzen. Wählt man beispielsweise das darauffolgende Komma, so werden etwaige Aufzählungen oder Relativsätze irrtümlicherweise als Ende der Tatbestandsvoraussetzung behandelt, obwohl der darauf-

folgende Text auch Teil der Tatbestandsvoraussetzung ist. Da keine sinnvollere Lösung gefunden wurde, ist das der Kompromiss, wie er in der finalen Implementierung angewandt wird. Weil die betroffene Regel im Bereich *-I I-* steht, hat ein falsches Erkennen des Endes der Tatbestandsvoraussetzung zusätzlich zur Folge, dass der Teil, der abgeschnitten wurde, als SONST markiert wird. Ist in dem Teil der Tatbestandsvoraussetzung zusätzlich noch eine *Negation* enthalten, so wird im Skript die *Negation* auf die gesamte Funktion der Norm bezogen, was zur Folge hat, dass eine Rechtsmacht beispielsweise als Nicht-Rechtsmacht markiert wird. Die Ergebnisse der Evaluation in Kapitel 6 unterstützen diese Vermutung.

5.3.4. Fehler mit Ruta Compiler

Während der Arbeit an den Ruta Skripten ist der Fall aufgetreten, dass zwei exakt gleiche Skripte auf dem selben Eingabetext unterschiedliche Annotationen erzeugt haben. Außerdem trat der Fehler auf, dass der Wert einer Boolean-Variable scheinbar willkürlich im Code verändert wurde. Dieser Fehler ist bei Ausführung des Sktipts Anspruchsgrundlage7-10.ruta im BLOCK-Statement zur Annotation *Anspruchsgrundlage9* zu beobachten. Das Skript ist im Anhang angefügt und enthält eine nachträgliche Behebung des Fehlers, die mit einem Kommentar gekennzeichnet wurde. Beide Fehler wurden ausgiebig getestet. Es konnte keine Fehlerquelle innerhalb der Skripte ausfindig gemacht werden, wodurch der Verdacht erhärtet, dass es einen Fehler im Ruta-Compiler von Lexia gibt. Es wurde bisher kein Versuch unternommen diesen Fehler im Programmcode von Lexia ausfindig zu machen. Allerdings muss mit Kenntnis dieser Fehler und ohne deren Behebung von weiterer ausgiebiger Arbeit mit Ruta in Lexia abgeraten werden.

6. Evaluation

Um die extrahierten Muster zur Klassifizierung von Normen auf Vollständig und Richtigkeit zu prüfen wurde eine quantitative Evaluation vorgenommen. In Abschnitt 6.1 werden die Vorgehensweise und die Kriterien der Evaluation erläutert. Im darauffolgenden Abschnitt 6.2 sind die Ergebnisse der Evaluation dargelegt bevor diese im Abschnitt 6.3 interpretiert werden.

6.1. Aufbau

Weil die in den vorherigen Kapiteln beschriebenen Skripte zur Klassifizierung von Normen stets auf dem BGB getestet und weiterentwickelt wurden, wurde für die Evaluation ein anderes Gesetzt herangezogen, um das Ergebnis nicht durch Anpassen an den Evaluationskorpus zu manipulieren. Hierfür kann man eine beliebige Menge an Normen nehmen, die nicht aus dem BGB sind. Je mehr Normen betrachtet werden und je mehr davon den hier betrachteten funktionalen Klassen zugehören, desto zuverlässiger ist das Ergebnis. Es wurde das Gesetz betreffend die Gesellschaften mit beschränkter Haftung (GmbHG) mit dem Stand vom 10. Mai 2016 ausgewählt, da es eine ausreichende Menge an Normen umfasst und jede der betrachteten funktionalen Klassen repräsentiert. Es besteht aus 88 Paragraphen und enthält insgesamt 640 Sätze. Als Satz wird dabei, wie bei der Spezifizierung der Muster zur Klassifizierung in Abschnitt 5.1 definiert, jede Zeichenfolge betrachtet, die durch einen Punkt, einen Doppelpunkt oder ein Semikolon von einem anderen Satz getrennt ist. Ausgenommen werden lediglich Punkte, die in Form der Abkürzung „Abs.“ vorkommen.

Auf eine Evaluation der vorgenommenen Markierungen der Rollen der Rechtssubjekte, der Tatbestandsvoraussetzungen und des Rechts- bzw. Anspruchsinhalts wurde aus den in den Absätzen 5.3.1 und 5.3.3 genannten Gründen abgesehen. Die Menge der irrtümlich markierten Phrasen ist so groß, das eine Evaluation der Treffgenauigkeit dort nicht nötig erscheint.

Der Domänenexperte Konrad Heßler übernahm die manuelle Klassifizierung des GmbHG. Dabei fielen 20 Sätze unter den Begriff der Anspruchsgrundlage, 3 unter den der Nicht-Anspruchsgrundlage, 38 den der Rechtspflicht, zwei den der Nicht-Rechtspflicht, 25 den der Rechtsmacht, 19 den der Nicht-Rechtsmacht und vier den der Soll-Vorschrift. Außerdem wurden die Stellen als *Sonstige Rechtsvorschrift* (SR) markiert, die zwar nicht im Sinne einer ausdrücklichen Verhaltensaufforderung an eine bestimmte Person formuliert sind, doch implizite Verhaltensaufforderungen enthalten bzw. Handlungsmöglichkeiten eröffnen. Diese Normen sollten zwar nicht als Rechtspflicht o. ä. getaggt werden, aufgrund der enthaltenen Verhaltensaufforderung und der Kenntnis davon, dass die extrahierten Muster nicht ausschließlich die richtigen Stellen markieren, ist es jedoch nicht so gravierend, wenn eine solche Norm von den Skripten fälschlicherweise getaggt wird. Im

GmbHG sind weitere 217 Normen dieser Kategorie zuzuordnen. Den Normen, die keiner funktionalen Klasse, wie sie in Kapitel 4 definiert wurden, zugehören werden im Folgenden als *Keine Funktionalität* (KF) bezeichnet.

Es entstehen fünf Kategorien zwischen denen in der Evaluation unterschieden wird.

Weak False Positives Im Folgenden werden die Normen als Weak False Positives (WFP) bezeichnet, die vom Ruta Skript einer Funktionale Klasse zugeordnet wurden, obwohl sie in die Kategorie SR gehört.

False Positives Normen, die weder einer funktionalen Klasse angehören, noch einer sonstigen Rechtsvorschrift im Sinne der WFP darstellen, doch von den Skripten als funktionale Klasse getaggt werden, sind False Positives (FP). Das Auftreten dieser Kategorie sollte minimiert werden, um das System für potenzielle Nutzer interessant zu machen. Wird eine Norm als FP anstatt als WFP markiert, so fällt sie negativer ins Gewicht.

True Positves Normen, denen genau die funktionale Klasse zugewiesen wird, die sie haben, werden als True Positives (TP) bezeichnet.

