Korpus-basierte effiziente Informationsextraktion und Grammatikinduktion der natürlichen Sprachen

Inaugural-Dissertation

Zur

Erlangen des Doktorgrades der Philosophie des Fachbereiches Sprache, Literatur und Kultur der Justus-Liebig-Universität Gießen

Vorgelegt von

Chunze Shen

Aus Jiangsu, VR China

01.2013

1. Berichterstatter: Prof. Dr. Henning Lobin

2. Berichterstatter: Prof. Dr. Thomas Gloning

Danksagung

Das Entstehen dieser Arbeit haben viele Personen auf unterschiedliche Weise beitragen. An dieser Stelle möchte ich mich bei allen bedanken, die mich in dieser Zeit begleitet haben. Mein Dank gilt besonders meinem Betreuer, Herrn Prof. Dr. Henning Lobin, der mich mit wertvollen fachlichen und menschlichen Hinweisen unterstützt hat. Mein Dank geht auch an den Zweitgutachter, Herrn Prof. Dr. Thomas Gloning, der meine Arbeit mit anregenden Anmerkungen kommentiert hat.

Des Weiteren bedanke ich mich bei Herrn Dr. Marc Kupietz und Frau Kathrin Beck, die mich die Nutzung der Testkorpora ermöglicht haben. Ganz besonders danke ich Anna-lena und Holger, die meine Arbeit sorgfältig korrigiert haben.

Nicht zuletzt bedanke ich mich bei meinen Eltern, die mich mit viel Geduld unterstützt und stets ermutigt haben.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Abkürzung

ABL	Alignment-Based Learning
AGI	Automatische Grammatik-Induktion
CGI	Computationale Grammatik-Induktion
EDA	Edit-Distance Algorithmus
EDSI	Effiziente Destillation Syntaktischer Informationen
GI	Grammatik-Induktion
GST	Gold-Standard Treebank
LG	Lokale Grammatiken
LGtm	Looks-Good-to-me
MLE	Minimum Length Encoding
MUC	Message Understanding Conference
NER	Named Entity Recognition
PAC	Probably Approximately Correct
PNF	Proper Name Facility
RKG	Rebuilding Known Grammars
RTN	Recursive Transition Networks
SCFG	Stochastic Context-Free Grammar
SPM	Syntagmatic-Paradigmatic Model
UG	Universale Grammatik

1. Einleitung

Grammatik-Induktion[1] (GI) gewinnt aufgrund ihrer vielseitigen Anwendungen seit einigen Jahren immer mehr Aufmerksamkeit (Brill 1993; Stolcke/Omobundro 1994; Adriaans/van Zaanen 2004 etc.). Traditionell wird GI in linguistischen Bereichen erforscht und oft für die Erkennung syntaktischer Patterns eingesetzt. Außerdem gibt es viele interdisziplinäre Anwendungen, wie z.B. die Verarbeitung genetischer Sequenzen (Gavrilis, Tsoulos & Dermatas 2006), automatische Musik-Komposition (Cruz-Alcazar & Vidal-Ruiz 1997/1998) usw.

In der Computerlinguistik ist die automatische Grammatik-Induktion (AGI) heute ein hochinteressantes Forschungsthema geworden. In der vorliegenden Arbeit wurde versucht, kontextfreie Grammatiken automatisch zu extrahieren und darüber hinaus lokale Grammatiken zu induzieren. Wie der Name schon verrät, fokussieren sich lokale Grammatiken (LG) viel mehr auf lokale syntaktische Phänomene, die in unmittelbarer Nachbarschaft eines Wortes oder einer Wortklasse vorkommen.

Lokale Grammatiken sind eigentlich endliche Automaten bzw. Transduktoren und werden in der Regel als Graphen dargestellt. Die Anwendungsgebiete lokaler Grammatik umfassen Informationsextraktion (Senellart 1998), automatische Übersetzung (Senellart/Dienes/Váradi 2001), lexikalische Disambiguierung (Nagel 2008, 8) etc.

Mit der Unterstützung der Computertechnik bilden sich in der GI neue Methoden heraus. Aber trotz der rasanten Entwicklung von Hardware stoßen heutige Ansätze oft an das Problem von Aufwand-Engpässen – besonders bei der Verarbeitung von großen Korpora. Daher ergibt sich das Bedürfnis nach einem effizienten Algorithmus. In der vorliegenden Arbeit wird ein neuer Korpus-basierter Ansatz vorgestellt, um lokale Grammatiken effizient zu extrahieren. Diese Arbeit versteht sich als ein Beitrag zur automatischen Grammatik-Induktion.

---

[1] Grammatik-Induktion, Grammatik-Inferenz und Erkennung syntaktischer Strukturen werden in der vorliegenden Arbeit als Synonyme betrachtet.

1.1 Gegenstand der vorliegenden Arbeit

Der Untersuchungsgegenstand der vorliegenden Arbeit besteht aus den Korpora der natürlichen Sprachen[2] und den daraus destillierten lokalen Grammatiken. Außerdem werden distributionale Informationen, die den Korpora entnommen und als syntaktische Exemplare für die Destillation lokaler Grammatiken benutzt werden, untersucht. So wurde in dieser Arbeit ein lernbasiertes System entwickelt, welches die Grammatikregeln ohne Überwachung aus Rohdaten extrahiert.

Weil die Wortstellung für die Extraktion syntaktischer Informationen eine wichtige Rolle spielt, kann ein Ansatz von der jeweiligen Sprache abhängig sein. D.h. wenn ein Ansatz für eine Sprache erfolgreich ist, muss er nicht zwangsläufig für andere Sprachen geeignet sein. Um die Ergebnisse zwischen den Sprachen zu vergleichen, wird die Untersuchung in zwei Sprachen durchgeführt. Für jede Sprache steht jeweils ein Korpus zur Verfügung. Die Extraktion syntaktischer Informationen kann auf unterschiedlichen sprachlichen Ebenen (Wort-, Phrasen- & Satzebene) durchgeführt werden. Für die Zwecke der vorliegenden Arbeit sind Sätze die größte Untersuchungseinheit.

Die aus den Korpora induzierten lokalen Grammatiken repräsentieren lokale syntaktische Informationen im unmittelbaren Kontext innerhalb eines Satzes. Dies ist ein gravierender Unterschied zur traditionellen Phrasenstruktur-Grammatik, „*die die Syntax einer Sprache allein auf der Basis einiger weniger Wortklassen wie Determinativ, Adjektiv, Nomen, Verb, Präposition und anhand einiger Typen von Phrasen (NP, PP, VP) zu beschreiben versucht*“ (Nagel 2008). Lokale Grammatiken werden normalerweise mit einem nützlichen Tool Unitex graphisch dargestellt und als Automaten oder Transduktor (Siehe Kapitel 2.2) implementiert.

1.2 Forschungslage und Motivation

Die Idee der Grammatik-Induktion stammt aus der Arbeit von Gold (1967). Mittlerweile hat sie sich durch zahlreiche Publikationen und Studien als eigenständige Disziplin etabliert (Fu

[2] In dieser Arbeit werden die Phänomene im Englischen und Deutschen untersucht.

1974; Bunke/Sanfeliu 1990; Brill 1993; Chen 1995; Hwa 2000 etc.). Für jeden GI-Ansatz sind folgende drei Fragestellungen zentral zu berücksichtigen:

- Welche syntaktischen Informationen sind nützlich und müssen aufgefunden werden?
- Wie können die Grammatikregeln gesammelt werden?
- Wie sollen die Grammatikregeln repräsentiert werden?

Wenn die Extraktion syntaktischer Informationen auf verschiedenen linguistischen Ebenen (z.B. Wort; Phrase/Satz; Diskurs oder Korpus etc.) durchgeführt wird, entstehen unterschiedliche Informationen. Die Auswahl nützlicher Informationen ist vom Ziel abhängig und ist im Hinblick auf die gegebenen spezifischen Aufgaben zu entscheiden. Für die vorliegende Untersuchung spielen syntaktische Eigenschaften des Satzes und morphosyntaktische Eigenschaften der Wortklassen die Hauptrolle, weshalb die entsprechenden Informationen aus Wort- und Phrase/Satz-Ebenen extrahiert werden müssen. Außerdem sind gewisse Informationen auf Korpus-Ebene (z.B. statistische Informationen) auch notwendig um die Ergebnisse zu optimieren.

Die zweite Frage bezieht sich auf die Grammatik-Induktion. Grundsätzlich gibt es zwei Strategien, um die Grammatikregeln zu induzieren (siehe Kapitel 2.1):

- Die klassische generative Strategie wird auch regelbasierte Strategie genannt. Dabei werden rationalistische Ansätze, die auf Musterregeln basieren, entwickelt. Die Musterregeln werden manuell explizit (symbolisch) repräsentiert. Diese Strategie dominierte den Forschungsbereich zwischen den 60er und 90er Jahren seit der Veröffentlichung der generativen Transformationsgrammatik (Chomsky 1965). Regelbasierte Systeme können generell eine bessere Qualität der Ergebnisse erzielen.
- Die statistische oder distributionale Strategie ermöglicht den empirischen Blickwinkel. Empirische Ansätze versuchen die Grammatikregeln aus einem annotierten Korpus abzuleiten. Seit Mitte/Ende der 90er Jahre sind statische Systeme das

vorherrschende Paradigma geworden (Nagel 2006, 182). Der Vorteil dieser Strategie ist, dass lernbasierte Systeme meistens leicht trainierbar und erweiterbar sind.

Die regelbasierte Strategie ist wegen des hohen Aufwands oft der Kritik von Empiristen ausgesetzt, weil alle Grammatikregeln manuell (meistens von Experten) erstellt werden müssen. Sowohl der Aufbau als auch die Erweiterung der Systeme kostet sehr viel Zeit. Außerdem sind symbolisierte Regeln schwer zu trainieren, d.h. die Systeme sind nicht in der Lage, neue Grammatiken zu lernen.

Statistische lernbasierte Systeme besitzen die Fähigkeit, die Grammatikregeln aus den Korpora zu lernen.

> „*Die Entwicklung rein lernbasierter NLP-Komponenten verspricht robuste und schnell entwickelbare Systeme, die den Entwickler vom mühsamen Handwerk des Regelschreibens befreien*“ (Rössler 2006, 61).

Allerdings sind nicht alle Systeme statistischer Strategie bequemer zu bedienen als klassische Systeme, da der Aufwand, ausreichend große Korpora zu annotieren, wahrscheinlich nicht geringer als der der Regelerstellung ist (Roth 2002, 106).

Wenn statistische Systeme ohne die Korpus-Annotation die Grammatiken induzieren können, wird dies eine gravierende Verbesserung darstellen. Für diese Zwecke wurden in den letzten Jahren automatische Ansätze entwickelt (van Zaanen 2000; Dennis 2001 etc.). Der Vorteil von rein lernbasierten Ansätzen ist offensichtlich, aber der Nachteil, dass extrahierte Regeln vergleichsweise weniger qualitativ sind (Nagel 2008, 183), kann nur schwer vermieden werden. In der vorliegenden Arbeit wurde neben der Effizienzerhöhung auch die Verbesserung der Resultate berücksichtigt.

Die letzte Frage kann mit der Darstellungsmethode lokaler Grammatiken beantwortet werden. Lokale Grammatiken wurden von Gross (Gross 1993) zur Repräsentation von Kollokationen und festen (und auch häufigen) Redewendungen erstmals vorgestellt. Die Grundidee ist, dass stark variierende linguistische Einheiten durch Automaten repräsentiert werden können. D.h. lokale Grammatiken versuchen einerseits syntaktische Strukturen zu beschreiben und anderseits die Relationen zwischen bestimmten Wörtersequenzen zu bilden.

Nach der Veröffentlichung dieser Theorie wurde sie stets weiterentwickelt (Mohri 1994; Garrigues 1995) und verbreitet. Heute werden lokale Grammatiken auch am CIS (*Centrum für Informations- und Sprachverarbeitung*) in München intensiv erforscht (Mallchok 2004; Nagel 2008; Lee 2008 etc.).

Neben der Theorie werden noch drei praktische Tools Unitex, INTEX und NooJ vom Labor LADL (*Laboratoire d'Automatique Documentaire et Linguistique*), das Gross im Jahr 1968 an der Universität Paris VII gründete, angeboten. Mit UNITEX, welches derzeit die Lexika von 14 Sprachen (Brasilianisches Portugiesisch, Deutsch, Englisch Finnisch, Französisch, Griechisch, Italienisch, Koreanisch, Norwegisch, Polnisch, Portugiesisch, Russisch, Spanisch und Thai) umfasst, können alle Kontexte bzw. Konkordanzen eines Schlüsselwortes aus den Korpora aufgelistet werden, um die Regeln lokaler Grammatiken zu erstellen.

Außerdem ist eine Darstellungsfunktion vom Unitex angeboten. Prinzipiell werden lokale Grammatiken als überschaubare Graphen (siehe Abbildung 2.12) dargestellt und durch endliche Automaten bzw. Transduktoren[3] repräsentiert. Diese Darstellungsform ist viel benutzerfreundlicher als andere äquivalente Repräsentationsformen wie reguläre Ausdrücke oder Ersetzungsregeln (Nagel 2008, 4). Als endliche Automaten sind die Grammatiken leicht anzuwenden.

Im Vergleich zu anderen Grammatik-Theorien sind lokale Grammatiken ein relativ junges Forschungsthema. Trotz vielfältiger Publikationen und Studien in diesem Bereich ist die wissenschaftliche Untersuchung der LG-Induktion immer noch unterpräsentiert. Die Erstellung lokaler Grammatiken ist bis heute noch eine mühsame Handarbeit. Daher besteht eine dringende Notwendigkeit einen automatischen oder semi-automatischen Ansatz zu entwickeln.

1.3 Zielsetzung

Dieser Arbeit liegen theoretische und praxisbezogene Forschungsinteressen zugrunde. Daher werden drei Ziele festgelegt:

[3] Lokale Grammatiken, endliche Automaten & Transduktoren werden bei Gross als Synonyme betrachtet.

- Hinsichtlich der theoretischen und praktischen Konzeption besteht die zentrale Zielsetzung darin, herkömmliche Ansätze der Grammatik-Induktion mit lokalen Grammatiken zu verbinden. Für diesen Zweck soll ein effizienter automatischer Ansatz entwickelt werden.

- Einer der wichtigen Punkte dieser Arbeit ist eine ausführliche Vorstellung lokaler Grammatiken. Aus bereitgestellten Korpora sollen die syntaktischen Informationen automatisch und effizient extrahiert werden. Weiterhin sollen die lokalen Grammatiken induziert werden. Eine praktische Beispielanwendung zur Erkennung von speziellen Firmeninformationen (z.B. Firmennamen, Adresse und Profile etc.) aus deutschen Webseiten soll die Fruchtbarkeit lokaler Grammatiken beweisen.

- Die auftretenden Probleme in der Untersuchung und die entsprechenden Ursachen sowie die ggf. vorhandenen oder potenziellen Lösungen sollen dargestellt und diskutiert werden, um zukünftige Verbesserung und Entwicklung zu ermöglichen.

1.4 Aufbau der Arbeit

Die vorliegende Arbeit besteht aus sechs Kapiteln. Zur Präsentierung der Übersicht können sie folgendermaßen skizziert werden:

In Kapitel 2 werden die Grundlagen über die Theorien, Methoden und Praxisansätze vorgestellt, die für die vorliegende Arbeit relevant sind. Da der in dieser Arbeit entwickelte Ansatz die Kenntnisse interdisziplinärer Forschungsbereiche benötigt, wird der aus maschinellem Lernen, automatischer Grammatik-Induktion und lokalen Grammatiken bestehende Hintergrund dargestellt.

Kapitel 3 präsentiert einen in dieser Arbeit entwickelten effizienten GI-Ansatz (EDSI: Effiziente Destillation der Syntaktischen Informationen). Dabei wird der Algorithmus des Ansatzes detailiert angegeben. Es wird jeweils ein Test auf syntaktischer Ebene im Englischen und im Deutschen durchgeführt.

In Kapitel 4 wird der EDSI-Ansatz mit dem klassischen Ansatz ABL an denselben Korpora durchgeführt, um die Performanz der beiden Ansätze zu vergleichen. Anbei müssen die Ergebnisse aus dem jeweiligen Ansatz qualitativ evaluiert werden, um eine plausible Schlussfolgerung ziehen zu können.

Kapitel 5 stellt ein praktisches System vor, das die Firmennamen, die Adressen und die Firmenprofile aus gecrawlten Webseiten erkennen kann.

In Kapitel 6 findet sich die Zusammenfassung dieser Arbeit. Dabei werden auch die Probleme diskutiert, die bei der Grammatikentwicklung aufgetreten sind. Darüber hinaus wird ein Ausblick für die potenziellen Verbesserungsmöglichkeiten gegeben.

2. Theoretische und methodische Grundlagen

In theoretischer und methodischer Hinsicht sind verschiedene Disziplinen grundlegend für die vorliegende Arbeit, insbesondere maschinelles Lernen, Grammatik-Induktion und lokale Grammatiken. Die wichtigen Theorien und Methoden, auf denen diese Untersuchung basiert, werden in diesem Kapitel detailliert erläutert, um eine Übersicht des Forschungsstandes darzustellen.