Weak True Posives Normen, die als als andere funktionale Klasse getaggt werden, als die, die sie darstellen, werden als Weak True Positives (WTP) bezeichnet. Die Annotation vieler Normen, die in diese Kategorie fallen, resultieren aus den in Absatz 5.3.1 beschriebenen Einschränkungen. Ist eine Norm ein WTP, so ist das in den meisten Fällen auf Grenzen der Implementierung zurückzuführen, denen sich die Ersteller der Muster bewusst sind. Ihr Auftreten hat einen weniger negativen Einfluss auf die Beurteilung der Klassifikation als die WFP.

False Negatives Normen, die einer funktionalen Klasse angehören aber von den Skripten nicht als eine solche erkannt werden, sind False Negatives (FN).

Es sei angemerkt, dass sich die vorgenommene Evaluation auf die semantischen Muster bezieht, die in Abschnitt 5.2 spezifiziert wurden. Die Inkompatibilität einzelner Ruta-Skripte, die in Absatz 5.3.2 beschrieben wurde, wurde durch manuelles Zusammenfügen der jeweiligen Outputs der Ruta-Skripte umgangen. Eine andere Form der Evaluation, die beispielsweise das Skript der Rechtspflicht betrachtet, ohne alle Ergebnisse der Anspruchsgrundlage zu verwenden, würde keine Schlüsse auf die Umsetzung der Klassifizierung zulassen, weil sehr viele Normen falsch und entgegen der spezifizierten Regeln eingeordnet würden.

6.2. Ergebnisse

Tabelle 6.1 zeigt die Auswertung der Evaluation mit der jeweiligen funktionalen Klasse und der Anzahl, wie oft die zu ihr gehörigen Normen als eine funktionale Klasse gataggt wurden oder nicht.

Die Diagonale in der Tabelle, in der X und Y gleich sind, enthält, wie oft eine Norm korrekt eingeordnet wurde. Mit einer perfekten Umsetzung der Klassifikation wäre kein Wert außerhalb dieser Diagonalen eingetragen abgesehen von der Summe in der letzten Spalte.

Tabelle 6.1.: Anzahl von „Norm X wird Annotation Y versehen“ im GmbHG

X \ Y	AGL	NAGL	RP	NRP	RM	NRM	SV	Nicht erkannt	Σ
AGL	8	1	2		1			8	20
NAGL						1		2	3
RP			26	2				10	38
NRP			1	1					2
RM			2		9	6		8	25
NRM				1	6	3		9	19
SV							2	2	4

Die angewendete Evaluation zeigt, dass es abseits dieser Diagonalen mit den entwickelten Skripten einige Werte gibt. Diese zeigen an, dass eine Norm falsch klassifiziert wurde. Die Häufigkeit dieser falschen Klassifizierung ist jedoch in den meisten Fällen sehr gering. Es handelt sich also lediglich um Ausnahmen, die bei dieser Art von Mustererkennung mit den verwendeten Mitteln unvermeidbar sind.

Es gibt aber auch ausschlagende Werte neben der Diagonalen. Beispielsweise wurden sechs Normen, die eine Rechtsmacht sind, als Nicht-Rechtsmacht getaggt. Das wurde bereits vor der Evaluation vermutet, weil die Zuordnung der Negationen noch zu ungenau ist. Negationen, die sich nicht direkt auf den Anspruch oder das Recht sondern auf einzelne Tatbestandsvoraussetzungen beziehen, werden teilweise falsch erkannt, was in den entwickelten Ruta-Skripten zu einer Negation der Funktionalität führt. Das selbe gilt auch anders herum. Auch sechs Normen der Nicht-Rechtspflicht wurden als Rechtspflicht getaggt. Der Grund ist ähnlich nur anders herum. Die Negation der Funktionalität wird irrtümlicherweise einer Tatbestandsvoraussetzung zugeordnet. So kommt es vor, dass Normen dem positivem Äquivalent ihrer eigentlichen negativ formulierten Funktionalität zugeordnet werden.

Bisher wurden die Normen betrachtet, die einer funktionalen Klasse angehören, und wie und ob diese Funktionalität erkannt wurde. Nun werden die Normen betrachtet, die keine Funktionalität im Rahmen der festgelegten Klassifikation haben und dennoch markiert wurden.

Tabelle 6.2.: Anzahl von „Norm X wird Annotation Y versehen“ im GmbHG

X \ Y	AGL	NAGL	RP	NRP	RM	NRM	SV
SR	15		16	1	40	8	1
KF	8	2	8		5	3	1
Σ	23	2	24	1	45	11	2

In Tabelle 6.2 wird unterschieden zwischen *Sonstigen Rechtsvorschriften* und *Keinen Funktionalität*, wie im vorherigen Abschnitt beschrieben. Dargestellt ist die Anzahl, wie häufig einer dieser beiden Arten Normen eine funktionale Klasse zugeordnet wird. Die Summe in der letzten Zeile zeigt, wie oft die jeweilige Funktionalität einer Norm zugeordnet wur-

de obwohl diese Norm weder der jeweilgen Klasse angehört, noch eine andere betrachtete Funktionalität hat. Es werden im GmbHG also über 100 Normen eine Funktionalität fälschlicherweise zugewiesen.

Das Verhältnis zwischen TP, WTP und FN und das zwischen WFP und FP zusammenfassend in Tabelle 6.3 dargestellt. Angegeben werden die jeweiligen Quote der im letzten Abschnitt definierten Begriffe. TPR steht beispielsweise für die True Positive Rate, die auf deutsch auch Trefferquote genannt wird. Sie gibt an wie hoch der Anteil der TP an allen Positives (P) ist. Folglich ist sie bei der Anspruchsgrundlage der Anteil der richtig erkannten Anspruchsgrundlagen an den Anspruchsgrundlagen des GmbHG. WTPR und FNR werden äquivalent dazu gebildet.

Die False Positive Rate (FPR) wird hier anders als gewöhnlich nicht im Bezug auf alle Normen, die keine funktionale Klasse haben, betrachtet. Stattdessen wird sie als der Anteil der FP an den Normen definiert, die keiner funktionalen Klasse angehören und dennoch als eine markiert wurden. Im gleichen Verhältnis bildet WFPR den Anteil von WFP an der Summe von WFP und FP.

Tabelle 6.3.: Relative Anteile der Evaluationskategorien nach Klasse

	TPR	WTPR	FNR	WFPR	FPR
AGL	40,0%	20,0%	40,0%	65,2%	34,8%
NAGL	0,0%	33,3%	66,7%	0,0%	100,0%
RP	68,4%	5,3%	26,3%	66,7%	33,3%
NRP	50,0%	50,0%	0,0%	100,0%	0,0%
RM	36,0%	32,0%	32,0%	88,9%	11,1%
NRM	15,8%	36,8%	47,4%	72,7%	27,3%
SV	50,0%	0,0%	50,0%	50,0%	50,0%

Es ist ersichtlich, dass manche Zellen leer bleiben. Das kommt immer dann vor, wenn sich von einer Klasse nur sehr wenige Repräsentanten im GmbHG befinden. NAGL, NRP und SV haben zwei, zwei und vier Normen im GmbHG. Folglich klafft dort der Unterschied zwischen den verschiedenen Quoten. Diese Werte sind dementsprechend kaum aussagekräftig.

Wie bereits bei Tabelle 6.1 angemerkt, ist auch in Tabelle 6.3 zu sehen, dass die FPR recht hoch ausfällt.

6.3. Interpretation

Die letzte Zeile von Tabelle 6.2 zeigt, dass im GmbHG sehr vielen falschen Normen eine funktionale Klasse zugeordnet wird. Beispielsweise werden insgesamt 45 Normen als Rechtsmacht getaggt, die keiner funktionalen Klasse angehören. Wichtig ist dabei jedoch der Blick auf den Anteil der SR. Wird ein KF anstatt eines SR getaggt, ist das ein sehr viel stärkerer Hinweis darauf, dass die semantischen Muster grobe Fehler machen. Der Häufigkeit dieser groben Fehler pro Klasse steht in der zweiten Zeile von Tabelle 6.2. In Summe werden also 27 Normen, die keinerlei Verhaltensaufforderung darstellen, einer Funktionalen Klasse zugeordnet. Diese Anzahl ist jedoch klein im Vergleich zu den 81

Normen, die zumindest implizite Verhaltensaufforderungen sind. Das tatsächliche Ergebnis ist dementsprechend nicht so negativ wie man auf den ersten Blick vermuten könnte.

Insgesamt wird durch die Resultate der Evaluation auf dem GmbHG ersichtlich, dass diese Umsetzung der funktionalen Klassifikation nicht ausreicht, um in der juristischen Praxis Verwendung zu finden. Die Ergebnisse der Skripte sind weder zuverlässig zutreffend, noch decken sie alle funktionalen Normen annähernd vollständig ab. Es ist nicht anzunehmen, dass das Ergebnis bei Verwendung eines anderen Gesetzes als Evaluationskorpus abgesehen vom BGB deutlich positiver ausfallen würde. Weil aber viele zukünftige Verbesserungen bereits bei der Spezifikation der Muster erkennbar wurden, ist davon auszugehen, dass die erzielten Resultate noch stark verbessert werden können.

Ergänzend kann vermutet werden, dass in noch nicht betrachteten Gesetzen andere Formulierungen für die funktionalen Klassen vorliegen, die die bisherigen Muster erweitern und damit deren Zutrefflichkeit steigern können. Das gilt auch für das GmbHG.

7. Zusammenfassung & Ausblick

In dieser Bachelorarbeit wurde nach einer Einführung in das Natural Language Processing ein Versuch einer automatisierten funktionalen Klassifizierung von Normen in deutschen Gesetzen vorgestellt zu dem kein vergleichbares Projekt bekannt ist. Über den gesamten Bearbeitungszeitraum wurde eng mit dem Domänenexperten Konrad Heßler, Wissenschaftlicher Mitarbeiter der Juristischen Fakultät der LMU, zusammengearbeitet. Für die Klassifizierung wurden im Rahmen eines sich wiederholenden Arbeitszyklus semantische Muster aus Rechtsnormen des BGB extrahiert und daraus syntaktische Regeln spezifiziert. Die Regeln wurden weiterhin in der Text-Annotationssprache Apache Ruta auf der Data-Science Umgebung Lexia implementiert, getestet und ausgeführt. Die Resultate der so erhaltenen Skripte dienten als Grundlage für die erneute Überarbeitung der spezifizierten Regeln, womit der Arbeitszyklus von neuem begann und sich einige Male wiederholte. Während des gesamten Prozesses sind gewisse Probleme aufgetreten, die nicht gelöst werden konnten und zu Einschränkungen führten. In einer abschließenden Evaluation am GmbHG wurden unter anderen die Auswirkungen dieser Einschränkungen festgestellt. Die evaluierten Kennzahlen der finalen Klassifikation legen nahe, dass sie für die praktische Anwendung im juristischen Alltag noch ungeeignet ist.

Nichtsdestotrotz stellt diese Bachelorarbeit eine fundierte Grundlage dar für zukünftige Schritte hin zu einem System, das die funktionale Klassifikation von Rechtsnormen umsetzt. Werden bestehende technische Grenzen überwunden, so können die präsentierten Artefakte in deutlichem Ausmaß verbessert werden. Eine fortführende Arbeit könnte beispielsweise vertiefte linguistische Eigenschaften nutzen, um die grammatikalische Rolle der Wörter treffender feststellen zu können. Einigen aufgetretenen Fehlern, deren Ursprung im Ruta-Compiler von Lexia zu vermutet wird, sollte dringend nachgegangen werden. Können diese nicht beseitigt werden, ist von weiterer ausgiebiger Arbeit mit Ruta-Skripten in Lexia abzuraten.