Durch die Verknüpfung zwischen der Grammatik-Induktion und maschinellem Lernen entsteht eine neue Forschungsdomäne: automatische/computationale Grammatik-Induktion (AGI), die effiziente Methoden und andere Auseinandersetzungen ermöglicht. Beispielsweise versuchen manche AGI-Ansätze die Lernfähigkeit zu simulieren, um das Wissen aus natürlichen Sprachen zu gewinnen (Mintz 2003; Stumpe et al. 2009 etc.). Im engeren Sinne ist AGI neben der klassischen Forschungstradition auch mit eigenen Regeln und Aufgabestellungen (Adriaans/van Zaanen 2004, 57) etabliert. Eine relativ neue Grammatik-Theorie (lokale Grammatiken) muss ebenfalls vorgestellt werden, da diese ein wichtiger Bestandteil dieser Arbeit ist. Die relevanten Grundlagen werden daher im Zusammenhang mit dem Wissenserwerb und insbesondere mit der Entwicklung der Grammatiken dargestellt.

2.1 Maschinelles Lernen

Aufgrund der zunehmenden enormen Datenflut ist eine rein manuelle Verarbeitung und Analyse sehr aufwendig bzw. nicht mehr durchführbar. Frawley berichtet, dass sich die Datenmenge auf unserer Welt alle 20 Monate verdoppelt (Frawley/Piatetsky-Shapiro/Matheus 1992, 57). Für Menschen ist es oft schwierig aus einem komplexen und unüberschaubaren System vollständige und korrekte Ableitungen durchzuführen (Harbordt 1974, 270). Im Gegensatz dazu lässt sich ein Computer nicht durch die Fülle der Datenflut verwirren. Sobald die Daten vollständig modelliert werden, können die Simulationsprogramme korrekte Ableitungen relativ schnell liefern. Es bedarf daher maschineller Unterstützung, um die Daten verarbeiten zu können. Somit wird aufwendige Handarbeit durch den Einsatz des Computers abgelöst. Ande-

rerseits werden neue Forschungsbereiche oder Methoden durch den Einsatz des Computers herangezogen.

Maschinelles Lernen ist in erster Linie eine auf „Künstlicher Intelligenz" bezogene Disziplin und beschäftigt sich damit, menschliche Lernfähigkeit auf einen Rechner zu übertragen. Lernen wird eigentlich nur Lebewesen zugesprochen, was den Einsatz im technischen Bereich problematisch macht. Erste theoretische Überlegungen zum maschinellen Lernen werden zu Beginn der Künstliche-Intelligenz-Forschung angestellt (Turing 1947/1950). In den fünfziger Jahren erschienen bereits Computerprogramme, die in der Lage waren, ihr Verhalten durch das „Lernen" anzupassen (Samuel 1959, 71). Durch theoretische und praktische Entwicklung seit den achtziger Jahren erzielte maschinelles Lernen vielseitige Fortschritte. Mit Hilfe fortschreitender Computertechnologien wurden verschiedene praktische Ansätze entwickelt, die es ermöglicht, (sehr) große Datenmengen zu verarbeiten, um Wissen effizient und effektiv zu generieren.

Die wichtigste Eigenschaft eines Rechners für die Intelligenzforschung ist die Lernfähigkeit. Aber was ist eigentlich Lernen? Die am meisten verbreitete Definition lautet wie folgt:

> *"Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time."* (Simon 1983, 28)

Lernen markiert also jede Veränderung, die ein System befähigt, bei der zukünftigen Bearbeitung derselben Aufgabe effizienter und besser vorzugehen. Diese Formulierung wird jedoch aus zwei Gründen kritisiert. Einerseits erfasst sie die Phänomene, welche nicht wirklich zum Lernen gehören dürfen. Und anderseits werden nicht alle lernbezogenen Phänomene vollständig abgedeckt (Morik 1993, 3). In Simons Definition wird eine „verbesserte Leistung" als Ziel des Lernens betont, die in der Realität nicht zwangsläufig beim Lernen erfolgt. Um die obigen Kritiken zu vermeiden haben Scott und Michalski neue Definitionen vorgeschlagen. Statt „verbesserte Leistung" wird der verallgemeinerte Begriff „Repräsentation" verwendet, um den Zusammenhang zwischen Lernen und Wissensrepräsentation zu beschreiben.

> *"Learning is any process through which a system acquires synthetic a posteriori knowledge."* (Scott 1983, 360)

> *"Learning is constructing or modifying representations of what is being experienced."* (Michalski 1986, 10)

D.h. das gelernte Wissen aus dem Lernmaterial wird in eine interne Repräsentation überführt und kann in dieser Form genutzt werden. Die Definitionen sind allerdings umfassend und nicht präzise, weil das Wissen im Hinblick auf verschiedene Aufgabenziele in unterschiedlichen Formen dargestellt werden muss, die in den Definitionen nicht erläutert werden. Trotz der Kritiken helfen die Definitionen, die grundlegenden Prinzipien im Bereich von maschinellem Lernen zu etablieren.

Weil bei der Erstellung von Lernprogrammen vielfach psychologische Erkenntnisse genutzt wurden, können diese Programme auch als kognitive Modellierung des menschlichen Lernens betrachtet werden (Schmalhofer 1994). Allerdings ist es nicht das Ziel des maschinellen Lernens, die menschlichen Fähigkeiten technisch exakt nachzubauen, sondern eine dem Menschen vergleichbare Leistung bei der Exploration zu erreichen.

Die maschinelle Lernfähigkeit in natürlichen Sprachen ist für Linguisten besonders interessant. Anhand empirischer Daten werden Ansätze entwickelt, die Grammatiken automatisch zu gewinnen.

> *„Ein Wort ist eine Sequenz von Schriftzeichen, und aufeinander folgende Schriftzeichen sind nicht unabhängig voneinander, sondern aufgrund der Wörter der Sprache eingeschränkt. … Es gibt Algorithmen für das maschinelle Lernen, mit deren Hilfe man Sequenzen erlernen und solche Abhängigkeiten modellieren kann."* (Alpaydin 2008, 7)

Solche Systeme können auf unterschiedlichen linguistischen Ebenen repräsentiert werden. Sogar ein rein auf syntaktischer Ebene aufgebautes System erzielt erfolgreiche Ergebnisse

von praktischer Verwendung, beispielsweise automatische Rechtschreibungskorrektur[4], syntaktisch unterstützte Volltextsuche[5] etc.

2.1.1 Prinzipien des maschinellen Lernens

Maschinelles Lernen verfolgt wie menschliches Lernen immer ein Lernziel, das mit Hilfe der Lernregeln oder der Lernmuster aus gegebenen Materialen erreicht wird. Dieser Prozess kann mit der folgenden Abbildung (Sester 2005, 22) dargestellt werden:

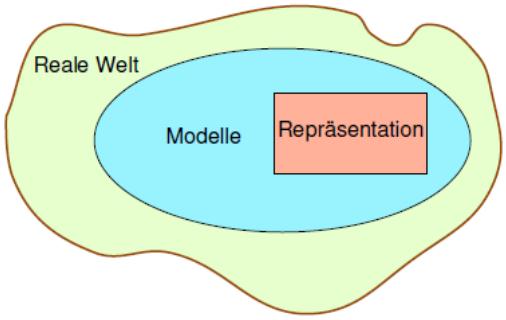


Abbildung 2.1: Von der Realität zur Repräsentation

Ausgehend von der „realen Welt“, die die Grundlage der Lernmaterialien darstellt, werden Ausschnitte und Zusammenhänge zwischen den gegebenen Informationen in die Modelle aufgenommen. Das Modell legt die Lernregeln und die mögliche Schlussfolgerung vor und wird in einem Rechner abgebildet. Repräsentation entspricht einer bestimmten Darstellungsform, die von den Aufgaben und dem Aufgabenziel abhängig ist. Dadurch soll eine möglichst hohe Effizienz der Analyse und/oder der Anwendung gewährleistet werden.

Der Übergang von der Realität auf die Modelle und von dort auf die Repräsentation ist immer mit einem Verlust behaftet (Sester 2005, 22). Zum einen drückt ein Modell die angesehenen Eigenschaften der Realität vereinfacht aus, um eine übersichtliche und berechenbare Untersuchung zu ermöglichen (Stachowiak 1973). Zum anderen bildet die Repräsentation in einer bestimmten Sprache eine eingeschränkte Sicht auf das Modell.

[4] Vgl. Zimmermann (1990) und Erdmenger (1997) etc.

[5] Vgl. Bsiri et al. (2008)

2.1.2 Modellierung durch syntaktische Merkmale

2.1.2.1 Modellierung

Modellierung ist ein wichtiger Arbeitsschritt, um das implizite Wissen der realen Welt in eine explizite Form zu überführen. Die aus den entsprechenden Objekten[6] abstrahierten Modelle werden üblicherweise über ihre Eigenschaften oder kennzeichnenden Merkmale definiert.

Das zentrale Problem für die Modellierung ist, welche Eigenschaften oder Merkmale aus dem Objekt in sein Modell aufgenommen werden sollen.

> *„A model is a representation for a particular purpose. "* (Kaplan 2009, 6)

D.h. die Abbildung der Modelle ist immer vom Aufgabenziel abhängig. Modelle vernachlässigen bestimmte Aspekte, die für die Zielsetzung unwichtig sind, um die Aufgaben in vereinfachter und überschaubarer Weise darzustellen. Dies ist jedoch nicht problematisch, solange sie das zu untersuchende Objekt noch zweckgemäß richtig beschreiben können.

- **Manuelle Modellierung**

  Die Qualität ist von der Kenntnis des Erstellers abhängig. Daher können nur die Merkmale, die für den Ersteller wichtig erscheinen, aufgenommen werden. Diese Vorgehensweise kann jedoch sehr zeitaufwendig sein, wenn viele Einzelfälle modelliert oder ggf. angepasst werden müssen.

- **Automatische Modellierung durchs Lernen**

  Die wichtigen Merkmale können beim Lernsystem aus den (positiven/negativen) Beispielen automatisch ermittelt werden. Weil die Merkmale durch das System nach dem gleichen Kriterium ausgesucht werden, ist dieses Verfahren objektiv. Dadurch entsteht aber auch der Nachteil, dass die Qualität der abgeleiteten Modelle von den Beispielen abhängig ist.

[6] Modelle beziehen sich immer auf ein Objekt oder einen Prozess der Realität. Weil es sich in dieser Arbeit nur um natürliche Sprache handelt, wird hier der Begriff „*Objekt*“ verwendet.

Im maschinellen Lernen kommen statistische Ansätze zunehmend zum Einsatz. Erst Mitte der 1980er Jahre wurden erfolgreiche Ergebnisse in linguistischer Forschung durch die Verwendung der Hidden-Markov-Modelle[7] erzielt. Heute nimmt die statistische Modellierung immer mehr an Bedeutung zu. Im Umfeld der hier durchgeführten Untersuchung werden die Modellobjekte auf natürliche Sprachen eingeschränkt. Die Sprachen werden mit Hilfe der nichtparametrischen Merkmale modelliert. Dabei wurde ein Ansatz entwickelt, der einerseits distributionale Informationen auf der Satzebene statistisch beschreibt und anderseits die darüber hinaus gewonnenen Strukturen modelliert. Das heißt, die Grammatiken werden in Abhängigkeit von syntaktischen Relationen direkt aus den Korpora erlernt.

2.1.2.2 Distributionale Informationen

In der vorgestellten Untersuchung werden distributionale Informationen als Bausteine der Modelle verwendet. Die Bausteine eines Modells bestehen einerseits aus „Data“ und anderseits aus „laws“[8] (Dempster 1998, 251). „Data“ erfassen nicht nur die Daten selbst, sondern auch die Beziehungen zwischen ihnen. Im Falle dieser Arbeit entsprechen sie distributionalen Informationen. Erman & Barren (2000, 37) berichten, dass 55% der englischen Wörter ein Bestandteil distributionaler Information in geschriebener Sprache sind. Distributionale Information bezieht sich auf linguistische Kontexte, in denen ein bestimmtes Wort vorkommt[9] (Redington/Chater/Finch 1998). Die Wörter, die in ähnlichen Kontexten vorkommen, sind durch einander ersetzbar (Harris 1954). Zahlreiche Publikationen und Studien haben gezeigt, dass Wörter aus derselben Kategorie dazu tendieren, viele gemeinsame distributionale Regularitäten zu besitzen (Finch/Chater 1992; Maratsos 1988; Kiss 1973). Das heißt, je semantisch ähnlicher die Wörter sind, desto distributional ähnlicher können sie sein.

> *“words which are similar in meaning occur in similar contexts”* (Rubenstein/Goodenough 1965, 627)

7 vgl. Rabiner/Juang 1986; Bahl/Brown/Souza 1986 etc.

8 „laws“ sind die Interpretationsvorschriften.

9 Distributionale Information wird in der vorliegenden Arbeit auf syntaktische Informationen beschränkt.

> *"words with similar meanings will occur with similar neighbours if enough text material is available"* (Schütze/Pedersen 1995, 4)

Distributionale Information bezieht sich eigentlich auf zwei Relationen. Einerseits die Relation zwischen ähnlichen Wörtern, die an selber Stelle (engl. *Slot*) vorkommen. Andererseits die Relation der Kollokationen, in denen dieselben Wörter oftmals zusammen vorkommen. Solche Informationen können als Merkmale für die Modellierung verwendet werden.

> *"A distributional model accumulated from cooccurrence information contains syntagmatic relations between words, while a distributional model accumulated from information about shared neighbours contains paradigmatic relations between words."* (Sahlgren 2006, 7)

Der Satz ist wahrscheinlich ein universaler Faktor in allen natürlichen Sprachen. Daher ist syntaktische Relation eine der fundamentalen linguistischen Relationen. Die syntagmatische und paradigmatische Relation sind die wesentlichen theoretischen Ausgangspunkte dieser Arbeit. Die zwei Relationen werden häufig in einer 2-dimensionalen Abbildung dargestellt:

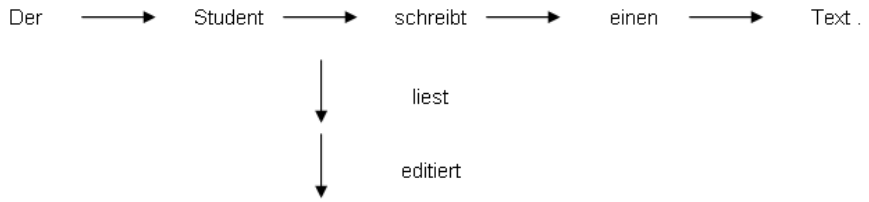


Abbildung 2.2: Syntagmatische und Paradigmatische Relationen

- **Syntagmatische Relation**

  Waagerecht: Syntagmatische Relation bezeichnet eine lineare Relation der linguistischen Einheiten, die in sequenziellen Kombinationen vorkommen (Sahlgren 2006, 6). Diese Relation wird durch Kombinierbarkeit der Einheiten definiert, beispielsweise die Relation zwischen („*der*“, „*Student*“, „*schreibt*“, „*einen*“ und „*Text*“) im ersten Satz. Aber eine syntagmatische Relation zwischen linguistischen Einheiten existiert nur, wenn die Grammatikalität nicht durch die Kombination der Einheiten im Kontext verletzt ist.

Das Modell der syntagmatischen Relation ist die Kollokation, die die inhaltliche Kombinierbarkeit der linguistischen Einheiten bezeichnet.

Besonders interessant sind festgefrorene Satzmodelle (auch Sprichwort oder Idiom genannt):

a) Übung macht den Meister.
b) Better late than never.
c) 熟能生巧。(Übung macht dem Meister)

Das Beispiel (*a*) ist eine feste Redewendung im Deutschen. Die vier Wörter bilden in diesem Kontext gemeinsam eine syntagmatische Relation, und zwar in einer bestimmten Reihenfolge. Eine ähnliche Redewendung ist auch in anderen natürlichen Sprachen zu finden, z.B. (*b*) im Englischen und (*c*) im Chinesischen.

Neben festen Redewendungen besteht die Kollokation oftmals aus häufigen Wortkombinationen in einem Kontext[10]:

d) Der Student hält einen Vortrag.

Aus statistischer Sicht tritt die Kombination aus den Einheiten „*halten*“ und „*einen Vortrag*“ relativ oft auf. Solche Einheiten können als Merkmale in ein Computermodell aufgenommen werden, um ihre syntaktischen Strukturen zu gewinnen.

- **Paradigmatische Relation**

Senkrecht: Paradigmatische Relation bezieht sich auf die Austauschbarkeit der linguistischen Einheiten, die in ähnlichen Kontexten vorkommen können (Sahlgren 2006, 6; Hjelmslev 1974, 46). D.h.: eine linguistische Einheit kann durch eine andere ersetzt

[10] Distributionale Informationen finden sich in Dokumenten an vielen Stellen. In einem bestimmten Text oder Absatz ist es sehr wahrscheinlich, dass besonders viele Wortkombinationen vorkommen. Aber nicht alle Vorkommnisse stellen syntagmatische Relationen dar. Daher entsteht die Notwendigkeit, die Kontexte einzuschränken, um überflüssige Informationen auszuschließen. In der vorliegenden Untersuchung werden syntaktische Relationen nur auf der Satzebene berücksichtigt. Unter dem Begriff „Satzebene“ ist zu verstehen, dass die Sätze oder Teile von Sätzen als Kontext der Untersuchung definiert werden.

werden, aber die Grammatikalität des gesamten Satzes wird durch diese Ersetzung nicht verletzt. Die paradigmatische Relation wird durch Austauschbarkeit der Einheiten definiert, also die Relation zwischen („*schreibt*“, „*liest*“ und „*editiert*“) im obigen Beispiel.