Appendix

A. Ruta-Skripte

EinfacheHilfsregeln.ruta

```
// Die vordefinierten Regeln, die von vielen höheren Regeln
// aufgegriffen werden und nicht doppelt ausgeführt werden

DECLARE ArtPro;
// {Der/die/das/dieser/diese/dieses/jeder/jede/jedes}
W{REGEXP("Der|der")->ArtPro};
W{REGEXP("Die|die")->ArtPro};
W{REGEXP("Das|das")->ArtPro};
W{REGEXP("Dieser|dieser")->ArtPro};
W{REGEXP("Dieses|dieses")->ArtPro};
W{REGEXP("Diese|diese")->ArtPro};
W{REGEXP("Jeder|jeder")->ArtPro};
W{REGEXP("Jede|jede")->ArtPro};
W{REGEXP("Jedes|jedes")->ArtPro};

DECLARE Ander_;
// {anderer/andere/anderes}
W{REGEXP("Anderer|anderer")->Ander_};
W{REGEXP("Andere|andere")->Ander_};
W{REGEXP("Anderes|anderes")->Ander_};

DECLARE DDEE;
// {des/der/eines/einer} ... [Substantiv]
W{REGEXP("Des|des")->DDEE};
W{REGEXP("Der|der")->DDEE};
W{REGEXP("Eines|eines")->DDEE};
W{REGEXP("Einer|einer")->DDEE};

DECLARE Djenige;
//{Derjenige/diejenige/dasjenige/diejenigen}
W{REGEXP("Derjenige|derjenige")->Djenige};
W{REGEXP("Diejenige|diejenige")->Djenige};
W{REGEXP("Dasjenige|dasjenige")->Djenige};
W{REGEXP("Derjenigen|derjenigen")->Djenige};

DECLARE ESE;
// {Er/Sie/Es}
```

```
W{REGEXP("Er|er")->ESE};
W{REGEXP("Sie|sie")->ESE};
W{REGEXP("Es|es")->ESE};

DECLARE DDD;
// {Dem/der/den}
W{REGEXP("Dem|dem")->DDD};
W{REGEXP("Der|der")->DDD};
W{REGEXP("Den|den")->DDD};

DECLARE Diese_;
// {diesem/dieser/diesen}
W{REGEXP("Diesem|diesem") -> Diese_};
W{REGEXP("Dieser|dieser") -> Diese_};
W{REGEXP("Diesen|diesen") -> Diese_};

DECLARE Djenigen;
// {Demjenigen/derjenigen/denjenigen}
W{REGEXP("Demjenigen|demjenigen")->Djenigen};
W{REGEXP("Denjenigen|denjenigen")->Djenigen};
W{REGEXP("Derjenigen|derjenigen")->Djenigen};

DECLARE DDDV;
// {dem/der/den/vom }
W{IS(DDD) -> DDDV};
W{REGEXP("Vom|vom")->DDDV};

DECLARE KKDD;
// KKDD markiert kann, können, darf und dürfen
W{REGEXP("darf|dürfen")->KKDD};
W{REGEXP("kann|können")->KKDD};
(W{REGEXP("ist")} W{REGEXP("gestattet")} COMMA) {->KKDD};

DECLARE MMHH;
// MMHH markiert müssen, muss, hat und haben
W{REGEXP("hat|haben")->MMHH};
W{REGEXP("Hat|Haben")->MMHH};
W{REGEXP("Müssen|müssen")->MMHH};
W{REGEXP("Muss|muss")->MMHH};
W{REGEXP("Müßen|müßen")->MMHH};
W{REGEXP("Muß|muß")->MMHH};

DECLARE WennWort2;
// [VERB]
pos.V{->WennWort2};
```

```
DECLARE DannWort;
// {so \wie\ / dann}
W{REGEXP("So|so")->DannWort}?;
W{REGEXP("Dann|dann")->DannWort};


DECLARE GGVV;
// gegenüber, gegen, von, vom
W{REGEXP("Gegenüber|gegenüber")->GGVV};
W{REGEXP("Gegen|gegen")->GGVV};
W{REGEXP("Von|von")->GGVV};
W{REGEXP("Vom|vom")->GGVV};

DECLARE ZU;
// ZU markiert alle Vorkommnisse von "zu"
"zu" -> ZU;
```

KomplexeHilfsregeln.ruta

```
// Import types
IMPORT PACKAGE
de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos
FROM _LEXIA_TypeSystem AS pos;

// Ruft das Skript EinfacheHilfsregeln.ruta auf
SCRIPT EinfacheHilfsregeln;
CALL(EinfacheHilfsregeln);

// § wird sonst Substantiv erkannt
W{REGEXP("§")->UNMARK(pos.N)};

DECLARE Tatbestandsvoraussetzung;

// Hilfsannotation für alle Wörter, Sonderzeichen,
// Ziffernfolgen, Kommas
DECLARE WCOMMA;  und den Punkt in "Abs."
//W{REGEXP("Abs")} PERIOD{-> WCOMMA};
Token{-> WCOMMA};
WCOMMA{OR(IS(PERIOD),IS(SEMICOLON),IS(COLON))
->UNMARK(WCOMMA)};
// Ausnahme bei "Abs."
W{REGEXP("Abs")} PERIOD{-> MARK(WCOMMA)};

DECLARE AnspruchsI_BerechtigterN;

// {Der/die/das/dieser/diese/dieses/jeder/jede/
// jedes} ...
// {[Substantiv]/anderer/andere/anderes}
// ({des/der/eines/einer} ... [Substantiv])
ArtPro (WCOMMA*? pos.N (DDEE WCOMMA*? pos.N)?)
{-> AnspruchsI_BerechtigterN};
// {Der/die/das/dieser/diese/dieses/jeder/jede/
// jedes} ...
// {[Substantiv]/anderer/andere/anderes}
// ({des/der/eines/einer} ... [Substantiv])
ArtPro (WCOMMA*? Ander_ (DDEE WCOMMA*? pos.N)?)
{-> AnspruchsI_BerechtigterN};
//{Derjenige/diejenige/dasjenige/diejenigen} ...
// ([Substantiv]), ___ ,
(Djenige WCOMMA*? pos.N? COMMA W* COMMA)
{-> AnspruchsI_BerechtigterN};
// {Er/Sie/Es}
W{IS(ESE) -> AnspruchsI_BerechtigterN};
```

```
//Wer ...,
(W{REGEXP("Wer|wer") -> AnspruchsI_BerechtigterN} W*
{->Tatbestandsvoraussetzung} COMMA);

DECLARE AnspruchsI_BerechtigterD;
//{ Dem/der/den}...[Substantiv]
DDD WCOMMA*? pos.N{-> AnspruchsI_BerechtigterD};
// {diesem/dieser/diesen}
W{IS(Diese_)->AnspruchsI_BerechtigterD};
// {Demjenigen/derjenigen/denjenigen}...([Substantiv]),___,
(Djenigen WCOMMA*? pos.N COMMA W* COMMA)
{-> AnspruchsI_BerechtigterD};

DECLARE AnspruchsI_BerechtigterA,AnspruchsI_BerechtigterG;

DECLARE AnspruchsG_VerpflichteterN;
// {Der/die/das/dieser/diese/dieses/jeder/jede/jedes}...
// [Substantiv] ({des/der/eines/einer} ... [Substantiv])
ArtPro WCOMMA*? (pos.N (DDEE WCOMMA*? pos.N)?)
{->AnspruchsG_VerpflichteterN};
// {Derjenige/diejenige/dasjenige/diejenigen} ...
// ([Substantiv]),___,
(Djenige WCOMMA*? pos.N? COMMA W* COMMA)
{-> AnspruchsG_VerpflichteterN};
// {Er/Sie/es}
W{IS(ESE)->AnspruchsG_VerpflichteterN};
// Wer ...,
W{REGEXP("Wer|wer")->AnspruchsG_VerpflichteterN} W*
{->Tatbestandsvoraussetzung} COMMA;

DECLARE AnspruchsG_VerpflichteterD;
// {dem/der/den/vom } ... {[Substantiv]\Vorschriften/anderen} 1
DDDV WCOMMA*? pos.N{-REGEXP("Vorschriften")
- >AnspruchsG_VerpflichteterD};
// {dem/der/den/vom } ... {[Substantiv]\Vorschriften/anderen} 2
DDDV WCOMMA*? W{REGEXP("anderen")
 -> AnspruchsG_VerpflichteterD};
// { diesem/dieser/diesen}
W{IS(Diese_)->AnspruchsG_VerpflichteterD};
// {demjenigen/derjenigen/denjenigen} ... ([Substantiv]),____,
(Djenigen WCOMMA*? pos.N? COMMA W* COMMA)
{->AnspruchsG_VerpflichteterD};

DECLARE AnspruchsG_VerpflichteterA;

DECLARE WennWort;
```

```
// {wenn/sofern/sobald/soweit/insoweit/falls)
W{REGEXP("Wenn|wenn")->WennWort};
W{REGEXP("Sofern|sofern")->WennWort};
W{REGEXP("Sobald|sobald")->WennWort};
W{REGEXP("Soweit|soweit")->WennWort};
W{REGEXP("Insoweit|insoweit")->WennWort};
W{REGEXP("Falls|falls")->WennWort};

DECLARE AusnahmePhrase;
// es sei denn, ... [Ende des Halbsatzes]
(W{REGEXP("Es|es")} W{REGEXP("sei")} W{REGEXP("denn")}
 COMMA # PM){-> AusnahmePhrase};
// Wenn nicht ... [Ende des Halbsatzes]
(W{REGEXP("Wenn|wenn")} W{REGEXP("nicht")} # PM)
{-> AusnahmePhrase};
// Vorbehaltlich {der/des} ... {[Ende des Halbsatzes]}
(W{REGEXP("Vorbehaltlich|vorbehaltlich")}
W{REGEXP("der|des")} # PM){-> AusnahmePhrase};

// Außer {im Fall(e) des/in den Fällen des} ...
// {[Ende des Halbsatzes]}
DECLARE IFD;
// im Fall(e) des/in den Fällen des
"Im Fall des|im Fall des" -> IFD;
"Im Falle des|im Falle des" -> IFD;
"In den Fällen des|in den Fällen des" -> IFD;
(W{REGEXP("Außer|außer")} IFD # PM){-> AusnahmePhrase};

DECLARE Negation;
DECLARE Negation Negation1;
DECLARE Negation Negation2;
// {Kein/keine/nicht/nur/lediglich/ausschließlich dann}
W{REGEXP("Kein|kein")->Negation1};
W{REGEXP("Keine|keine")->Negation1};
W{REGEXP("Nicht|nicht")->Negation1};
W{REGEXP("Weder|weder")->Negation1};

W{REGEXP("Nur|nur")->Negation2};
W{REGEXP("Lediglich|lediglich")->Negation2};
W{REGEXP("Ausschließlich|ausschließlich")->Negation2};

// Annotationen für die Erkennung von
// [Wenn-Wort] ... [Ende des Halbsatzes]
DECLARE Wenn_;
WennWort W*? COMMA{-PARTOF(Wenn_) -> MARK(Wenn_,1,2)};
// Hilfsannotation zum Erkennen der einzelnen Token
```

```
// von Wenn_
DECLARE Wenn_Token;
W{PARTOF(Wenn_)->Wenn_Token};
```

Anspruchsgrundlage7bis10.ruta

```
// Import types
IMPORT PACKAGE
de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos
FROM _LEXIA_TypeSystem AS pos;

// Führt das Skript KomplexeHilfsregeln.ruta aus für
// die Annotationen von KomplexeHilfsregeln und
// EinfacheHilfsregeln
SCRIPT KomplexeHilfsregeln;
CALL(KomplexeHilfsregeln);

DECLARE Anspruchsinhaber;
DECLARE Anspruchsinhalt;
DECLARE Anspruchsgegner;
DECLARE NichtAnspruchsinhaber;
DECLARE NichtAnspruchsinhalt;
DECLARE NichtAnspruchsgegner;

DECLARE Anspruchsgrundlage;
DECLARE NichtAnspruchsgrundlage;

DECLARE Anspruchsgrundlage Anspruchsgrundlage7;
DECLARE Anspruchsgrundlage Anspruchsgrundlage8;
DECLARE Anspruchsgrundlage Anspruchsgrundlage9;
DECLARE Anspruchsgrundlage Anspruchsgrundlage10;
DECLARE NichtAnspruchsgrundlage NichtAnspruchsgrundlage7;
DECLARE NichtAnspruchsgrundlage NichtAnspruchsgrundlage8;
DECLARE NichtAnspruchsgrundlage NichtAnspruchsgrundlage9;
DECLARE NichtAnspruchsgrundlage NichtAnspruchsgrundlage10;

DECLARE AnspruchsgrundlageEnde7;
DECLARE AnspruchsgrundlageEnde8;
DECLARE AnspruchsgrundlageEnde9;
DECLARE AnspruchsgrundlageEnde10;
DECLARE NichtAnspruchsgrundlageEnde7;
DECLARE NichtAnspruchsgrundlageEnde8;
DECLARE NichtAnspruchsgrundlageEnde9;
DECLARE NichtAnspruchsgrundlageEnde10;

DECLARE Sonst;
BOOLEAN ContainsNegationInSonst;

// Hilfsannotationen zum Erkennen der einzelnen
Token der Annotationen
```

```
DECLARE AnspruchsI_BerechtigterAToken;
DECLARE AnspruchsI_BerechtigterDToken;
DECLARE AnspruchsI_BerechtigterGToken;
DECLARE AnspruchsI_BerechtigterNToken;
DECLARE AnspruchsG_VerpflichteterNToken;
DECLARE AnspruchsG_VerpflichteterAToken;
DECLARE GegAnspruchsI_BerechtigterD,
 GegAnspruchsI_BerechtigterDToken;
DECLARE GegADToken;
DECLARE AusnahmePhraseToken;
DECLARE Verpflichtet;

// -Hilfsannotationen zum Erkennen von Optionalen Phrasen -

// Annotationen für die Erkennung von AnspruchsI_BerechtigterA
W{PARTOF(AnspruchsI_BerechtigterA)
->AnspruchsI_BerechtigterAToken};

// Annotationen für die Erkennung von AnspruchsI_BerechtigterD
W{PARTOF(AnspruchsI_BerechtigterD)
->AnspruchsI_BerechtigterDToken};

// Annotationen für die Erkennung von AnspruchsI_BerechtigterG
W{PARTOF(AnspruchsI_BerechtigterG)
->AnspruchsI_BerechtigterGToken};

// Annotationen für die Erkennung von AnspruchsG_VerpflichteterN
W{PARTOF(AnspruchsG_VerpflichteterN)
->AnspruchsG_VerpflichteterNToken};

// Annotationen für die Erkennung von AnspruchsG_VerpflichteterA
W{PARTOF(AnspruchsG_VerpflichteterA)
->AnspruchsG_VerpflichteterAToken};

// Annotationen für die Erkennung von "gegenüber"
// GegAnspruchsI_BerechtigterD
(W{REGEXP("gegenüber")} AnspruchsI_BerechtigterD)
{ -> GegAnspruchsI_BerechtigterD};
W{PARTOF(GegAnspruchsI_BerechtigterD)
->GegAnspruchsI_BerechtigterDToken};

// Annotationen für die Erkennung von [Ausnahme-Phrase]
W{PARTOF(AusnahmePhrase)->AusnahmePhraseToken};

// Annotationen für die Erkennung von Verpflichtet
W{REGEXP("verpflichtet") -> Verpflichtet};
```

```
//--Regeln zum Erkennen der Anspruchsgrundlage--

// 8
// {[Wenn-Wort]/[Wenn-Wort2]} ..., ([Dann-Wort]) {ist/sind} ...
// -| Verpflichtet / [Anspruchsgegner N] /
// {[Anspruchsinhaber G] / [Anspruchsinhaber D] /
// [Anspruchs-inhaber A]} / (SONST)|-
(WCOMMA* @W{OR(IS(WennWort), IS(WennWort2))}
#{AND(-CONTAINS(COLON),-CONTAINS(SEMICOLON),-CONTAINS(PERIOD))}
@COMMA DannWort? W{REGEXP("ist|sind")} WCOMMA*)
{AND(-PARTOF(Anspruchsgrundlage8),-CONTAINS(Anspruchsgrundlage8))
->Anspruchsgrundlage8};

BLOCK(ForEach) Anspruchsgrundlage8{} {
W{OR(IS(WennWort), IS(WennWort2))}
#{AND(-CONTAINS(COLON),-CONTAINS(SEMICOLON),-CONTAINS(PERIOD))}
@COMMA DannWort? W{REGEXP("ist|sind")}
WCOMMA*{-PARTOF(AnspruchsgrundlageEnde8)
->MARK(AnspruchsgrundlageEnde8)};

AnspruchsgrundlageEnde8{OR(-CONTAINS(Verpflichtet),
-CONTAINS(AnspruchsG_VerpflichteterN),
AND(-CONTAINS(AnspruchsI_BerechtigterG),
-CONTAINS(AnspruchsI_BerechtigterD),
-CONTAINS(AnspruchsI_BerechtigterA)))
->UNMARK(AnspruchsgrundlageEnde8)};

BLOCK(ForEach) AnspruchsgrundlageEnde8{} {
WCOMMA*{AND(-CONTAINS(Verpflichtet),
-CONTAINS(AnspruchsG_VerpflichteterNToken),
-CONTAINS(AnspruchsI_BerechtigterGToken),
-CONTAINS(Wenn_Token),
-CONTAINS(GegAnspruchsI_BerechtigterDToken),
-CONTAINS(AnspruchsI_BerechtigterAToken),
-CONTAINS(AusnahmePhraseToken), -PARTOFNEQ(Sonst))
->MARK(Sonst)};

ASSIGN(ContainsNegationInSonst, false);
Sonst{CONTAINS(Negation)->
ASSIGN(ContainsNegationInSonst,true)};

Sonst{CONTAINS(W),ContainsNegationInSonst->
NichtAnspruchsinhalt};
Sonst{CONTAINS(W),-ContainsNegationInSonst->
```

```
Anspruchsinhalt};
Wenn_{-> Tatbestandsvoraussetzung};