Das Modell der paradigmatischen Relation wird durch austauschbare linguistische Einheiten (Wörter oder Phrasen) abgebildet, die gemeinsame funktionale Ähnlichkeiten besitzen. Die austauschbaren Einheiten kommen nicht gleichzeitig in einem Kontext vor, bilden stattdessen mit denselben Wörtern an der gleichen Stelle unterschiedliche Sätze.

e) Der Professor hält einen Vortrag.

Ein extremes Beispiel dieser Relation sind Synonyme oder Antonyme. Die Synonyme „*Fahrstuhl*“ und „*Aufzug*“ sind fast überall miteinander ersetzbar, ohne einen grammatischen Unterschied zu verursachen. In den allgemeineren Beispielen können „*der Student*“ aus (*d*) und „*der Professor*“ aus (*e*) als eine funktionale Gruppe klassifiziert werden.

Um distributionale Informationen im Zusammenhang mit statistischen Objektmodellen zu verwenden, kann die Methode statistischer Modellierung direkt eingesetzt werden. Das Verfahren modelliert das statistische Verhalten durch die linguistischen Merkmale in Bezug auf die syntaktische Relation. Nach der Berechnung der Wortfrequenz können die linguistischen Einheiten als syntaktische Merkmale unterschiedlich gewichtet werden. Unter der Annahme, dass ähnliche Wörter in ähnlichen Kontexten oft vorkommen, weisen distributionale Informationen aus den homogenen Korpora eine relativ hohe Frequenz auf. Je nach dem Aufgabenziel kann ein Schwellwert festgelegt werden, um die für die Aufgaben wichtigen Merkmale auszuwählen. Ein Satz ist dann eine Kombination aus mehreren Merkmalen mit den Markierungen ihrer Position. Durch den Vergleich zwischen Sätzen entstehen gemeinsame Strukturen, die potentiell die Grammatiken induzieren.

2.1.3 Formalismen zur Repräsentation

2.1.3.1 Wissensrepräsentation

Die aus den zu untersuchenden Objekten abstrahierten Modelle müssen schließlich in einer Form abgebildet werden, die für maschinelle Systeme verständlich ist. Nur in dieser Form kann das Wissen durch einen Rechner repräsentiert und später verwendet werden. Die wesentlichen Bestandteile der Repräsentation sind eine formale Sprache und eine Interpretationsvorschrift (Reimer 1991). Eine Beschreibung der Repräsentation in natürlicher Sprache ist zwar für einen Menschen verständlich, eignet sich aber nicht für die maschinelle Verarbeitung. Stattdessen wird eine formale Sprache (eine Menge Symbole) eingesetzt.

Es gibt inzwischen eine schwer überschaubare Menge an Formalismen der Repräsentation, z.B. logikbasierte Repräsentationen (Nowell 1973); Frames (Minsky 1974) oder Expertensysteme (Gottlob 1990) etc. Die Wahl einer Repräsentationsform ist häufig vom Aufgabenziel abhängig. Eine gute Repräsentation für die Modelle soll grundsätzlich folgende Anforderungen erfüllen (Hellwig 2011):

- Sie ist vollständig in Bezug auf die angestrebte Problemlösung.
- Sie beschreibt die ausschlaggebenden Eigenschaften explizit und unterdrückt irrelevante Faktoren.
- Sie lässt sich effizient mit dem Computer verarbeiten.
- Idealerweise ist sie verständlich für den Benutzer.

Viele Verfahren beruhen auf der deklarativen Repräsentation, die den Anspruch hat, dass die Darstellung sehr übersichtlich und benutzerfreundlich ist. Ein anderer Vorteil ist, dass das System sich leicht erweitern und pflegen lässt. Allerdings ist die Verwendung solcher Repräsentation oft von dem Problem der schlechten Performanz beschränkt (Dorn/Gottlob 1997, 977).

2.1.3.2 Syntaktische Strukturen & formale Grammatiken

Die generative Grammatiktheorie hat sich seit der chomskyanischen Revolution zu einer etablierten Teildisziplin der Sprachforschung entwickelt. Diese Theorie beschreibt nicht das konkrete sprachliche Verhalten, sondern die Fähigkeit des Menschen, unendlich viele (oder neue) Sätze zu verstehen und zu produzieren. Diese Fähigkeit erklärt Chomsky (1965) durch ein komplexes Regelsystem.

Er hat einen formalen Ansatz vorgestellt, der auf vordefinierten Grammatiken basiert. Eine Sprache kann durch die dafür definierten Grammatiken auf syntaktischer Ebene beschrieben werden, das heißt, formale Grammatiken berücksichtigen nur die Syntax der entsprechenden Sprache. Im Hinblick auf das Aufgabenziel der vorliegenden Arbeit wird das Problem der Repräsentationsform über formale Grammatiken gelöst.

Formale Sprachen sind künstlich definierte Sprachen, die dem Computer die Daten oder Informationen zu verarbeiten ermöglichen. Formale Grammatiken bestehen aus einer Menge von Zeichenketten, die aus den Symbolen eines Alphabets aufgebaut sind, und einer Menge von Regeln, die angeben, welche Symbole durch andere ersetzt werden können (siehe Kapitel 2.3.3.1). Gegenüber den natürlichen Sprachen haben formale Sprachen neben expliziten Definitionen für jeden zulässigen Ausdruck und dessen Bedeutung kontextfreie Grammatiken. Dadurch wird eine maschinelle Verarbeitung erheblich erleichtet (oder ermöglicht). Formale Grammatiken werden überwiegend in der Sprach- und Informationsforschung eingesetzt. Sie erlauben vielseitige Anwendungen, wie z.B. die Mustererkennung[11], die Erzeugung neuer Sätze etc. Die Beschreibung der abstrakten Grammatiken kann graphische Darstellungen enthalten. Diese Repräsentationsform wird auch für die in dieser Arbeit extrahierten Grammatiken verwendet. Die Graphen sind nicht nur für Menschen übersichtlich und verständlich, sondern auch geeignet für die spätere maschinelle Verarbeitung.

Es bestehen mehrere Möglichkeiten eine Grammatik zu schreiben. In der Linguistik wird die Beschreibungsform durch theoretische und praktische Anforderungen in Bezug auf die komp-

[11] Vgl. Fu (1982); Juang/Rabiner (2004) haben eine Zusammenfassung über automatische Mustererkennung erstellt.

lexen Eigenschaften der natürlichen Sprachen entschieden. Die in dieser Arbeit extrahierten Grammatiken werden graphisch in Form von Knoten und Rechtspfeilen dargestellt. Jeder Knoten (ein nicht-terminales Symbol) symbolisiert einen Zustand, in dem sich alle Komponenten befinden, die eine gleiche Funktion tragen und gegeneinander ersetzbar sind. D.h. ein Knoten wird von einer Wortklasse abgebildet. Die Rechtspfeile repräsentieren die Überführung vom vorherigen Zustand zum nächsten Zustand, normalerweise von links nach rechts. Sie werden auch als Startsymbol in den Grammatiken verwendet. Da die Sätze der natürlichen Sprachen normalerweise endlich sind, sollen die Grammatiken auch durch ein terminales Symbol (das Endsymbol) geschlossen werden.

2.1.4 Lernstrategien

Maschinelle Lernverfahren beruhen auf unterschiedlichen Inferenz-Strategien, die für verschiedene Aufgaben eingesetzt werden können. Grundsätzlich gibt es drei Schlussarten, wie das Wissen beim Lernen gewonnen werden kann. Die Schlüsse werden folgendermaßen beschrieben:

- **Induktion** – Lernen des Allgemeinen aus dem Speziellen. Induktives Lernen leitet allgemeine[12] Regeln aus einer Menge von einzelnen Fällen ab (Sester 1995, 32). Die Induktion steht heute beim maschinellen Lernen im Vordergrund und ist die gebräuchlichste Form, in logischen Formalismen zu lernen.

  Diese induktiv abgeleiteten allgemeinen Regeln gelten aber nicht zwangsläufig für andere Fälle, die neu auftreten. Deswegen ist ein induktiver Schluss im Sinne von „wahrheitswerthaltend“ nicht logisch korrekt; die Wissenschaft könnte so automatisiert werden (Hess 2004, 9) und wäre infolgedessen überflüssig. Bei der Grammatikentwicklung wird Induktives Lernen dadurch gekennzeichnet, dass systematische Zusammenhänge in einer Abfolge von Sätzen erkannt, beschrieben und in einer Regel festgehalten werden (Henrici 1986, 236).

[12] Hier bedeutet „allgemein“ nur die Gesamtheit aller Beispielfälle, die bei der Induktion getestet werden.

- **Deduktion** – Lernen des Speziellen aus dem Allgemeinen (Sester 1995, 32). Im Gegensatz zur Induktion kommt die Deduktion von allgemeinen Regeln zu einem speziell beobachteten Fall.

  Die aus Regeln deduktiv abgeleiteten Aussagen werden auch eindeutig als wahr klassifiziert. Die Deduktion bezeichnet eine formale Schlussfolgerungsmethode aus der Logik und ist eine wahrheitswerthaltende Inferenz (Hess 2004, 10). Diese Schlussart ist bisher am längsten und gründlichsten untersucht geworden. Bei der Untersuchung von grammatischen Problemen ist deduktives Lernen als ein Erklärungsverfahren zu verstehen, das durch Beispiele beschrieben wird (Henrici 1986, 236).

- **Abduktion**[13]– eine Art von Schlussfolgerung, welche versucht, für eine Reihe von Fakten und Regeln eine (gemeinsame) Erklärung zu ermitteln (Hess 2004, 14). Sie ist wie die Induktion synthetisch und schließt weder vom Allgemeinen auf das spezielle noch vom Speziellen auf das Allgemeine.

  Bei diesem Schluss handelt es sich um eine „Vermutung", die Plausibilität nahelegt. Diese abduktive Schlussfolgerung ist daher ebenso wenig wahrheitswerthaltend.

In linguistischen Forschungen werden häufig induktive und deduktive Lernstrategien eingesetzt. Hinsichtlich kognitionspsychologischer Motivation sind diese beiden Lernstrategien von Vorteil.

Induktives Lernen schließt von konkreten Beispielen auf abstrakte allgemeine Regeln. Konkrete Sprachbeispiele sind offensichtlich einfacher und verständlicher als abstrakte Metasprachen (Schmidt 1990, 161). Ebenso erscheinen grammatische Phänomene mit konkreten Beispielen nicht als gezwungene künstliche Regeln sondern als Ordnungskriterien natürlicher Art (Weisgerber 1982, 121). Bei dieser Art von Lernen ist zur Entwicklung der Regeln Eigenaktivität gefordert. Da Menschen sehr gut induktiv lernen können, ist es immer vorteilhaft, das Lernen möglichst stark auf die induktive Strategie zu stützen (Stangl 2008). Dies kann aus psychologischer Sicht hilfreich für die Untersuchung sein, wie menschliches Lernen effizienter funktionieren kann.

[13] Abduktion wird auch Hypothese genannt.

Obwohl eine induktive Strategie sowohl qualitativ als auch quantitativ durch überzeugendere Argumente unterstützt wird, wird ausdrücklich betont, dass die deduktive Strategie des Wissenserwerbs ihren Platz in bestimmten Kontexten haben sollte. Beispielsweise eignen sich deduktive Lernverfahren zur Ermittlung systematischer Wissensinhalte (Klinge 2007, 85).

Die Induktion spielt beim Aufstellen von Hypothesen über regelhafte Zusammenhänge, z.B. das Erschließen von Grammatiken aus Korpora, eine zentrale Rolle in jeder wissenschaftlichen Forschung. Die vorliegende Arbeit basiert somit auf einer induktiven Strategie.

2.1.5 Überwachtes /Unüberwachtes Lernen

Das Ergebnis des maschinellen Lernens ist eigentlich eine Hypothese zur Lösung der gegebenen Aufgabe, die entsprechend eine möglichst hoch qualitative Ausgabe produziert. Diese Hypothese wird mit verschiedenen Algorithmen auf unterschiedliche Art und Weise gesucht. Nach dem Kriterium, ob ein „Lehrer“ den Lernvorgang überwacht, können maschinelle Lernverfahren in überwachtes und unüberwachtes Lernen gegliedert werden.

- **Überwachtes Lernen** (engl. ***supervised learning***) versucht durch Training die Lernfähigkeit zu erhalten, zielsichere Voraussagen zu treffen. Zu diesem Zweck müssen vom Trainer Beispieldaten zur Verfügung gestellt werden. Diese Trainingsbeispiele beinhalten die Eingabewerte mit den gewünschten Zielwerten. Im Lernprozess erkennt ein „Lehrer“ die Ausgabewerte im Vergleich zu den Zielwerten als korrekt oder falsch, um die Systemparameter zu verbessern. Nach dem Training soll das System in der Lage sein, für jede neue unbekannte Eingabe eine korrekte Ausgabe zu liefern. Überwachte Lernverfahren sind besonders geeignet für die Klassifikationsprobleme, bei denen ein „überwachtes“ Training zur Klassenzugehörigkeit erfolgt.

- **Unüberwachtes Lernen** (engl. ***unsupervised learning***) wird ohne vorausgesagte Ausgabewerte, also nur durch die Eingabemuster, trainiert.

Der in der vorliegenden Arbeit entwickelte Ansatz lernt induktiv aus nicht-annotierten Korpora. In diesem Sinn ist der Ansatz als korpusbasiertes unüberwachtes Lernverfahren zu kennzeichnen. Weil keine Abhängigkeit von lexikalischen Ressourcen besteht, handelt es sich um

ein sprachunabhängiges Verfahren. Einerseits ist dieses Verfahren von Interesse für die grundlegende Forschung zur Lernbarkeit, anderseits erzielt es gute Analyseergebnisse für bestimmte Aufgaben.

2.1.6 Fazit

In den vorangehenden Abschnitten wurden die Grundlagen aus technischer Sicht vorgestellt. Der in der vorliegenden Arbeit entwickelte EDSI-Ansatz basiert im Prinzip auf den statistischen unüberwachten Lernmethoden.

Der Einsatz einer lernfähigen Maschine in der linguistischen Untersuchung baut einen neuen Forschungsbereich auf und bringt die folgenden Vorteile:

- Es ermöglicht, große Korpora zu verarbeiten und zu analysieren. Darüber hinaus ist die aufwendige manuelle Datenverarbeitung nicht mehr notwendig.
- Die Maschine liefert die Resultate durch die vorbereiteten Lernregeln. Sobald die Daten vollständig modelliert sind, können die korrekten und objektiven (im Gegensatz zur manuellen Verarbeitung) Schlussfolgerungen abgeleitet werden.

Die obigen Vorteile erfordern jedoch die folgenden Voraussetzungen:

- Die passenden Merkmale der „realen Welt“ müssen zur Modellierung gefunden und aufgenommen werden.
- Das maschinell gelernte Wissen soll in einer solchen Form präsentiert werden, dass es benutzerfreundlich und für die zukünftige Anwendung oder Erweiterung einfach ist.

Im Falle der vorliegenden Arbeit geht es um die Grammatik-Induktion aus unstrukturierten Korpora. Die Existenz der Merkmale in natürlichen Sprachen wird durch die Theorie „distributionaler Informationen“ begründet. Durch die induktive Strategie werden die Grammatiken, die anschließend als formale Grammatiken präsentiert werden, extrahiert.

Das maschinelle Lernen heutzutage hat jedoch den einen Nachteil, dass die gelernten Resultate im Allgemeinen schlechter ausfallen als beim manuellen Lernen. Dieser Nachteil kann durch eine zusätzliche manuelle Korrektur, die allerdings einen entsprechenden Aufwand verursacht, beseitigt werden.

2.2 Grammatik Induktion

2.2.1 Einleitung

Grammatik-Inferenz ist seit den 60er Jahren ein etabliertes Forschungsfeld im Bereich der künstlichen Intelligenz. Sie beschäftigt sich mit dem induktiven Inferenz-Problem, aus einer Menge von Beispielen korrekte Grammatiken zu gewinnen.

- **Grammatik-Induktion** (**GI**)

  ist ein besonderer Fall des induktiven Lernens. Das Ziel der Grammatik-Induktion besteht darin, aus endlichen Beispielen einer Sprache die Grammatiken oder die äquivalenten Repräsentationen (z.B. Automaten) zu induzieren und so unendliche Strukturen dieser Sprache zu identifizieren. Das zentrale Problem der Grammatik-Induktion ist, dass die induzierten Grammatiken nicht nur die Beispiele abdecken sollen, sondern auch die nicht in Beispielen auftretenden Sätze richtig generieren können müssen (Lucas 1993).

- **Automatische**(/**computationale**) **Grammatik-Induktion** (**AGI**/**CGI**)

  beschäftigt sich mit der Erstellung der maschinellen Modelle, die die Grammatik-Induktion unterstützen. Damit können die Informationen in relativ kurzer Zeit aus einer enormen Menge von Daten gelernt werden. Eine andere wichtige Aufgabe hinsichtlich der Anwendung der AGI auf natürliche Sprachen ist, die Produktion des Satzes in verschiedenen Kommunikationssystemen zu reflektieren, z.B. die Dialoge zwischen Mutter und Kind etc.

Eine Grammatik ist die Syntaxrepräsentation einer Sprache und eine Menge von endlichen Automaten, die die Sätze ihrer Sprache beschreiben können. Chomsky (1965) zufolge befinden sich in jeder Sprache mehrere Grammatiken, die für diese Beschreibung adäquat sind. Daher ist es erforderlich, die linguistische Theorie zu erklären, weshalb Menschen Sprachen mit Hilfe von Grammatiken sehr effizient erwerben können. Nach der Auffassung Chomskys erhält jeder Mensch eine angeborene Universale Grammatik (UG), die die formalen[14] und substantiellen[15] Universalien aller Sprachen enthält. Das heißt, die UG-Theorie beschreibt den angeborenen und autonomen Mechanismus des menschlichen Spracherwerbs (engl. *language acquisition*). Als eine der zugrunde liegenden Annahmen der generativen Grammatiken wird die Universalgrammatik als eine Reihe von Regeln verstanden. Die generative Grammatik wurde in zahlreichen Studien zum kindlichen Spracherwerb untersucht und bildet einen interdisziplinären Bereich aus Linguistik, Psychologie, Audiologie, Neurophysiologie und Computertechnologie (Adreaans/van Zaanen 2004, 58). Durch Imitations-, Verstehens- und Produktionsexperimente (vgl. Berko 1958; Goldin-Meadow /Seligman/Gelman 1976; Mintz 2003) wurde bewiesen, dass Kinder beim Erlernen ihrer Muttersprache nicht nur die vorgegebenen Beispiele nachahmen, sondern auch Morpheme und Strukturen in der Zwei-Wort-Phase interpretieren und korrekte Flexionsformen von Wörtern bilden, die sie zuvor noch nie gehört haben. Die Experimente bestätigten daher die Annahme, dass Grammatikregeln bereits in der Kindersprache erworben und verwendet werden.

Bis heute wird eine Vielzahl von Publikationen veröffentlicht, die anstreben, effektive und effiziente Inferenz-Methoden zu finden und zur Lösung für praktische Probleme beizutragen. Die Methoden der Grammatik-Induktion werden in vielseitigen Bereichen praktisch eingesetzt.

Eine klassische Anwendung der Grammatik-Induktion ist die Untersuchung (Gold 1967), den Erwerbsmechanismus der natürlichen Sprachen zu modellieren. Eine andere wichtige typische Anwendung in der linguistischen Forschung ist die zur Erkennung syntaktischer Muster (Fu 1982). Außerdem lassen sich Ansätze, die zwar zurzeit noch nicht praktisch anwendbar, po-

[14] Formale Regeln, die es in allen natürlichen Sprachen gibt.

[15] Kategorien, die es in allen natürlichen Sprachen gibt.

tenziell aber sehr interessant sind, in der Literatur finden. Beispielsweise hat Hutchens (1995) ein System, das auf Shannon's Markov Modell[16] basiert, entwickelt, das zufällige Texte automatisch generiert.

In Bezug auf die Webseiten können die GI-Methoden auch für Data-Mining verwendet werden. Beispielsweise versuchen Levene & Borges (2000) das Verhalten von Benutzern aus ihren Navigationspatterns zu lernen. Chidlovskii et al. (2000) berichten, dass die GI-Methoden für die Generierung der Resultate der Meta-Suche eingesetzt werden können.

Außer linguistischen Anwendungen werden die GI-Methoden auch zur Erkennung biologischer Sequenzen (Gavrilis/Tsoulos/Dermatas 2006) und zur Generierung der Musik (Cruz-Alcazar/Vidal-Ruiz 1997/1998; Conklin 2003) verwendet.

Aus verschiedenen Blickwinkeln können die in den letzten fünfzig Jahren entwickelten GI-Ansätze unterschiedlich klassifiziert werden.

- **Regelbasierte (symbolische) Systeme vs. statistische (distributionale) Systeme**:

  Ein regelbasiertes System wird mit manuell kodierten Regeln angefüllt. Dagegen lernt ein statistisches System, aus vorhandenen Training-Beispielen geeignete Modelle für die Problemlösung zu finden.

- **Deterministische Systeme vs. stochastische Systeme**:

  Bei stochastischen Systemen spielen die Wahrscheinlichkeiten eine wichtige Rolle, wohingegen ein deterministisches System binäre Entscheidungen treffen kann.

- **Unüberwachte Systeme vs. überwachte Systeme**:

  Ein überwachtes System benötigt im Vergleich zum unüberwachten System während des Trainings nicht nur die Eingaben, sondern auch die Zielausgaben.

Im Folgenden werden ein paar regelbasierte und statistische Systeme vorgestellt, die in Bezug auf den in dieser Arbeit entwickelten Ansatz relevant sind.

[16] Vgl. Shannon (1951)

2.2.2 Lernmodelle der (automatischen) Grammatik-Induktion

Im Hinblick auf die Logik der Schlussfolgerung ist eine Induktion deduktiv nicht machbar, weil es unmöglich ist, eine universale Regel aus endlichen Beispielen zu schließen. Daher ist die Grammatik-Induktion der natürlichen Sprachen nicht nur schwierig, sondern auch unmöglich. Einerseits sind die Sätze einer natürlichen Sprache sehr wahrscheinlich unendlich und anderseits existieren viele aus linguistischer Sicht ungrammatische Ausdrücke (besonders in gesprochenen Sprachen) in den Sprachen (Hasan et al. 2007). Allerdings ist das Ziel der Grammatik-Induktion in den meisten Fällen nicht, die Sprache exakt nachzubauen, sondern sich ihr ausreichend anzunähern und sie so zu modellieren. Die Induktion kann sowohl auf der syntaktischen Ebene als auch auf der semantischen Ebene durchgeführt werden. Die vorliegende Untersuchung fokussiert allerdings nur den Erwerb von syntaktischen Informationen der Zielsprache.

2.2.2.1 Generelles Lernmodell

Eine generelle Lernmaschine der Grammatik-Induktion kann nach der Trainingsart wie folgt dargestellt werden (Fu & Booth 1986):

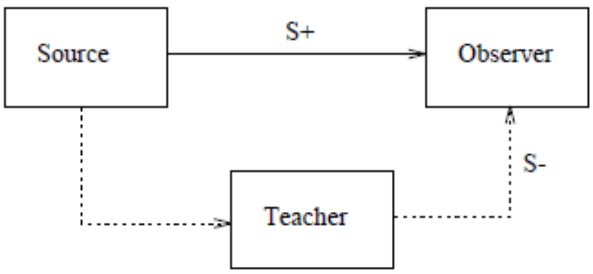


Abbildung 2.3: Struktur einer Grammatischen Inferenz-Maschine

Das Symbol „S+“ repräsentiert eine Menge von gültigen Beispielsätzen, die aus einer Sprache ausgewählt werden, während „S-“ eine Reihe von ungültigen Sätzen bezeichnet, die durch

einen externen Lehrer[17] formuliert werden. Ein maschinelles Lernmodell besteht üblicherweise aus den folgenden Komponenten:

- **Lerner**: Der „Observer“ ist in dieser Untersuchung ein Computerprogramm.
- **Ziel**: Was soll der Lerner erreichen?
- **Informationsquelle**: Die „Source“ und der „Teacher“ bieten dem Lerner positive und/oder negative Beispiele an.
- **Performance**: Die Performance wird vermessen und so der Lernansatz ausgewertet, z.B. die Präzision, die Effizienz etc.

Wobei der Lehrer nicht bei allen Lernmaschinen erforderlich ist. Die meisten statistischen Ansätze lernen allein aus positiven Beispielen.

2.2.2.2 Relevante Lernmodelle der Grammatik-Induktion

Die induktive Inferenz der computationalen Lern-Theorie (engl. *computational learning theory*) wird hauptsächlich durch drei Lernmodelle beschrieben. Jedes Modell enthält ein eigenes Lernprotokoll und ein entsprechendes Kriterium zur Evaluation des Lernens.

Lernmodell der Identifikation im Limit (engl. model of *identification in the limit*)

Eines der ersten Lernmodelle der Grammatik-Induktion hat Gold (1967) vorgestellt. In seiner Arbeit hat er ein Lernspiel präsentiert, welches die Lernfähigkeit definiert. In dem Spiel gibt es zwei Parteien: ein Informant und ein Lerner. Für das Spiel stehen noch eine Klasse von Sprachen sowie eine Repräsentationsmethode dieser Sprachen zur Verfügung. Die Zeit wird als Beschränkungskriterium verwendet. Zu jedem Punkt der Zeitsequenz wählt der Informant eine Sprache aus und aus dieser Sprache werden ein paar Strukturen an den Lerner übergeben. Hat der Lerner die Informationen erhalten, soll er auf Basis dieser Informationen die Sprache identifizieren. Falls der Lerner inner-

[17] Der Lehrer muss über eine ausreichende Kenntnis dieser Sprache verfügen.

halb der Zeitbeschränkung die Sprache immer richtig abschätzt, ist diese Sprache lernbar. Damit wird eine der wichtigen Prinzipien in Golds Arbeit beschrieben:

> *"Using information presentation by recursive text and the generator-naming relation, any class of languages which contains all finite languages and at least one infinite language L IS NOT identificable in the limit."* (Gold 1967, 470)

Eine Klasse Sprachen, die nicht nur endliche Sprachen sondern auch mindestens eine unendliche Sprache enthält, ist nicht lernbar. Solche Klassen werden als „unendliche Elastizität" bezeichnet. Das heißt, es gibt unendliche Sequenzen in der realen Sprache und der Lerner hat zu jedem Zeitpunkt des Lernprozesses unendliche Optionen zu wählen. Daher ist es unwahrscheinlich, eine richtige, stabile Lernstrategie in einem beschränkten Zeitraum zu gewinnen.

Des Weiteren hat Gold in der Arbeit festgestellt, dass die Zielsprache in seinem Experiment allein durch positive Strukturen nicht erkennbar ist. Eine richtige Identifikation kann nur durch sowohl positive als auch negative Beispiele ermöglicht werden. Dahingegen hat Solomonoff (1969) einen wahrscheinlichkeitsbasierten Algorithmus entwickelt, der nur aus positiven Beispielen lernen kann. Die Grundidee dieses alternativen Lernmodells kann mit Hilfe der algorithmischen Komplexitätstheorie formuliert werden, die auf einer tiefen Analyse computationaler Komplexität des Objekts und seiner vordefinierten Wahrscheinlichkeit basiert. Kurz gesagt haben die Objekte, die einfacher zu berechnen sind, eine höhere Wahrscheinlichkeit. Diese Idee ist für den Lernprozess von enormem Vorteil, weil damit eine Annäherung von kooperativen linguistischen Verhaltensweisen formuliert werden kann. Wenn ein Lehrer eine Sprache lehren will, wählt er sehr wahrscheinlich zunächst die Beispiele, die leicht zu analysieren sind, aus. Falls mehrere Grammatiken aus einer Sprache als Kandidaten gelernt werden, dann ist mit der algorithmischen Komplexitätstheorie einfach zu schließen, dass die beste Grammatik die höchste Wahrscheinlichkeit besitzt.

Außerdem gibt es die Kritiken (vgl. Adriaans/van Zaanen 2004, 59) an Golds Experiment, dass diese Sprachidentifikation kein wirklich induktiver Ansatz ist. Der Lerner

wartet einfach ab, bis er ausreichende Informationen erhält, um daraus die Grammatiken deduktiv herzuleiten. Außer der einzigen Einschränkung, dass jede Struktur aller Sprachen in dem Zeitraum mindestens einmal ausgewählt werden muss, ist die Präsentationsform der Sprachen für den Informanten vollkommen unbeschränkt. Im Falle von menschlicher Kommunikation sollten die beiden Parteien vielleicht viel kooperativer sein.

Lernmodell mit Query (engl. *query learing model*)

Basierend auf Golds Idee hat Angluin (1980) ein neues Modell entwickelt. In diesem Lernmodell steht ein Lehrer (Orakel) für die Identifikation zur Verfügung, der Antworten auf spezifische Anfragen liefern kann, die durch den Inferenz-Algorithmus über die Grammatik generiert werden. Typische Anfragen sind folgende:

- **Mitgliedschaft** (engl. *membership*). Die Eingabe ist eine Sequenz. Wenn sie bei der Grammatik generiert werden kann, wird ein „*yes*“ als Ausgabe geliefert. Andernfalls ein „*no*“ (Angluin 1988, 320).

- **Äquivalenz** (engl. *equivalence*) Die Eingabe ist diesmal eine Grammatik. Wenn diese Grammatik dieselbe Sprache wie die zu identifizierende Grammatik erzeugen kann, ist die Ausgabe „*yes*“. Ansonsten „*no*“ (Angluin 1988, 320). Für den Fall, dass die Antwort „*no*“ ist, wird ein String in symmetrischer Differenz der beiden Sprachen, die durch die zwei Grammatiken jeweils generiert werden, zurückgeliefert. Das zurückgelieferte String wird als Gegenbeispiel (engl. *counter-example*) bezeichnet.

Dieses Paradigma, dass Orakel die Antworten an die Queries liefern, wird „minimaler adaptiver Lehrer“ (engl. *minimally adequate Teacher)* genannt (Angluin 1987, 88). Die Mitgliedschaft-Queries liefern zwar nur binäre Informationen zurück, spielen aber oft eine wichtige Rolle zur effizienten und exakten Identifikation. Beispielsweise können definite endliche Automaten nicht durch Äquivalenz- und Mitgliedschaft-Queries effizient identifiziert werden, da die Identifikation allein durch Äquivalenz-Queries sehr zeitaufwändig ist (Angluin 1987/1990).

Lernmodell PAC (engl. *probably approximately correct learning*)

Valiant (1984) hat ein komplett neues Lernkonzept vorgestellt. PAC-Lernen ist ein distributionsunabhängiges statistisches Modell, das aus zufälligen Beispielen lernt. Es geht davon aus, dass die verwendeten zufälligen Beispiele eine beliebige unbekannte Distribution haben. Der Inferenz-Algorithmus erhält ein Beispiel als Eingabe und produziert eine Grammatik als Ausgabe. Die Ergebnisse der Identifikation werden durch zwei Parameter der Präzision und des Konfidenzintervalles ausgewertet, die in den Inferenz-Algorithmus eingegeben werden. Die Fehlerquote der produzierten Grammatik wird in Bezug auf die zu identifizierende Grammatik und die Distribution definiert. Wenn die Fehlerquote kleiner als die Präzision ist, wird diese Inferenz der Grammatik als erfolgreich bezeichnet.

Die aus diesem statistischen Ansatz gewonnenen Grammatiken sind aus rein linguistischer Sicht sehr wichtig für die Identifikation der Sprachen in einem bestimmten Zeitlimit. Die Grundidee hinter dem PAC-Lernen ist, dass, falls das Lernen kein hundertprozentig richtiges Resultat gewährleisten kann, die Fehlerquote beim Lernen zumindest möglichst minimiert wird. Daher wird eine für den Benutzer akzeptable Fehlerquote im Modell vordefiniert. Leider kann das PAC-Lernen allein in der Distributionsform noch keine linguistischen Klassen lernen. Im Hinblick auf die algorithmische Komplexitätstheorie werden die Lernbedingen vom PAC-Modell reduziert. PAC-s ist ein einfaches PAC-Modell (engl. *Simple-PAC*). Es lernt aus einfachen Strukturen, die eine hohe statistische Distribution in ihrer Sprache aufweisen.

Neben der theoretischen Forschung werden noch weitere praktische Ansätze entwickelt, um die Korpora der natürlichen Sprachen mehr oder weniger erfolgreich und effizient zu verarbeiten. Wolff (1980/2003) hat ein SP-Modell, das auf der Idee MLE (engl. *Minimum Length Encoding*[18]) basiert, vorgestellt. Dieser Ansatz sucht den minimalen Wert von (G + E), wobei G die Länge der entwickelte Grammatik und E die binäre Länge der durch diese Grammatik kodierten Rohdaten ist. Mit diesem Prinzip werden

[18] MLE ist ein Oberbegriff für „Minimum Message Length encoding“ (MML) und „Minimum Description Length encoding“ (MDL) (siehe Wolff 2003, 3)

trivial kleine Grammatiken[19] und übergroße Grammatiken[20] induziert. Methodologisch ähnlich wie das SP-Modell hat van Zaanen (2000) den Algorithmus ABL (engl. *Alignment-Based Learning*) vorstellt. Dennis (2001) hat Ein Frage-Antwort-System (engl. *Question-Answer-System*) SPM (engl. *Syntagmatic-Paradigmatic Model*) entwickelt.

Alle Lernmodelle von Gold, Query und PAC betonen die Wichtigkeit der computationalen Effizienz der Inferenz-Algorithmen. Grammatik-Induktion ist ein computational sehr aufwändiges Problem. Dies ist bis heute noch immer der Hauptfaktor, der die tiefe Untersuchung der großen Korpora verhindert.

2.2.3 Klassische regelbasierte (oder symbolische) Systeme

Regelbasierte Systeme beruhen auf rein intellektuell erzeugtem Wissen. Das abstrakte Wissen wird in Form von symbolischen Regeln dargestellt. Solche Ansätze werden bereits in der linguistischen Forschung vielseitig verwendet, z.B. für die morphologische (Karttunen et al. 1996) oder die syntaktische Analyse (Samuelson /Voutilainen 1997; Lin 1998; Butt/King 2002). Zur Kombination können bei den Ansätzen lexikalische Ressourcen eingesetzt werden.