AusnahmePhrase{-> Tatbestandsvoraussetzung};
}
// Fügt die Tatbestandsvoraussetzung hinzu
(W{OR(IS(WennWort), IS(WennWort2))} W*)
{PARTOF(Anspruchsgrundlage8)
-> Tatbestandsvoraussetzung} COMMA;

AnspruchsgrundlageEnde8{ContainsNegationInSonst
->UNMARK(AnspruchsgrundlageEnde8),
MARK(NichtAnspruchsgrundlageEnde8)};
}
Anspruchsgrundlage8{CONTAINS(NichtAnspruchsgrundlageEnde8)
 -> MARK(NichtAnspruchsgrundlage8)};
Anspruchsgrundlage8{-CONTAINS(AnspruchsgrundlageEnde8)
 -> UNMARK(Anspruchsgrundlage8)};

// 7
//[Anspruchsgegner N] {ist/sind} ... -|Verpflichtet/
// {[Anspruchsinhaber D]/[Anspruchsinhaber A]}/(SONST)|-
(WCOMMA* @AnspruchsG_VerpflichteterN
W{REGEXP("ist|sind")} WCOMMA*)
{AND(-PARTOF(Anspruchsgrundlage7),
-IS(Anspruchsgrundlage8),-IS(NichtAnspruchsgrundlage8))
->Anspruchsgrundlage7};

BLOCK(ForEach) Anspruchsgrundlage7{} {

AnspruchsG_VerpflichteterN W{REGEXP("ist|sind")} WCOMMA*
{-PARTOF(AnspruchsgrundlageEnde7)
->MARK(AnspruchsgrundlageEnde7)};

AnspruchsgrundlageEnde7
{OR(AND(-CONTAINS(AnspruchsI_BerechtigterD),
-CONTAINS(AnspruchsI_BerechtigterA)),-CONTAINS(Verpflichtet))
->UNMARK(AnspruchsgrundlageEnde7)};
BLOCK(ForEach) AnspruchsgrundlageEnde7{} {

WCOMMA*{AND(-CONTAINS(Verpflichtet),
-CONTAINS(AnspruchsI_BerechtigterAToken),
-CONTAINS(AnspruchsI_BerechtigterDToken),
-CONTAINS(Wenn_Token),-CONTAINS(AusnahmePhraseToken),
 -PARTOFNEQ(Sonst))->MARK(Sonst)};

ASSIGN(ContainsNegationInSonst, false);
```

```
Sonst{CONTAINS(Negation)
-> ASSIGN(ContainsNegationInSonst, true)};
Sonst{CONTAINS(W),ContainsNegationInSonst
-> NichtAnspruchsinhalt};
Sonst{CONTAINS(W),-ContainsNegationInSonst ->Anspruchsinhalt};
Wenn_{-> Tatbestandsvoraussetzung};
AusnahmePhrase{-> Tatbestandsvoraussetzung};
}
AnspruchsgrundlageEnde7{ContainsNegationInSonst
->UNMARK(AnspruchsgrundlageEnde7),
MARK(NichtAnspruchsgrundlageEnde7)};
}
Anspruchsgrundlage7{CONTAINS(NichtAnspruchsgrundlageEnde7)
-> MARK(NichtAnspruchsgrundlage7)};
Anspruchsgrundlage7{-CONTAINS(AnspruchsgrundlageEnde7)
->UNMARK(Anspruchsgrundlage7)};


// 10
// {[Wenn-Wort]/[Wenn-Wort2]} ..., ([Dann-Wort]) {hat/haben}
// ... [Anspruchsgegner N] ... {[Anspruchsinhaber G]/
// [Anspruchsinhaber D] / [Anspruchsinhaber A]}...
(WCOMMA* W{OR(IS(WennWort), IS(WennWort2))}
#{AND(-CONTAINS(COLON),-CONTAINS(SEMICOLON),-CONTAINS(PERIOD))}
@COMMA DannWort?
W{REGEXP("hat|haben")} WCOMMA*?
AnspruchsG_VerpflichteterN WCOMMA*?
ANY{OR(IS(AnspruchsI_BerechtigterD), IS(AnspruchsI_BerechtigterA),
IS(AnspruchsI_BerechtigterG))} WCOMMA*
{AND(-PARTOF(AnspruchsgrundlageEnde10))
->MARK(AnspruchsgrundlageEnde10)})
{AND(-PARTOF(Anspruchsgrundlage10),
-CONTAINS(Anspruchsgrundlage10))->Anspruchsgrundlage10};

BLOCK(ForEach) Anspruchsgrundlage10{} {
BLOCK(ForEach) AnspruchsgrundlageEnde10{} {

WCOMMA*{AND(-CONTAINS(AnspruchsG_VerpflichteterNToken),
-CONTAINS(AnspruchsI_BerechtigterGToken),
-CONTAINS(Wenn_Token),-CONTAINS(GegAnspruchsI_BerechtigterDToken),
-CONTAINS(AnspruchsI_BerechtigterAToken),
-CONTAINS(AusnahmePhraseToken), -PARTOFNEQ(Sonst))->MARK(Sonst)};

ASSIGN(ContainsNegationInSonst, false);
Sonst{CONTAINS(Negation)-> ASSIGN(ContainsNegationInSonst, true)};
Sonst{CONTAINS(W),ContainsNegationInSonst -> NichtAnspruchsinhalt};
```

```
Sonst{CONTAINS(W),-ContainsNegationInSonst ->Anspruchsinhalt};
Wenn_{-> Tatbestandsvoraussetzung};
AusnahmePhrase{-> Tatbestandsvoraussetzung};
}
AnspruchsgrundlageEnde10{ContainsNegationInSonst
->UNMARK(AnspruchsgrundlageEnde10),
MARK(NichtAnspruchsgrundlageEnde10)};

// Fügt die Tatbestandsvoraussetzung dazu
(W{OR(IS(WennWort), IS(WennWort2))} W*)
{PARTOF(Anspruchsgrundlage10) -> Tatbestandsvoraussetzung} COMMA;
}
Anspruchsgrundlage10{CONTAINS(NichtAnspruchsgrundlageEnde10)
-> MARK(NichtAnspruchsgrundlage10)};
Anspruchsgrundlage10{-CONTAINS(AnspruchsgrundlageEnde10)
->UNMARK(Anspruchsgrundlage10)};

// 9
// [Anspruchsgegner N] ... {hat/haben/muss/müssen} ...
// {[Anspruchsinhaber D] / [Anspruchsinhaber A] /
// [Anspruchsinhaber G]} ...
(WCOMMA* @AnspruchsG_VerpflichteterN WCOMMA*? MMHH WCOMMA*)
{AND(-CONTAINS(Anspruchsgrundlage9),
-IS(Anspruchsgrundlage10),-IS(NichtAnspruchsgrundlage10))
->Anspruchsgrundlage9};

BLOCK(ForEach) Anspruchsgrundlage9{} {
AnspruchsG_VerpflichteterN WCOMMA*? MMHH WCOMMA*
{->AnspruchsgrundlageEnde9};

AnspruchsG_VerpflichteterN WCOMMA*?
{->MARK(Tatbestandsvoraussetzung)} MMHH;
AnspruchsgrundlageEnde9{AND(-CONTAINS(AnspruchsI_BerechtigterA),
-CONTAINS(AnspruchsI_BerechtigterG),
-CONTAINS(AnspruchsI_BerechtigterD))
->UNMARK(AnspruchsgrundlageEnde9)};

BLOCK(ForEach) AnspruchsgrundlageEnde9{} {
WCOMMA*{AND(-CONTAINS(AnspruchsI_BerechtigterGToken),