2.2.3.1 SPARSER

Regelbasierte Systeme werden für die Erkennung von Eigennamen (engl. *named entity recognition, NER*) ziemlich erfolgreich eingesetzt. McDonald (1993) hat das System SPARSER bei der MUC[21] (engl. *Message Understanding Conference*) präsentiert, das natürliche Sprache zu verstehen versucht und die lexikalischen Informationen aus Korpora erwerben kann. Anbei

[19] Falls die Rohdaten E sehr groß sind, sind die Grammatiken G relativ klein. (Wolff 2003, 3).

[20] Falls die Grammatiken G sehr groß sind, sind die Rohdaten E relativ klein. (Wolff 2003, 3).

[21] Message Understanding Conferences (MUCs) sind eine Reihe von Forschungswettbewerben, die von NRAD (engl. the Naval Command, Control and Ocean Surveillance Conter) organisiert und von DARPA (engl. the Defense Advanced Research Projects Agency) finanziell unterstützt wurden, um neue Methoden der Informationsextraktion zu entwickeln (sieht auch http://www.muc.saic.com).

wurde der Begriff PNF (engl. *Proper Name Facility*) zum ersten Mal klar definiert, der hilft, die Eigennamen zu erkennen und zu klassifizieren.

Das wichtigste Konzept dieses Systems ist „interne und externe Evidenz" (engl. *intern and extern evidence*). Die externe Evidenz kann die Kontexte in der Untersuchung berücksichtigen, die bei vorherigen vorgestellten rein listenbasierten Ansätzen vernachlässigt wurden.

- **Interne Evidenz**

  beschreibt die Eigenschaften der Wörter, die zu klassifizieren sind. Beispielsweise bezeichnen „*Ltd*" und „*G.m.b.H*" eine Firma, aber ein bekannter Nachname weist meistens auf eine Person hin (McDonald 1993, 32). Um eine relativ hohe Performance der Erkennung und der Klassifikation zu erreichen, werden hier oft lexikalische Ressourcen (z.B. eine große Menge von Gazetteers oder eine Liste von bekannten Namen) eingesetzt. Lexikalische Ressourcen beinhalten die Informationen, die sehr hilfreich dabei sind, die Wörter in die vordefinierten Kategorien einzuordnen. Weil diese Klassifizierung von lexikalischen Ressourcen abhängig ist, hat sie ein paar Nachteile, beispielsweise die nicht im Lexikon erfassten Wörter oder die mehrere Kategorien tragenden Wörter können evtl. falsch klassifiziert werden.

- **Externe Evidenz**

  berücksichtigt den Kontext, in dem das zu klassifizierende Wort vorkommt, um die erwähnten Nachteile interner Evidenz zu verringern/beseitigen. Die Grundannahme dieser Evidenz basiert auf der offensichtlichen Beobachtung, dass Eigennamen (für Menschen, Organisationen & Firmen etc.) charakteristische Eigenschaften (z.B. Verwendung in Gazetteers) besitzen und in typischen Kontexten (z.B. Trigger-Wörtern oder syntaktischen Strukturen) vorkommen. Solche Eigenschaften und Kontexte, die mit den Eigennamen zusammen eine syntaktische Relation bilden, können in hohem Maße dazu beitragen, die Eigennamen zu identifizieren (McDonald 1993, 33). Genauer gesagt, versuchen die internen Eigenschaften (Kontexte), die möglichen Eigennamen zu erkennen. Darüber hinaus dienen die externen Kontexte dazu, die Eigennamen von den nicht-Eigennamen zu unterscheiden. Daher ist die externe Evidenz eine sehr bedeutungsvolle Maßnahme, die eine hohe Präzision gewährleistet. Einerseits kann das vordefinierte Le-

xikon in der Realität nie komplett sein. Das nicht erfasste Wort „*DaCaPro*“ kann aber mit Hilfe des Kontextes „*eine deutsche Firma*“ richtig als Firmenname erkannt werden. Andererseits können die mehrfach kategorisierten Wörter mit dem Kontext disambiguiert werden. Beispielsweise lässt sich „*Sara Lee*“ durch „*Ms.*“ als Personenname von Firmennamen unterscheiden. Die Maßnahme externer Evidenz für die Klassifikation ist allerdings nur hilfreich, wenn ein solcher Kontext wirklich voraussagekräftige Informationen liefern kann.

Interne und externe Evidenz bilden das Kernkonzept des SPARSER-Systems, das bei allen später entwickelten regelbasierten NER-Systemen genutzt wurde.

Vorgehensweise

Das SPARSER hat einen Vorbereitungsprozess, in dem die eingegebenen Texte tokenisiert und die Trigger-Wörter markiert werden. Danach kann SPARSER prozedural durchgeführt werden. Der Verarbeitungsprozess vom SPARSER besteht grundsätzlich aus drei Schritten (McDonald 1993, 34):

- **Erkennen** (engl. *delimit*):

  In dieser Phase werden so viele Wörter eingelesen bis man auf ein kleingeschriebenes Wort stößt. Alle benachbarten großgeschriebenen Wörter bilden eine potenzielle Wortsequenz. Die potenziellen Wortsequenzen werden anhand von Lexika und Merkmalen (großgeschriebene Wörter oder Interpunktion „*&*“ etc.) erkannt oder aussortiert. Anders gesagt, werden die Wortsequenzen zunächst mit endlichen Automaten erkannt und dann durch die Nachbearbeitung genau abgegrenzt.

- **Klassifizieren** (engl. *classify*):

  In dieser Phase wird zunächst interne Evidenz durchgeführt. Ein Lexikon oder eine Namensliste hilft dabei Wörter wie Ortsnamen, Titel von Personen oder andere Hinweise zu identifizieren. Ein am Satzanfang großgeschriebenes unbekanntes Wort wird danach durch externe Evidenz klassifiziert. Die Wortsequenzen werden anhand von Lexika, Namenslisten und ggf. hilfreichen Kontexten in die vordefinierten Kategorien eingeordnet.

- **Rekord** (engl. *record*):

  Im Erkennungsprozess wird noch ein weiteres Diskursmodell erstellt. Die klassifizierten Wortsequenzen und ihre entsprechenden Markierungen werden mit Hilfe dieses Modells interpretiert und als Referenz für die spätere Erkennung von anderen Wortsequenzen verwendet.

2.2.3.2 NYU-System

Weil das SPARSER-System sich nur an der Erkennung der Eigennamen orientiert, verzichtet es von Anfang an auf eine Analyse der syntaktischen Satzstruktur. Hingegen richtet das von Grishman (1995) entwickelte NYU-System (engl. *New York University*) den Fokus auf eine voll syntaktische Analyse. Das Ziel vom NYU-System ist es, mit Hilfe eines Pattern-Matching Ansatzes die Stärke und Schwäche von Teil- und Voll-Pasing herauszufinden. Die syntaktische Analyse hat dabei zwei Hauptvorteile. Einerseits kann sie durch unterschiedliche semantische Relationen linguistische Strukturen generieren (Beispielsweise ist die Struktur der Hauptsätze meist identisch, wenn die Verben dieselbe semantische Relation besitzen, wie: „*zu besiegen*“ und „*zu entlassen*“). Anderseits können paraphrastische Relationen zwischen unterschiedlichen syntaktischen Strukturen erfasst werden (z.B. „*X besiegt Y*“, „*Y wird bei X besiegt*“ und „*Y, der X besiegt*“). Es gilt jedoch auch, die Nachteile des NYU-Systems zu nennen:

- Das System läuft sehr, beinahe zu langsam, um die Resultate in einer vernünftigen Zeit zu liefern. Das heißt, die Effizienz des Systems ist zu niedrig.
- Globales Parsing führt den Prozess manchmal zu Fehlern. Vergleichsweise kann ein rein lokales Parsing oft bessere Resultate liefern.
- Die Erweiterung syntaktischer Konstruktionen benötigt stets ein neues Szenario. Das heißt, die Pflege des Systems ist sehr aufwendig.

Die obigen Nachteile bestehen aus der tiefen und breiten Analyse mit kontextfreien Grammatiken (engl. *context-free grammar*, CFG), die bei der praktischen Verwendung das Performance-Problem verursachen. So ist beispielsweise das LOLITA-System (Gariglia-

no/Urbanowicz/Nettleton 1998), das auch die syntaktische Satzstruktur untersucht, wegen desselben Problems fehlgeschlagen.

Weil flache Regeln, die sich auf lokale Phänomene konzentrieren, im Vergleich zu voll syntaktischer Analyse weniger komplex sind, stützen sich viele regelbasierte Systeme auf endliche Automaten (engl. *finite-state automata*), die eine höhere Effizienz versprechen. Um den Umstand zu vermeiden, dass unterschiedliche Regeln in unterschiedlichen Reihenfolgen zu unterschiedlichen Ergebnissen führen können, werden Kaskade-Ansätze (siehe Abney 1996) eingesetzt, die die Regeln organisieren. Das heißt, die originalen Eingaben werden durch endliche Automaten in einer bestimmten Reihenfolge stufenweise verarbeitet.

2.2.3.3 Ansatz mit lokalen Grammatiken

Regelbasierte Systeme tendieren dazu, endliche Automaten anstelle von kontextfreien Grammatiken als Verarbeitungsmittel einzusetzen. Friburger und Maurel (2002b) haben ein System vorgestellt, das auf lokalen Grammatiken (siehe Kapitel 2.3) basiert und den Prinzipien vom SPASER-System nachfolgt, um so die Eigennamen zu identifizieren und zu klassifizieren. Das System besteht grundsätzlich aus zwei Teilen:

1) Vorverarbeitung durch das Intex-Tool[22]:

- Weil die Interpunktion „ **.** “ einerseits als Zeichen des Satzendes andererseits als Bestandteil der Abkürzung (z.B. Personenname:*„J. Dupont*“; Organisation: „*N.A.T.O.*“; abgekürztes Wort: „*Chap. 5*“) verwendet werden kann, müssen alle Satzenden und andere Ambiguitätsfälle im Text mit Hilfe des Intex erkannt werden, sodass die potenziell falsche Erkennung vermieden wird.

- Der Text wird aus morphosyntaktischer Sicht mit der Unterstützung von Lexika markiert, die umfangreiche Informationen (grammatische Kategorien, semantische Eigenschaften und Lemmata etc.) von Wörtern enthalten. Solche Informationen sind bei der Ermittlung von Eigennamen sehr hilfreich.

[22] Siehe Kapitel 1.2

2) Kaskadierende Verarbeitung durch endliche Automaten: Die dahinterstehende Idee ist sehr einfach. Endliche Automaten werden in einer genauen Reihenfolge am Text durchgeführt, um die gesuchten Pattern zu extrahieren.

- Prozesskaskade 1: Lokale Grammatiken beschreiben linke und rechte Kontexte eines Kandidaten. Falls dieser Kandidat als ein Eigenname erkannt wird, wird er klassifiziert.

- Prozesskaskade 2: Die in Kaskade 1 gefundenen Eigennamen werden im Text entsprechend markiert.

In dem vorgestellten System werden endliche Automaten zum Einsatz gebracht, die den Text vorverarbeiten und die Eigennamen ermitteln. Daher ist der Ansatz sehr einfach und effektiv. Ein Evaluierungstest an einem Korpus (mit ungefähr 80.000 Wörtern) der Zeitung *Le Monde* zeigt, dass das System eine Präzision über 94% mit einem besonders hohen Recall erreicht hat. Friburger zufolge liefert das System auf anderen Korpora vergleichsweise schlechtere Resultate, obwohl alle Korpora ausschließlich aus Texten von Zeitungen wie *Le Monde* bestehen. Ein von Friburger angegebener Grund ist, dass der Schreibstil von anderen Zeitungen viel spontaner als bei *Le Monde* ist. Das bedeutet aber auch, dass das verwendete vollständige Lexikon sehr viel zu den Ergebnissen beitragen kann.

Üblicherweise erzielen regelbasierte Systeme im Vergleich zu statistischen Systemen bessere Ergebnisse. Allerdings werden sie auch oft kritisiert, weil die manuelle Erstellung und die zukünftige Veränderung der Regeln sehr mühsam und aufwendig sind.

2.2.4 Statische oder Lernbasierte Systeme

Ein Schwerpunkt der Sprachtechnologie-Forschung der letzten Jahre liegt auf statistischen oder automatischen, lernbasierten (engl. *machine learning*) Methoden (Collin 1997; Charniak 2000; Klein/Manning 2004; van Zananen 2004 etc.). Damit können große Korpora der natürlichen Sprachen analysiert und ausgenutzt werden. Der größte Vorzug von statistischen Systemen ist, dass das arbeitsaufwändige, manuelle Schreiben und Testen der Regeln durch einen automatisierten Prozess ersetzt wird. Die ersten statistischen Systeme wurden in den Zeiten

der MUC-6 (1995) und MUC-7 (1998) entwickelt. Sie erzielen zwar im Vergleich zu regelbasierten Systemen schwächere Ergebnisse, sind aber robust und schnell entwickelbar (Rössler 2006, 59). Ein anderer Vorteil von statistischen Systemen ist meist die extrem große Abdeckung der Grammatiken. Allerdings muss dabei immer eine gewisse Fehlerquote der Ergebnisse in Betracht gezogen werden. Die Verlässlichkeit der Ergebnisse kann durch eine menschliche Korrektur gewährleistet werden. Dies bedeutet zwar einen zusätzlichen Aufwand, ist aber immer noch kostensparsamer als das rein manuelle Verfahren.

Die meisten bewährten statistischen Systeme basieren auf dem Schema des überwachten Lernens und werden an annotierten Korpora trainiert. Die Annotation enthält gezielte hilfreiche Informationen, um die Regeln abzuleiten. Es findet sich jedoch auch Kritik an solchen Systemen. Beispielweise, dass die Kosten, die es benötigt, um ausreichende Korpora für ein ausgefeiltes statistisches System aufzubauen, berücksichtigt werden sollten. Der Aufbau eines solchen Systems benötigt wahrscheinlich ähnlich viele Ressourcen wie der Aufbau eines leistungsfähigen regelbasierten Systems (Roth 2002, 112). Um diesen Nachteil zu vermeiden, greifen statistische Systeme auf nicht-überwachte Lernverfahren zurück, die die Regeln aus nicht-annotierten Trainingskorpora lernen können (Nagel 2008, 183). Unüberwachte Systeme erzielen gute Analyseergebnisse in bestimmten Aufgabenbereichen: z.B. für Part-of-Speech-Tagging (Brill 1993) oder Wortbedeutungs-Disambiguierung (Schütze 1998).

Statistische Systeme versuchen die Grammatiken durch lernbasierte Verfahren zu extrahieren. Dies führt dazu, dass automatische Grammatik-Induktion zu einem bevorzugten Forschungsbereich geworden ist. Im Folgenden werden die wichtigen Beiträge vorgestellt, die für das Verständnis der typischen Ansätze dieses Gebietes und des Systems in dieser Arbeit bedeutsam sind.

2.2.4.1 ABL

Der von van Zaanen (2000) vorgestellte Ansatz „Alignment-basiertes Lernen“ (engl. *Alignment-Based Learning*, *ABL*) ist ein Korpus-basierter, nicht-überwachter Lernansatz ohne Eingriff des Mensches. ABL lernt aus nicht-annotierten Korpora syntaktische Satzstrukturen und

gegeneinander austauschbare Konstituenten, die jeweils paradigmatische und syntagmatische Relationen[23] aufweisen.

Der dem ABL zugrunde liegende Gedanke ist, dass die Grammatiken, die das Korpus ursprünglich generiert haben, aus diesem Korpus induziert werden können. Allerdings sind diese Grammatiken in der Eingabe unbekannt.

Das Kernstück vom ABL-Ansatz ist der EDA (engl. *Edit Distance Algorithm*) (Wagner/Fischer 1974), der identische Wortsequenzen ermittelt. Der EDA wurde ausgehend vom Levenshtein-Distanz-Algorithmus (Levenshtein 1966) erweitert und kalkuliert die minimalen Kosten der Editierungsoperationen (**Einfügen**, **Löschen** und **Ersetzen**) beim Überführen von einem Satz in einen anderen. Die identischen Wortsequenzen der Sätze haben keine Operationskosten und können daher als syntagmatische Relationen erkannt werden. Die ungleichen Wortsequenzen zwischen den identischen Teilen (engl. *slots*) gehören zu einer semantischen Klasse und werden als die Konstituenten betrachtet, die sich in paradigmatischen Relationen befinden.

Der ABL-Ansatz benötigt als Eingabe getrennte Sätze, aus denen er entsprechend markierte und eingeklammerte Sätze generiert. Dieser Ansatz besteht grundsätzlich aus zwei Schritten (van Zaanen 2000, 962):

- **Aligniertes Lernen** (engl. *Alignment Learning*):

  In dieser Phase wird ein Vergleich zwischen jedem eingegebenen Satz mit jedem anderen (engl. *all-against-all*) auf der Wortebene durchgeführt, sodass die längsten gleichen und unterschiedlichen Wortsequenzen ermittelt werden. Die unterschiedlichen Wortsequenzen, die im gleichen Kontext vorkommen, sind die potenziellen Konstituenten, die zu einer funktionalen Kategorie gehören. Die Konstituenten sind miteinander ersetzbar oder austauschbar und werden mit nichtterminalen Symbolen annotiert.

[23] Siehe Kapitel 2.1.2.2 und Bünting (1996)

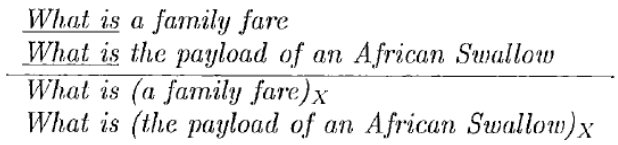


Abbildung 2.4: Beispiele des Alignierten Lernens

In der obigen Abbildung (van Zaanen 2000, 2) werden die zwei eingegebenen Sätze durch ihre identischen enthaltenden Wörter aligniert. Die Phrasen „*a family fare*“ und „*the payload of an African Swallow*“ vom jeweiligen Satz bilden ein Paradigma. Wenn eine Phrase durch die andere ersetzt wird, ergibt sich immer noch ein gültiger Satz.

- **Selektion-Lernen** (engl. *Selection Learning*):

  Dieser Schritt ist eine zusätzliche Maßnahme, um sich in den Fällen der konkurrierenden Konstituenten für den besten Kandidaten zu entscheiden. In der Phase des alignierten Lernens werden gefundene Konstituenten mit entsprechenden Symbolen annotiert, wobei bereits existierende Annotationen, die in früheren Iterationen eingefügt wurden, zunächst ignoriert werden sollen. Dies generiert konkurrierende Konstituenten innerhalb eines Satzes. Dafür werden drei Kriterien vorgeschlagen (van Zannen 2000, 964), die angeben, welche der konkurrierenden Strukturen am besten ist:

  (1) ABL:incr – geht davon aus, dass die erste Konstituente der beste Kandidat ist.

  (2) ABL:leaf – berechnet die Wahrscheinlichkeiten der konkurrierenden Konstituenten im gesamten Korpus. Es wird angenommen, dass die häufigste Konstituente die beste Wahl ist.

  (3) ABL:branch – berücksichtigt die Häufigkeiten der Wörter, die in Konstituenten erhalten sind, um die beste Entscheidung zu treffen.

Sobald eine Konstituente erkannt wird, wird sie mit einer Kategorie verschachtelt. Der Lern-Prozess wiederholt sich iterativ bis keine neuen Strukturen mehr gefunden werden können.

Ein Pseudo-Algorithmus wird aus der technischen Sicht entworfen, um den ABL-Ansatz darzustellen (van Zaanen 2000, 962):

```
For each sentence s1 in the corpus:
    For every other sentence s2 in the corpus:
        Align s1 to s2
        Find the identical and distinct parts
            between s1 and s2
        Assign non-terminals to the constituents
            (i.e. distinct parts of s1 and s2)
```

Abbildung 2.5: Alignment learning Algorithmus

Da das ABL die Konstituenten nach der Position im Kontext mit statistischen Methoden klassifiziert, muss dieser dem Umstand Rechnung tragen, dass die Konstituenten einer falschen Klasse zugeordnet werden können, insbesondere in den Sprachen, die eine relativ lockerere Wortstellung (z.B. Deutsch oder Chinesisch etc.) haben.

Eine Evaluation vom ABL-Ansatz wird an zwei Korpora (ATIS[24] und OVIS[25]) durchgeführt (van Zaanen 2000) und erreicht eine Präzision von über 80% mit einem Recall von knapp 90%. Allerdings ist dieses Verfahren zu ineffizient, um große Korpora zu verarbeiten. Van Zaanen zufolge enthalten die zur Evaluation getesteten Korpora jeweils nur 716 und 6797 Sätze.

2.2.4.2 ADIOS

ADIOS (engl. *Automatic Distillation Of Structure*) (Solan 2006) ist ein nicht-überwachter Ansatz, der versucht, Grammatiken mit Hilfe statistischer Methodik aus nicht-annotierten Korpora automatisch zu induzieren. Die Ergebnisse werden als Baumstruktur präsentiert und

[24] ATIS-Korpus basiert auf dem “Air Travel Information System” (Siehe auch Hemphill et al. 1990)

[25] OVIS (dutch. *Openbaar Vervoer Informatie Systeem*) (engl. *Public Transport Information System*) (siehe auch Bonnema et al. 1997)

beschreiben kontextfreie Grammatiken. Dieser Ansatz nutzt den MEX-Algorithmus[26] (*Tokyo Metropolitan Expressway Algorithmus*), um die identischen Teile mehrerer Sätze zu entdecken.

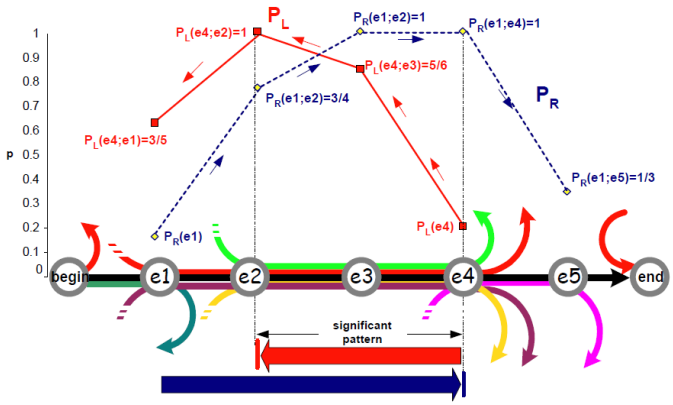


Abbildung 2.6: Kalkulation der wichtigen Patterns

Sein Algorithmus wird wie in Abbildung 2.6 (Solan 2006, 31) graphisch dargestellt und besteht grundsätzlich aus drei Schritten:

- **Initialisierung**:

  Die Sätze aus dem eingegebenen Korpus werden in Pseudographen visualisiert. Jeder Graph enthält zwei spezielle Symbole („*begin*“ und „*end*“) und Knoten, die lexikalische Einheiten bezeichnen. Jeder Satz wird durch einen Pfad des Graphen repräsentiert. In der obigen Abbildung werden die Sätze durch farbige Linien präsentiert. Die Knoten „*e1*“ bis „*e5*“ sind jeweils eine lexikalische Einheit.

- **Pattern Destillation**:

  In diesem Schritt wird der MEX-Algorithmus eingesetzt, der die Wahrscheinlichkeit für jede Einheit durch die Anzahl von ihren einkommenden und ausgehenden Pfaden berechnet. Die wichtigen Patterns tragen eine höhere Wahrscheinlichkeit und können daher als Kandidaten ausgewählt werden. Die Berechnung der Wahrscheinlichkeit wird in beide Richtungen durchgeführt. Der gemeinsame Teil zwi-

[26] Siehe Chung (1998, 7) und Solan (2006, 28)

schen den jeweiligen Spitzen „e2“ und „e4“ wird als ein wichtiges Pattern betrachtet.

- **Generalisierung**:

  Für die wichtigsten Patterns aus den im letzten Schritt gefundenen Kandidaten werden äquivalente Klassen generiert. Danach werden diese Patterns als neue Elemente in das Lexikon eingetragen. Entsprechend werden die Subpfade solcher Patterns durch einen Knoten im originalen Graphen ersetzt. So wird eine Bootstrapping-Induktion ermöglicht.

ADIOS ist in der Lage syntaktische Strukturen zu extrahieren und funktional ähnliche Konstituenten zu klassifizieren. Der MEX-Algorithmus ist dem Levenshtein-Algorithmus jedoch prinzipiell ähnlich. Der ADIOS-Ansatz ist daher hinsichtlich der Effizienz gleich dem ABL-Ansatz und eignet sich nicht in der Verarbeitung großer Korpora.

2.2.4.3 SPM

Dennis (2001) hat ein Question-Answering-System SPM (engl. *Syntagmatic-Paradigmatic Model*) entwickelt, das auf dem Instanz-Lernen basiert. Dieser Ansatz benutzt das aus nicht-annotierten Korpora gelernte syntagmatische und paradigmatische Wissen, um die eingegebenen Fragen zunächst zu analysieren und dann die entsprechenden Antworten darauf zu liefern.

Im ersten Schritt werden das durch SPM aus Rohdaten ermittelte syntagmatische und paradigmatische Wissen in drei Speicherformen als Vorlagen zur Verfügung gestellt:

- **Lexikalischer Speicher** (engl. *Lexical Memory*): lässt sich als paradigmatisches Wissen zwischen Konstituenten im gesamten Korpus definieren.
- **Syntaktischer Speicher** (engl. *Syntactic Memory*): bezeichnet syntagmatisches Wissen in jedem Satz.
- **Relationaler Speicher** (engl. *Relational Memory*): wird als das paradigmatische Wissen in jedem Satz definiert.

Corpus
Steve kicks the can.
Dave kicks the ball.
Ellen is loved by Bert.
Jody is loved by George.
Alison is loved by Steve.

Lexical Memory
Steve ⇒ Bert, Steve ⇒ George, Steve ⇒ Dave
Dave ⇒ Steve
Alison ⇒ Ellen, Alison ⇒ Jody
Ball ⇒ can
etc.

Syntactic Memory
Steve ⇒ kicks, Steve ⇒ the, kicks ⇒ the, Steve ⇒ can, kicks ⇒ can, the ⇒ can
Dave ⇒ kicks, Dave ⇒ the, kicks ⇒ the, Dave ⇒ ball, kicks ⇒ ball, the ⇒ ball
Ellen ⇒ is, Ellen ⇒ loved, is ⇒ loved, Ellen ⇒ by, is ⇒ by, loved ⇒ by, Ellen ⇒ Bert, is ⇒ Bert, loved ⇒ Bert, by ⇒ Bert
Jody ⇒ is, Jody ⇒ loved, is ⇒ loved, Jody ⇒ by, is ⇒ by, loved ⇒ by, Jody ⇒ George, is ⇒ George, loved ⇒ George, by ⇒ George
Alison ⇒ is, Alison ⇒ loved, is ⇒ loved, Alison ⇒ by, is ⇒ by, loved ⇒ by, Alison ⇒ Steve, is ⇒ Steve, loved ⇒ Steve, by ⇒ Steve

Relational Memory
Steve ⇒ Dave, can ⇒ ball
Dave ⇒ Steve, ball ⇒ can
Ellen ⇒ Jody, Ellen ⇒ Alison, Bert ⇒ George, Bert ⇒ Steve
Jody ⇒ Ellen, Jody ⇒ Alison, George ⇒ Bert, George ⇒ Steve
Alison ⇒ Ellen, Alison ⇒ Jody, Steve ⇒ Bert, Steve ⇒ George

Abbildung 2.7: Drei Speicherformen vom SPM

Die drei Speicherformen werden etwa wie in der obigen Abbildung (Dennis 2001, 3) dargestellt und dienen dazu, die aus Korpora induzierten Informationen festzuhalten.

Im nächsten Schritt verarbeitet das SPM die eingegebene Frage und sucht, wie in der folgenden Abbildung (Dennis 2001, 4) ersichtlich, in den drei Speichern die korrekte Antwort.

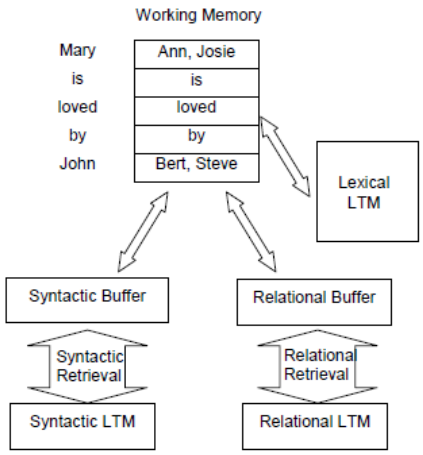


Abbildung 2.8: SP Architektur

Jedes Wort des eingegebenen Satzes wird im lexikalischen Speicher gesucht und so wird das paradigmatische Wissen im gesamten Korpus gesammelt. Das gesammelte Wissen ist daher kontextunabhängig und enthält überflüssige Informationen. Nun werden also die zur Eingabe ähnlichsten Kontexte, die als potentielle syntagmatische Vorlagen für die weitere Verarbei-

tung dienen, im syntaktischen Speicher durch SET[27] (engl. *String Edit Theory*) ermittelt. Für den Fall, dass der Eingabesatz vom syntaktischen Speicher abweicht, wird zur Weiterbearbeitung der Eingabe der relationale Speicher benötigt.

Die besondere Fähigkeit des SPM ist einerseits, dass syntagmatische Abhängigkeiten, die sich über eine lange Distanz erstrecken, korrekt und rekursiv verarbeitet werden können. Z.B. Dennis (2005, 27):

> *The man who saw the altar and gave thanks was awed.*
> *The men who saw the altar and gave thanks were awed.*

Nach dem Eingreifen des SPM kann vorausgesagt werden, dass „*was*" viel wahrscheinlicher mit „*man*" zusammen vorkommen kann und „*were*" eher zusammen mit „*men*" in einem Kontext steht. Im Gegensatz zu vorherigen Ansätzen ist SPM in der Lage syntagmatische Relationen in einem relativ großen Kontext zu entdecken.

Darüber hinaus kann SPM semantische Kategorisierungen vornehmen, die für eine statistische Untersuchung bedeutungsvoll sind. Eine der heftigsten Kritiken an statistischen Modellen lautet, dass solche Lernmodelle bei Sprachverarbeitung die systematischen Eigenschaften der natürlichen Sprachen nicht gewinnen können[28]. Fodor/Pylyshyn (1988, 18) diskutieren, dass wenn ein Mensch den Satz „*John loves Mary*" verstehen kann, er auch den Satz „*Mary loves John*" versteht. Menschen können erkennen, dass „*Mary*" sowohl die Rolle vom Subjekt als auch die Rolle vom Objekt spielen kann. Dies können statistische Modelle normalerweise nicht automatisch aus den Trainingsdaten generieren. Mit paradigmatischer Repräsentation ist SPM in der Lage diesen Nachteil zu vermeiden.

Zur Evaluation wurden 69 Artikel aus www.atptennis.com (Association of Tennis Professionals) gesammelt und manuell in Sätze segmentiert. Nach der Analyse der Texte wurde das System mit 377 Anfragen getestet und erreichte eine Präzision von 67% (Dennis 2004).

---

[27] SET ist eine EDA-ähnliche Methode, um die Strings zu alignieren. (Siehe Kruskal 1983)

[28] Siehe Foder/Pylyshyn (1988) und Marcus (1998)

2.2.5 Methoden zur Evaluation

Wie in anderen Bereichen der maschinellen Sprachverarbeitung gibt es heute verschiedene Ansätze der Grammatik-Induktion. Dabei gilt es die Frage zu beantworten, welches System am besten ist. Dazu müssen die GI-Systeme evaluiert werden. Die Evaluation ist nicht trivial. Sie ist vielmehr eine der schwierigsten Aufgaben in der Verarbeitung der natürlichen Sprachen (engl. *Natural Language Processing*, NLP). Die Schwierigkeit liegt darin, dass keine einheitliche, korrekte Ausgabe für unterschiedliche Systeme existiert (Atwell/Demetriou/Hughes 2000; Christodoulopoulos/Godwater /Steedman 2010). Beispielsweise extrahiert jedes GI-System seine definierten Grammatiken in der eigenen Darstellungsform. Ein genauer Vergleich zwischen den Systemen ist daher oft unmöglich.

Die generelle Vorgehensweise der Algorithmen zur Evaluation der GI kann durch die Eingabe von unstrukturierten Daten und die Ausgabe der Evaluation dargestellt werden. In der Literatur werden verschiedene Evaluierungsmethoden vorgeschlagen, die grundsätzlich auf drei Strategien beruhen[29]:

- **Methode Looks-Good-to-me** (LGtm):

  Das zu evaluierende System lernt an einem kleinen Stück von unstrukturiertem Text. Danach bewertet ein Evaluator (meistens ein linguistischer Experte) mit seiner linguistischen Intuition die Qualität der resultierten Grammatik manuell. Diese Methode war sehr beliebt wegen ihrer offensichtlichen Einfachheit. Weil die Evaluierung in unterschiedlichen Sprachen nur nicht strukturierte (rohe) Korpora benötigt, ist sie sehr effizient (van Zaanen 2001). Diese Methode ist wegen der manuellen Bewertung jedoch sehr subjektiv. Allerdings kann die Evaluation vertrauenswürdig sein, wenn die Ergebnisse durch mehrere unabhängige linguistische Experten bestätigt werden. LGtm ist in dem Sinn, dass eine mathematische oder statistische Messung nicht nötig ist, eine wichtige Methode zur Evaluierung der GI-Systeme.

  Die Nachteile dieser Methode sind dennoch nicht zu vernachlässigen. Eine solche Evaluierung mit Hilfe menschlicher Intuition fordert spezielle Kenntnisse und Erfahrungen

[29] Vgl. van Zaanen (2001) und D'Ulizia et al. (2011)

des Systems, über welche die Forscher aus anderen Bereichen nicht immer verfügen. Andererseits wird die Evaluierung meist durch den Entwickler des Systems anstatt durch einen unabhängigen Experten durchgeführt (D'Ulizia et al. 2011, 19). Dies führt oft zu einer voreingenommenen Schlussfolgerung. Schließlich kann diese Methode keinen statistischen Test für eine präzise Messung der Systemperformanz durführen.

- **Methode Gold-Standard Treebank** (GST)

  Diese Methode extrahiert die originalen Sätze aus einer existierenden Treebank, die als „Gold-Standard" betrachtet wird. Das zu evaluierende System nimmt wiederum solche Sätze als Eingabe an und produziert daraus neue Sätze. Mit einem Vergleich zwischen den produzierten Sätzen und den originalen, strukturierten Sätzen aus der Treebank wird die Systemperformanz evaluiert. Dabei werden zwei übliche statische Parameter[30] (z.B. Präzision, Recall und F1-Wert etc.) verwendet, womit ein deutliches und objektives Ergebnis der Systemperformanz kalkuliert werden kann.