-CONTAINS(Wenn_Token),-CONTAINS(GegAnspruchsI_BerechtigterDToken),
-CONTAINS(AnspruchsI_BerechtigterDToken),
-CONTAINS(AnspruchsI_BerechtigterAToken),
-CONTAINS(AusnahmePhraseToken), -PARTOFNEQ(Sonst))->MARK(Sonst)};

ASSIGN(ContainsNegationInSonst, false);
```

```
// Ist aus nicht erkennbaren Gründen Fehlerhaft
Sonst{AND(CONTAINS(Negation))
->ASSIGN(ContainsNegationInSonst, true)};

Sonst{AND(CONTAINS(W),ContainsNegationInSonst)
-> NichtAnspruchsinhalt};
Sonst{AND(CONTAINS(W),-ContainsNegationInSonst)
->Anspruchsinhalt};

Wenn_{-> Tatbestandsvoraussetzung};
AusnahmePhrase{-> Tatbestandsvoraussetzung};
}
AnspruchsgrundlageEnde9{ContainsNegationInSonst->
UNMARK(AnspruchsgrundlageEnde9),
MARK(NichtAnspruchsgrundlageEnde9)};
}
Anspruchsgrundlage9{CONTAINS(NichtAnspruchsgrundlageEnde9)
-> MARK(NichtAnspruchsgrundlage9)};
Anspruchsgrundlage9{-CONTAINS(AnspruchsgrundlageEnde9)
->UNMARK(Anspruchsgrundlage9)};
// Behandlung des Fehlers mit ContainsNegationInSonst:
NichtAnspruchsgrundlage9{-CONTAINS(Negation)
->UNMARK(NichtAnspruchsgrundlage9),MARK(Anspruchsgrundlage9)};
NichtAnspruchsinhalt{PARTOF(Anspruchsgrundlage9)
->MARK(Anspruchsinhalt),UNMARK(NichtAnspruchsinhalt)};

// -- Annotationen anpassen --

// Hinzufügen der Tatbestandsvoraussetzung
// durch AnspruchsI_BerechtigterN
(AnspruchsI_BerechtigterN{REGEXP("Wer|wer")}
W*{->Tatbestandsvoraussetzung} COMMA)
{PARTOF(Anspruchsgrundlage)};

//Entfernen sinnloser Tatbestandsvoraussetzungen
Tatbestandsvoraussetzung{AND(-PARTOF(Anspruchsgrundlage),
-PARTOF(NichtAnspruchsgrundlage))
->UNMARK(Tatbestandsvoraussetzung)};
Tatbestandsvoraussetzung{PARTOFNEQ(Tatbestandsvoraussetzung)
->UNMARK(Tatbestandsvoraussetzung)};
pos.V{IS(Tatbestandsvoraussetzung)
-> UNMARK(Tatbestandsvoraussetzung)};

// Entfernen von einzelnen Artikeln aus Anspruchsinhalt
pos.ART{IS(Anspruchsinhalt)->UNMARK(Anspruchsinhalt)};
pos.ART{IS(NichtAnspruchsinhalt)->UNMARK(NichtAnspruchsinhalt)};
```

```
Token{AND(IS(Anspruchsinhalt),-IS(pos.N))
->UNMARK(Anspruchsinhalt)};
Token{AND(IS(NichtAnspruchsinhalt),-IS(pos.N))
->UNMARK(NichtAnspruchsinhalt)};

// --- Entfernen von Hilfsannotationen ---

Anspruchsgrundlage10{->UNMARK(Anspruchsgrundlage10),
MARK(Anspruchsgrundlage)};
Anspruchsgrundlage9{->UNMARK(Anspruchsgrundlage9),
MARK(Anspruchsgrundlage)};
Anspruchsgrundlage8{->UNMARK(Anspruchsgrundlage8),
MARK(Anspruchsgrundlage)};
Anspruchsgrundlage7{->UNMARK(Anspruchsgrundlage7),
MARK(Anspruchsgrundlage)};
NichtAnspruchsgrundlage10{->UNMARK(NichtAnspruchsgrundlage10),
MARK(NichtAnspruchsgrundlage)};
NichtAnspruchsgrundlage9{->UNMARK(NichtAnspruchsgrundlage9),
MARK(NichtAnspruchsgrundlage)};
NichtAnspruchsgrundlage8{->UNMARK(NichtAnspruchsgrundlage8),
MARK(NichtAnspruchsgrundlage)};
NichtAnspruchsgrundlage7{->UNMARK(NichtAnspruchsgrundlage7),
MARK(NichtAnspruchsgrundlage)};

AnspruchsgrundlageEnde7 {-> UNMARK(AnspruchsgrundlageEnde7)};
AnspruchsgrundlageEnde8 {-> UNMARK(AnspruchsgrundlageEnde8)};
AnspruchsgrundlageEnde9 {-> UNMARK(AnspruchsgrundlageEnde9)};
AnspruchsgrundlageEnde10 {-> UNMARK(AnspruchsgrundlageEnde10)};
NichtAnspruchsgrundlageEnde7
{-> UNMARK(NichtAnspruchsgrundlageEnde7)};
NichtAnspruchsgrundlageEnde8
{-> UNMARK(NichtAnspruchsgrundlageEnde8)};
NichtAnspruchsgrundlageEnde9
{-> UNMARK(NichtAnspruchsgrundlageEnde9)};
NichtAnspruchsgrundlageEnde10
{-> UNMARK(NichtAnspruchsgrundlageEnde10)};

AnspruchsG_VerpflichteterN {PARTOF(Anspruchsgrundlage)
-> MARK(Anspruchsgegner)};
AnspruchsG_VerpflichteterD {PARTOF(Anspruchsgrundlage
)-> MARK(Anspruchsgegner)};
AnspruchsI_BerechtigterD {PARTOF(Anspruchsgrundlage)
-> MARK(Anspruchsinhaber)};
AnspruchsI_BerechtigterN {PARTOF(Anspruchsgrundlage)
-> MARK(Anspruchsinhaber)};
AnspruchsI_BerechtigterA {PARTOF(Anspruchsgrundlage)
```

```
-> MARK(Anspruchsinhaber)};
AnspruchsG_VerpflichteterN {PARTOF(NichtAnspruchsgrundlage)
-> MARK(NichtAnspruchsgegner)};
AnspruchsG_VerpflichteterD {PARTOF(NichtAnspruchsgrundlage)
-> MARK(NichtAnspruchsgegner)};
AnspruchsI_BerechtigterD {PARTOF(NichtAnspruchsgrundlage)
-> MARK(NichtAnspruchsinhaber)};
AnspruchsI_BerechtigterN {PARTOF(NichtAnspruchsgrundlage)
-> MARK(NichtAnspruchsinhaber)};
AnspruchsI_BerechtigterA {PARTOF(NichtAnspruchsgrundlage)
-> MARK(NichtAnspruchsinhaber)};

-- Entfernen von inneren Annotationen --
Anspruchsinhaber{PARTOFNEQ(Anspruchsinhaber)
->UNMARK(Anspruchsinhaber)};
NichtAnspruchsinhaber{PARTOFNEQ(NichtAnspruchsinhaber)
->UNMARK(NichtAnspruchsinhaber)};
Anspruchsgegner{PARTOFNEQ(Anspruchsgegner)
->UNMARK(Anspruchsgegner)};
NichtAnspruchsgegner{PARTOFNEQ(NichtAnspruchsgegner)
->UNMARK(NichtAnspruchsgegner)};
```