  Die Anwendung dieser Methode in der Praxis ist jedoch eingeschränkt. Da der Aufbau einer Treebank sehr viele Ressourcen (Zeit und Geld) kostet, sind zurzeit noch keine ausreichenden Baumbanken verfügbar. Ansonsten ist die Qualität und Gültigkeit der Annotation der Treebank nicht zwangsläufig befriedigend, obwohl sie als „Gold-Standard" bezeichnet werden (Roberts/Atwell 2003, 2). Wenn eine durch das GI-System generierte Grammatik den Strukturen in den Treebanks unähnlich ist, muss sie nicht unbedingt falsch sein, sondern evtl. äquivalent.

- **Methode Rebuilding Known Grammars** (RKG)

  Diese Methode enthält eine vordefinierte (einfache) Grammatik, die eine Menge von Beispielsätzen generiert. Das GI-System erhält die Beispiele als Eingabe und produziert darüber hinaus eine neue Grammatik, die mit der originalen Grammatik manuell verglichen wird. Falls die beiden Grammatiken ähnlich oder äquivalent sind, wird dieses Lernsystem als „gut" bezeichnet.

[30] Definitionen siehe Kapitel 4.1

Die RKG-Methode besitzt ähnliche Vorteile wie die GSTb-Methode. Zum einen ist ein Experte während der Evaluierung nicht mehr notwendig. Das heißt, die Methode ist auch für die Forscher ohne spezielle Kenntnisse zugänglich. Zum anderen ist der Prozess teilweise automatisiert, daher ist das Ergebnis viel objektiver im Vergleich zur LGtm-Methode.

Allerdings beeinflusst auch die Wahl der Grammatiken die Evaluierung (D'Ulizia/Ferri/Grifoni 2011, 19). Ein anderer Nachteil ist, dass nur kleine einfache Grammatiken getestet werden können (van Zaanen/Geertzen, 2008, 302).

Scheinbar ist eine perfekte und einheitliche Problemlösung zur zuverlässigen (manuellen oder automatischen) Evaluierung für alle Ansätze nicht realisierbar. In Bezug auf das Ziel dieser Arbeit steht der Effizienzvergleich bei der Evaluation an erster Stelle. Weil die wichtigen unüberwachten statistischen GI-Systeme dem ABL-Ansatz ähnlich sind, findet sich hinsichtlich der Effizienz kein gravierender Unterschied. Daher soll im Folgenden der ABL-Ansatz mit dem hier entwickelten Ansatz verglichen werden. Um einen fairen, objektiven und vertrauenswürdigen Vergleich zu gewährleisten, werden die Ergebnisse der zwei Ansätze quantitativ (durch die Messungsmaße: Recall, Präzision, F-Wert und Laufzeit etc.) analysiert und dargestellt.

2.2.6 Fazit

Zahlreiche Methoden oder Ansätze werden zur Grammatik-Inferenz entwickelt, um die Grammatiken aus endlichen Beispielen einer natürlichen Sprache zu extrahieren. Die verschiedenen für diese Arbeit relevanten Lernmodelle der Grammatik-Induktion werden im weiteren Verlauf vorgestellt.

In letzten Jahren ist die Entwickelung der statistischen oder lernbasierten Methoden ein Schwerpunkt in diesem Forschungsfeld geworden. Der größte Vorzug einer solchen Methode ist, dass die arbeitsaufwendige Erstellung der Grammatiken durch einen automatisierten effizienten Prozess gespart werden kann.

Die vorhandenen klassischen Ansätze (z.B. ABL, der in dieser Arbeit der referenzierte Ansatz ist.) sind nicht effizient genug, um große Korpora zu verarbeiten. Daher wird der Einsatz solcher Ansätze in der Praxis eingeschränkt.

Um diesen Nachteil zu vermeiden wird ein effizienter Ansatz – EDSI – entwickelt und quantitativ evaluiert. Durch einen Vergleich mit dem referenzierten Ansatz ABL wird die Effizienzerhöhung deutlich dargestellt.

2.3 Lokale Grammatiken

2.3.1 Einleitung

Bisher wurden viele Ansätze entworfen oder realisiert, die versuchen, umfassende und stark verallgemeinerte Grammatiken zu entwickeln. Beispielsweise beschäftigt sich das Markovian-Model mit grammatischen Kategorien, die die Sprachen beschreiben. Dieses Modell bildet einen Satz durch eine Sequenz von grammatischen Kategorien ab, die von den entsprechenden Wörtern ersetzt werden. Dieser Ansatz kann positionelle Features (z.B.: Artikel oder Adjektive auf der linken Seite von Normen) auffinden. Auf einem verfeinerten Niveau hat Chomsky (1957) solche Grammatiken unter den Namen „kontextfreie Grammatiken“[31] zusammengefasst. Aber alle diese Ansätze führen die Ergebnisse zu einem Übergenerierungsproblem und müssen viele Ausnahmen ausschießen. Die entwickelten Grammatiken sind daher wenig befriedigend.

Tendenziell gewinnen Korpus-basierte Ansätze immer mehr Aufmerksamkeit. Die Grammatiken werden nicht mehr von Linguisten „ausgedacht“, sondern durch die induktiven Ansätze aus vorhandenen Texten extrahiert. Durch die Analyse der syntaktischen und/oder semantischen Phänomene auf realen Korpora entstehen die Regeln zur Beschreibung der natürlichen Sprachen. Dies ist auch einer der grundlegenden Gedanken von lokalen Grammatiken.

[31] Siehe Kapitel 2.2

Lokale Grammatiken wurden von Maurice Gross (1993) in LADL[32] zur Repräsentation von Kollokationen und festen Redewendungen (Idiomen) vorgestellt. Statt die Sprachen global zu analysieren, fangen lokale Grammatiken zunächst mit der Analyse der lokalen syntaktischen Phänomene an, die in unmittelbarer Nachbarschaft eines Wortes oder einer Wortklasse in einem Satz auftreten. Basierend auf den lokalen Beschreibungen können größere syntaktische Strukturen präsentiert werden. Das heißt, lokale Grammatiken beschreiben einerseits die Wörterklassen, und andererseits syntaktische Strukturen.

> *"In the study of collocation and of frozen sentences (idioms, cliches, collocations, many metaphors and figurative meanings, etc..) one often encounters sets of similar forms that cannot be related by formal rules of either type: phrase structure or transformational. […] the formalism of finite automata can be used to represent them in a natural way."* (Gross 1993, 26)

> *"[…], the global nature of language results from the interaction of a multiplicity of local finite-state schemes which we call finite-state local automata.* (Gross 1997, 330)

> *[…] This is somewhat similar to the way small molecules combine to produce much larger ones in organic chemistry."* (Gross 1997, 331)

> *"Local grammars are finite-state grammars or finite-state automata that represent sets of utterances of a natural language[33]."* (Gross 1999, 229)

Lokale Grammatiken sind eine Menge von manuell erstellten Grammatikregeln und gehören im Prinzip zu den regelbasierten Methoden. Wie in anderen Grammatiktheorien (z.B. Dependenz- bzw. Valenzgrammatik[34]) werden lexikalische Ressourcen, die umfangreiche Notationen enthalten, oft an die Grammatikregeln geknüpft.

---

[32] Laboratoire d'automatique documentaire et linguistique, das von Maurice Gross gegründete Labor in Paris.

[33] Lokale Grammatiken, endliche Automaten und reguläre Grammatiken werden bei Gross als Synonyme betrachtet.

[34] Siehe Tarvainen (1981)

2.3.2 Definition

2.3.2.1 Informelle Beschreibung

Jeder Pfad einer lokalen Grammatik ist ein linearer endlicher Automat. Eine eingegebene Sequenz wird von ihrem Anfang bis zum Ende inkrementell erschlossen. Eine Grammatik besitzt nur einen Anfangszustand, der anstatt einer Wortklasse lediglich den Start aller Sätze symbolisiert. Ein einziger Endzustand, wie der Anfangszustand, bezeichnet das Ende der Grammatik. Während der Analyse werden für jeden Zustand alle Verzweigungspfade parallel und unabhängig voneinander gesucht. Ist die Suche für einen Pfad erfolgreich, wird ein neuer Zustand produziert. Diese Analyse wird so lange fortgesetzt, bis entweder keine neuen Zustände erzeugt werden können, oder die Sequenz komplett eingelesen wurde. Falls ein oder mehrere Endzustände in einem zweiten Fall erreicht werden, wurden vollständige grammatische Sequenzen gefunden. Ansonsten liegt eine unvollständige Sequenz vor.

2.3.2.2 Formale Definition lokaler Grammatiken

Eine formale Definition für lokale Grammatiken ist bis heute noch nicht in der Literatur zu finden. Hier soll versucht werden, eine formale Definition zu erfassen.

> **Lokale Grammatiken** (engl. *Local Grammars*, LGs) können durch ein 5-Tupel $A=(Q, P, \Sigma, \delta, s, f)$ definiert werden:
>
> Q ist eine endliche Menge von Zuständen;
>
> P ist eine endliche Menge von Subgraphen, $P \subseteq Q$;
>
> Σ ist ein Alphabet der Eingabesymbole;
>
> δ ist eine vollständige Übergangsfunktion zwischen den Zuständen. $\delta: Q \times \Sigma \rightarrow Q$;
>
> s ist der Startzustand, $s \notin Q$;
>
> f ist der Endzustand, $f \notin Q$;

Die Sprache, die von LG akzeptiert wird, wird als L(*A*) bezeichnet.

2.3.3 Repräsentation lokaler Grammatiken

Lokale Grammatiken sind eigentlich endliche Automaten, die komplexe linguistische Strukturen beschreiben können. Außerdem können lokale Grammatiken auch als Transduktoren repräsentiert werden, die zusätzlich eine Ausgabefunktion enthalten. Der Ausgabeteil dient dazu, einerseits den erkannten Text in eine andere Sprache oder eine kanonische Form zu übersetzen (Nagel 2004) und andererseits die erkannten Sequenzen mit linguistischen Informationen zu markieren.

2.3.3.1 Chomsky-Hierarchie

Lokale Grammatiken sind auch formale Grammatiken, die einerseits grundliegende Eigenschaften endlicher Automaten besitzen, und andererseits kontextfreie Sprachen beschreiben können. Daher ist es notwendig die Chomsky-Hierarchie zunächst vorzustellen.

Syntaktische Satzstrukturen der natürlichen Sprachen werden typischerweise durch grammatische Regeln repräsentiert, die dazu dienen, einerseits alle wohl-geformten (grammatisch korrekten) Sätze in der Sprache abzudecken und anderseits die nicht zu dieser Sprache gehörenden Sätze auszuschließen. Durch formale Sprachen können natürliche Sprachen modelliert werden, damit der Lernprozess besser zu verstehen ist (Chomsky 1956). Diese Modellierung ermöglicht außerdem die automatische Induktion der Grammatiken. Ein Vorzug formaler Sprachen ist, dass sie mathematisch exakt analysiert werden können. Ein anderer Vorteil ist, dass unendliche grammatische Sätze der Sprachen ohne Aufzählen dargestellt werden können. Um sie exakt zu beschreiben, sind formale Sprachen nötig, die eine unendliche Menge definieren können.

Die Chomsky Hierarchie ist nach dem Linguisten Noam Chomsky benannt und enthält vier Klassen[35] formaler Grammatiken. Die Klassen werden wie folgt definiert:

- **Typ-0 Grammatiken** (nicht eingeschränkte Grammatiken).

  Diese Grammatiken sind die allgemeinste Form der vier Grammatiktypen. Sie erlauben beliebige Produktionen und beschreiben aufzählbare Sprachen.

  Eine Sprache, die durch nicht eingeschränkte Grammatiken erzeugt wird, kann von einer Turing-Maschine akzeptiert werden und umgekehrt. D.h. wenn eine Sprache von einer Turing-Maschine erkannt wird, kann sie auch durch nicht eingeschränkte Grammatiken generiert werden.

  Da eine effiziente Implementierung dieser Grammatiken bis heute noch nicht machbar ist, sind sie in der Praxis nicht sehr relevant.

- **Typ-1 Grammatiken** (kontextsensitive Grammatik).

  Diese Grammatiken sind eine Teilmenge von Tpy-0 Grammatiken. Sie beschränken die Form möglicher Produktionen. Eine Erzeugung ist von den linken und rechten Kontexten des zu ersetzenden Symbols abhängig.

  Kontextsensitive Grammatiken erzeugen kontextsensitive Sprachen, die von einer nichtdeterministische Turing-Maschine erkannt werden können.

- **Typ-2 Grammatiken** (kontextfreie Grammatik).

  Diese Grammatiken sind eine Teilmenge von Typ-1 Grammatiken. Sie beschränken auch die Form möglicher Produktionen. Allerdings ist die Erzeugung nicht mehr von Kontexten abhängig.

  Kontextfreie Grammatiken erzeugen kontextfreie Sprachen, die von einem nichtdeterministischen Kellerautomaten erkannt werden können.

[35] Formale Sprachen können nach ihren Automatentypen, die die entsprechenden Sprachen erkennen können, klassifiziert werden. Das heißt, für jede Sprachklasse existiert ein bestimmter Automat, der identifizieren kann, ob die Sätze zu dieser Sprache gehören.

- **Typ-3 Grammatiken** (reguläre Grammatik).

  Diese Grammatiken sind eine Teilmenge von Typ-2 Grammatiken. Eine Produktion ist nur möglich, wenn die rechte Seite allein ein Terminalsymbol, bzw. ein Terminalsymbol und ein Nichtterminalsymbol enthält.

  Reguläre Grammatiken erzeugen reguläre Sprachen, die von einem endlichen Automaten erkannt werden können.

Die oben dargestellten Grammatiken unterscheiden sich in ihrer Erzeugungsmächtigkeit und werden nach der Mächtigkeit absteigend aufgelistet.

(1) Reguläre Grammatiken & Endliche Automaten

- **Reguläre Grammatiken** (engl. *regular grammars*) bilden die einfachste Grammatikklasse in der Hierarchie und können durch endliche Automaten (engl. *finite state automata*) erkannt werden. Reguläre Grammatiken sind eine Teilmenge von kontextfreien Grammatiken. Auf der rechten Seite darf genau ein Terminalsymbol mit maximal einem weiteren Nichtterminalsymbol auftreten.

  Wenn die Produktionen der Grammatiken wie folgt:

  (1) $A \rightarrow \omega B$ oder $A \rightarrow \omega$

  (2) $A \rightarrow B \omega$ oder $A \rightarrow \omega$

  formuliert werden können, wobei A und B Variablen und ω eine Zeichenkette (darf leer sein) von Terminalen ist, ist die Grammatik (1) rechtslinear und die Grammatik (2) linkslinear. Die zwei Regeln werden reguläre Grammatiken genannt.

- **Endliche Automaten** (engl. *finite-state automaton*, FSA) entsprechen in ihrer Komplexität regulären Grammatiken und können durch 5-Tupel $A=(Q, \Sigma, \delta, q_0, F)$ definiert werden.

  Q ist eine endliche Menge von Zuständen;

  Σ ist ein Alphabet der Eingabesymbole;

δ ist eine vollständige Übergangsfunktion zwischen den Zuständen. $\delta: Q \times \Sigma \rightarrow Q$.

q_0 ist der Startzustand $q_0 \in Q$;

F ist eine Menge von Endzuständen, $F \subseteq Q$.

Die Sprache, die von DFSA A akzeptiert wird, wird als $L(A)$ bezeichnet.

Ein angepasster Graph kann mit dem entsprechenden endlichen Automaten assoziiert werden. Die Knoten im Graphen bezeichnen die Zustände des endlichen Automaten, während ein Pfeil den Übergang von dem vorherigen Zustand zum nachherigen Zustand symbolisiert. Neben der Flexibilität sind endliche Automaten auch sehr mächtig und effizient zur Repräsentation linguistischer Beschreibung (Roche/Schabes 1996).

(2) Kontextfreie Grammatiken & Kellerautomaten

Allerdings ist die Repräsentationsfähigkeit regulärer Grammatiken eingeschränkt, da die Einbettung für diese Grammatiken schwach adäquat ist. Dieser Nachteil kann durch kontextfreie Grammatiken, die mit Kellerautomaten erkannt werden, vermieden werden.

- **Kontextfreie Grammatiken** (engl. *context-free grammars*, CFG) können mit einem Quadrupel: $G = (N, \Sigma, P, S)$ definiert werden.

  N ist ein Alphabet von nicht-terminalen Symbolen;

  Σ ist ein Alphabet von terminalen Symbolen, wobei $N \cap \Sigma = \emptyset$;

  P ist eine endliche Menge von Produktionsregeln in Form $A \rightarrow \alpha$, wobei $A \in N$ und $\alpha \in (N \cup \Sigma)^*$.

  $\boldsymbol{S}$ ist ein besonderes Element, ein Startsymbol.

  Die Sprache, die von CFGs erzeugt werden, wird mit $L(G)$ bezeichnet. Die linke Seite einer kontextfreien Grammatik besteht nur aus einzelnen nicht-terminalen Symbolen, die immer unabhängig vom Kontext sind. Die Sätze der entsprechenden Sprache werden ausgehend von einem Startsymbol durch iterierte Anwendung der Grammatikregeln

erzeugt. Nicht-terminale Symbole, die sich auf der linken Seite befinden, werden ersetzt, bis die erzeugten Wörter nur noch aus Terminalsymbolen bestehen.