Literaturverzeichnis

[1] Steven P. Abney. *Parsing By Chunks*. Bell Communications Research, November 1994.

[2] Michael Collins. Probabilistic context-free grammars (pcfgs). University Lecture, http://www.cs.columbia.edu/~mcollins/courses/nlp2011/notes/pcfgs.pdf, 2013.

[3] Guenther Goerz and Guenter Schellenberger. Chunk parsing in corpora. Technical report, University of Erlangen-Nuremberg, Chair of Computer Science 8 (AI), October 2007.

[4] Jurawiki. Rechtssubjekt, 2016. [Online; accessed 8-August-2016].

[5] Peter Kluegel, Martin Toepfer, Philipp-Daniel Beck, Georg Fette, and Frank Puppe. UIMA Ruta: Rapid development of rule-based information extraction applications. Technical report, University of Würzburg, Department Computer Science VI, Department of Heart Failure Center, October 2014.

[6] Sandra Kuebler, Ryan McDonald, and Joakim Nivre. *Dependency Parsing*. Morgan & Claypool Publishers, 2009.

[7] Mitchell P. Marcus Lance A. Ramshaw. *Text Chunking using Transformation-Based Learning*. Bell Communications Research, 1995.

[8] Daniel Jurafsky & James H. Martin. Semantic role labeling. 2015.

[9] Referat Parlamentsdokumentation. Statistik der gesetzgebung – Überblick 18. wahlperiode. Technical report, Deutscher Bundestag, 2016. [Online; accessed 8-August-2016].

[10] Martin Porter. German stemming algorithm. Website. http://snowball.tartarus.org/algorithms/german/stemmer.html, [Online; accessed 6-October-2016].

[11] Martin Porter. The porter stemming algorithm. Website. https://tartarus.org/martin/PorterStemmer/, [Online; accessed 6-October-2016].

[12] Anne Schille, Simone Teufel, Christine Stöckert, and Christine Thielen. Guidellnes für das tagging deutscher textcorpora mit stts (kleines und großes tagset). 1999.

[13] John Wilder Tukey. *Exploratory Data Analysis*. Addison-Wesley Publishing Company, 1977.

[14] Wikipedia. Morph — wikipedia, die freie enzyklopädie. Website, 2015. https://de.wikipedia.org/w/index.php?title=Morph&oldid=149063476, [Online; Stand 16. Oktober 2016].

[15] Wikipedia. Named-entity recognition — wikipedia, the free encyclopedia, 2016. [Online; accessed 3-October-2016].

[16] Wikipedia. Natural language processing — wikipedia, the free encyclopedia. Website, 2016. https://en.wikipedia.org/w/index.php?title=Natural_language_processing&oldid=742902283, [Online; accessed 6-October-2016].

[17] Wikipedia. Sentence boundary disambiguation — wikipedia, the free encyclopedia, 2016. [Online; accessed 8-August-2016].

[18] Marco Zierl. Entwicklung und Implementierung eines Datenbanksystems zur Speicherung und Verarbeitung von Textkorpora, 1997.

[19] Heike Zinsmeie, Ulrich Heid, and Kathrin Beck. Journal for language technology and computational linguistics, volume 28. 2013.