- **Kellerautomaten** (engl. push-down automata, PDA) sind der entsprechende Automatentyp zu kontextfreien Grammatiken und können folgendermaßen $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ definiert werden:

  Q ist eine endliche Menge von Zuständen;

  Σ ist das Alphabet der Eingabesymbole;

  Γ ist die Menge der Kellersymbole;

  δ ist eine vollständige Übergangsfunktion zwischen Zuständen $Q \times \Sigma \times (\Gamma \cup \alpha) \rightarrow Q \times (\Gamma \cup \alpha)$, wobei α den leeren Keller bezeichnet.

  q_0 ist der Anfangszustand, wobei $q_0 \in Q$;

  Z_0 ist das Anfangssymbol, wobei $Z_0 \in \Gamma$;

  F ist eine Menge von Endzuständen, wobei $F \subseteq Q$;

Es bestehen zwei äquivalente Möglichkeiten für Kellerautomaten, eine Sprache zu akzeptieren. Die erste Vorgehensweise gleicht den endlichen Automaten; eine Eingabe wird durch den Übergang in einen Endzustand akzeptiert. Als Alternative kann die Eingabe durch einen leeren Keller akzeptiert werden.

(3) RTN

In der Praxis werden die letzten drei Grammatiktypen (überwiegend reguläre Grammatiken und kontextfreie Grammatiken) zur Beschreibung der natürlichen und formalen Sprachen eingesetzt. Über die Entscheidung, welche formale Grammatik zur Beschreibung der natürlichen Sprachen angemessen oder ausreichend ist, wird von Anfang an diskutiert. Endliche Automaten sind verbreiteter als kontextfreie Grammatiken. Grishmann (1995, 167) zufolge sind CFGs nicht nur viel zu langsam, um in praktisch anwendbarer Zeit die Ergebnisse zu liefern oder einen Test durchzuführen. Sondern die Erweiterung oder Veränderung der Systembasis für CFGs ist auch sehr aufwändig.

Endliche Automaten sind der erzeugungsmächtigste Grammatiktyp von den vier formalen Grammatiken. Mehrere Studien zeigen, dass endliche Automaten ausreichend sind, zahlreiche Sätze der natürlichen Sprachen zu beschreiben. Sogar in einer kontextfreien oder kontextsensitiven natürlichen Sprache kann es sinnvoll sein, wesentliche Teile der Sprache mit regulären Grammatiken zu beschreiben (Mohri/Sproat 2006).

Darüber hinaus werden zur Beschreibung kontextfreier Grammatiken rekursiver Übergangsnetzwerke als ein alternativer Automatenformalismus zu Kellerautomaten vorgeschlagen. Obwohl die RTNs nicht den aussagekräftigsten Formalismus besitzen, werden sie wegen ihrer Einfachheit für zahlreiche praktische Anwendungen eingesetzt.

- **Ein einfaches Übergangsnetzwerk** (engl. *Transition Networks*, TNs) ist ein gerichteter Graph, der aus genau einem Startknoten, mehreren (mindestens einem) terminalen Knoten und mehreren (mindestens einem) markierten Kanten besteht.

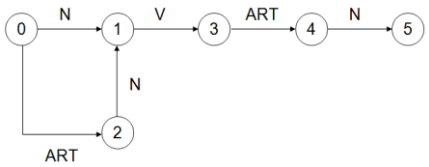


Abbildung 2.9: Einfaches Übergangsnetzwerk (TN)

  Wie in der obigen Abbildung dargestellt, akzeptiert dieses TN solche Sätze wie „*Peter schreibt einen Text*." oder „*Ein Student schreibt einen Text*." Die Sätze werden durch den Graphen visualisiert.

- **Ein rekursives Übergangsnetzwerk**[36] (engl. *Recursive Transition Networks*, RTNs) kann die Kanten enthalten, die selbst Übergangsnetzwerke sind. Das heißt, die markierten Kanten dürfen nicht nur terminale Symbole sein, sondern auch beliebige nichtterminale Symbole (wie im folgenden Beispiel: NP).

[36] Siehe Woods (1970)

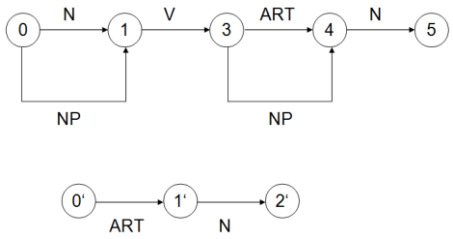


Abbildung 2.10: Rekursive Übergangsnetzwerke (RNT)

Beispielsweise wird ein Subgraph für „NP“ erstellt, der in den originalen Graphen eingebunden werden kann. Diese Einbindung kann beliebig wiederholt werden. RTNs sind nicht-deterministisch. Das heißt, alle Verzweigungsmöglichkeiten müssen mit einem Rücksetzverfahren berücksichtigt werden. Normalerweise besteht ein RTN aus mehreren endlichen Automaten, die kaskadierend aufgerufen werden können, und einem Keller, der alle Zustände speichern kann[37].

RTNs sind grundsätzlich eine Erweiterung endlicher Automaten, die in einer intuitiven und leicht erfassbaren graphischen Form dargestellt werden können. Es besteht für sie auch die Möglichkeit, die Grammatiken zu modularisieren (Gross 1997). Die immer wieder verwendeten Graphen können als Module gespeichert werden. Durch eine Verknüpfung der Module (Subgraphen) kann leicht eine komplexe Struktur erstellt und übersichtlich dargestellt werden.

Durch den Einsatz der RTNs präsentieren sich die Vorteile lokaler Grammatiken deutlich:

- Der Formalismus ist sehr einfach. Daher ist die Wartung für lokale Grammatiken ziemlich leicht. Um z.B. eine Grammatik zu erweitern, ist ein Einfügen des neuen Pfades in den Graphen ausreichend.
- Lokale Grammatiken sind präzise definierte Automaten und können daher sehr genaue Erkenntnisse liefern.
- Durch graphische Darstellungen sind die Grammatiken einerseits sehr verständlich – auch für Benutzer ohne spezielle Kenntnisse des Systems. Anderseits er-

[37] Siehe Winograd 1983

möglicht diese Darstellungsform die Zusammenarbeit zwischen unterschiedlichen Grammatikentwicklern. Da die Grammatikentwicklung der natürlichen Sprachen eine komplexe und aufwendige Aufgabe ist, ist die Möglichkeit der Zusammenarbeit sehr vorteilhaft.

- Die Modularisierung durch RTNs ist das wichtigste Feature lokaler Grammatiken. Sie bietet einige Vorzüge. Erstens, besteht die Möglichkeit, lokale Grammatiken inkrementell zu entwickeln. Zweitens können die Module einander aufrufen, kaskadiert ausgeführt werden und so komplexe Strukturen beschreiben. Somit ist die Erweiterungs- und Wartungsarbeit, im Vergleich zu in anderen Formalismen geschriebenen Grammatiken, viel leichter.
- Endliche Automaten sind in der Regel besonders effizient.

Allerdings werden üblicherweise die Regeln lokaler Grammatiken vereinzelt manuell erstellt. Obwohl die Entwicklung teilweise automatisiert werden kann, ist die Erstellung immer noch sehr aufwändig. Sobald jedoch die Grammatiken vorhanden sind, können sie auch in anderen Korpora wiederverwendet werden.

2.3.3.2 Lesbarkeit der Grammatikregeln & Graphen

Ein Vorteil regelbasierter Systeme im Vergleich zu statistischen Systemen ist die Transparenz und Lesbarkeit der Grammatikregeln. Die Repräsentationsform der induzierten Grammatiken ist vom Aufgabenziel abhängig, muss aber auch die Benutzerfreundlichkeit berücksichtigen. Eine zunehmende Vielzahl der induzierten komplexen Grammatiken führt oft zu einer Unübersichtlichkeit und Unverständlichkeit. Die Grammatiken werden irgendwann allein für den Entwickler selbst lesbar und schwierig zu erweitern oder zu verändern sein.

Bei der Wahl der Beschreibungsmittel stehen zwei zentrale Aspekte zur Grammatikentwicklung im Vordergrund. Eine geeignete Darstellungsform der Grammatikregeln muss in der Lage sein, die relevanten Informationen deklarativ vorzulegen, sodass einerseits die spätere Anwendung der Grammatiken ermöglicht und anderseits die Transparenz sowie die einfache Wartbarkeit der Grammatiken gewährleistet wird.

Graphentransformation der Grammatiken

Eine graphische Darstellung mit deklarativen und transparenten Notationen ist sehr benutzerfreundlich und ist sogar für einen Benutzer ohne linguistische Kenntnisse lesbar und verständlich. Die in dieser Arbeit induzierten Grammatiken sind eigentlich endliche Automaten, die üblicherweise als Graphen dargestellt werden. Die graphisch dargestellten Automaten enthalten generell folgende Bestandteile:

- **Startsymbol** ist ein besonderes Element, das das allgemeinste Konstrukt einer Sprache darstellt.
- **Terminalsymbole** sind Zeichen oder Gruppen von Zeichen. Beispielsweise bestehen die Wörter einer Sprache aus Terminalsymbolen.
- **Nichtterminalsymbole** sind Symbole, die syntaktische Klassen (z.B. Satz, Subjekt oder Objekt, etc.) bezeichnen. Solche Symbole können wiederum in andere Nichtterminalsymbole oder Terminalsymbole zerlegt werden und bilden eine Metasprache der syntaktischen Konstruktion ab.
- **Produktionsregeln** sind die Erzeugungsvorschriften, die beschreiben, wie neue Sätze einer Sprache aus den existierenden Elementen gebildet werden können.

Wenn eine Sequenz durch einen Pfad des Automaten verarbeitet werden kann, dann ist sie ein korrekter Satz, der zu der beschreibenden Sprache gehört.

Ein endlicher Graph $G = (V, E)$ enthält 2-Tupel: eine endliche Menge von Knoten (engl. *nodes* oder *vertices*) V und eine Menge von Kanten, wobei die Knoten durch die Kanten paarweise verbunden werden.

Jeder Knoten symbolisiert einen Zustand und erhält entsprechend deklarative Informationen (z.B. lexikalische Information). Jede Kante dient als Übergang vom vorherigen Zustand zum nächsten. In dieser Untersuchung handelt es sich nur um gerichtete Graphen, die gerichtete Kanten enthalten. Das heißt, die Kanten gehen von links nach rechts und vom vorherigen

Knoten zum nachherigen Knoten. Solche Graphen werden als gerichtete Azyklische Graphen (engl. *Directed Acyclic Graphs,* DAGs) bezeichnet.

2.3.3.3 Graphische Darstellung lokaler Grammatiken

Wie in den obigen Auswahlkriterien der Repräsentationsform definiert, sollen sprachspezifische linguistische Informationen in einem transparenten und deklarativen Format repräsentiert werden. Die Repräsentationsform soll einerseits für die zukünftige Datenverarbeitung effizient einsetzbar sein, und andererseits für Benutzer ohne spezielle linguistische Kenntnisse des Systems lesbar und verständlich sein.

Grundsätzlich kann ein Automat als ein gerichteter Graph betrachtet werden. Lokale Grammatiken werden üblicherweise in Form von Graphen wie RTNs (siehe Kapitel 2.3.2.1) dargestellt. Damit können die folgenden Vorteile dieser Repräsentationsform genutzt werden:

- Die graphische Darstellung ist deutlich übersichtlicher und verständlicher (siehe Abbildung 2.12);
- Als endlicher Automaten ist die Durchsuchung viel effizienter als reguläre Ausdrücke (Gross 1993, 28).

Außerdem können die Graphen modularisiert werden. Dies ermöglicht es, die Graphen als Subgraphen miteinander zu verbinden oder als Knoten in einen komplexen Graphen zu integrieren. Das heißt, wenige Module können mit verknüpften Pfaden viele variierende Einträge repräsentieren.

Ein Graph lokaler Grammatiken besteht normalerweise aus einem Anfangszustand, mindestens einem Rechtspfeil, mehreren Knoten (oder keinen) und schließlich einem Endzustand. Damit repräsentiert jeder Pfad einen Satz. Weil solche endlichen Graphen oder Automaten auf endliche Pfade beschränkt werden, werden sie als gerichtete azyklische Graphen (engl. *Directed Acyclic Graphs,* DAGs) bezeichnet.

- **Knoten** symbolisiert einen Zustand und bedeutet eine Wortklasse oder Phrasenklasse, die funktional ähnliche Wortsequenzen enthält. Wobei das leere Wort durch „<E>“ im Knoten markiert wird.

- **Rechtspfeil** symbolisiert keinen Zustand sondern einen Übergang vom vorherigen Zustand zum nächsten Zustand. Wie die gewöhnliche Leserichtung der meisten Sprachen wird die Grammatik von links nach rechts interpretiert, um die Wortsequenzen im Text zu identifizieren.
- **Anfangszustand** wird durch einen Rechtspfeil symbolisiert und bezeichnet keinen echten Zustand. Er führt lediglich zu dem nächsten Zustand im Graphen.
- **Endzustand** symbolisiert wie der Anfangszustand allein das Ende einer Grammatik und wird durch einen doppelt umrandeten Kreis bezeichnet.

Die in dem Graph verknüpften Subgraphen werden als Knoten markiert und beim Ausführen des Graphen aufgerufen.

Ein einfaches Beispiel

Die lokale Grammatik in der unteren Abbildung (Gross 1999, 249) besteht aus 24 endlichen Automaten, die 24 mögliche Pfade beschreiben und gleichzeitig sowohl syntagmatische als auch paradigmatische Relationen in Bezug auf den Kontext „*HealthAndN*“ bilden.

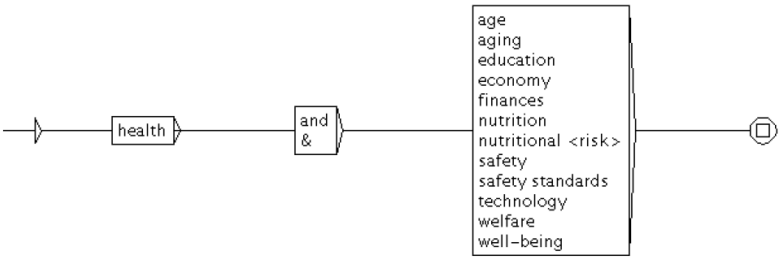


Abbildung 2.11: Graph „HealthAndN“

Die in einer Box aufgelisteten Wörter oder Phrasen sind „funktionale Synonyme“ und gehören zu einer semantischen Klasse. Das heißt, eine Ersetzung zwischen den Wörtern aus derselben Box verändert zwar wahrscheinlich die semantische Bedeutung (z.B. zwischen *age* und *education*), verletzt aber nicht die Grammatikalität der Sätze. Im Gegensatz zu lokalen Grammatiken kann eine Ersetzung aus traditionellen Wortarten (z.B. Nomen) die Grammatikalität nicht gewährleisten. Daher unterscheiden sich lokale Grammatiken durch deklarative

Beschreibung von den traditionellen Grammatiken. Deklarative Information verspricht zwar eine hohe Qualität der Verarbeitung, produziert aber eine Vielzahl von Grammatikregeln, die meistens einen effizienten Algorithmus in Anspruch nimmt.

Der Graph kann leicht in einen Transduktor umgeschrieben werden, der in der Lage ist, den eingegebenen Text zu erkennen und dem Aufgabenzweck nach zu verändern. Beispielsweise kann eine erkannte Sequenz in eine andere Sprache oder in eine kanonische Form übersetzt werden.

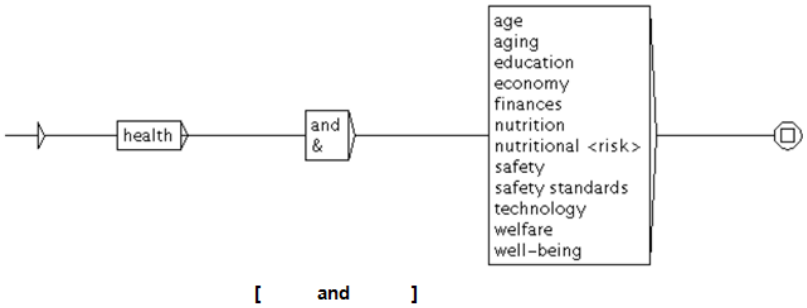


Abbildung 2.12: Transduktor – kanonische Form für „*and*“ und „*&*“

Der Graph in der obigen Abbildung ist ein Typ der Transduktoren. Das Sonderzeichen „*&*“ in allen erkannten Sequenzen wird durch die kanonische Form „*and*“ ersetzt.

Ein komplexes Beispiel

Für das Schlüsselwort kann außer dem obigen einfachen Beispiel auch eine komplexe Struktur aufgebaut werden. Beispielsweise werden die ökonomischen Kontexte in den Graphen „*HealthEco*“ (Gross1999, 245) zusammengefasst (siehe Anhang 7.3.1).

Lokale Grammatiken bestehen normalerweise nicht aus einem einzigen Graphen, sondern aus einer Menge von Graphen. Ein einzelner Graph kann in anderen Graphen eingebunden werden. Daher enthält ein großer komplexer Graph wahrscheinlich mehrere Subgraphen. Die Grammatiken „*HealthAndN*“ und „*HealthEco*“ können entweder als einzelner Graph verwendet werden oder als Subgraphen mit anderen Graphen eine komplexe Grammatik „*HEALTH*“ (Gross 1999, 249) wie im Anhang 7.3.1 darstellen.

2.3.4 Implementierung lokaler Grammatiken

2.3.4.1 Vorgehensweise: Bootstrapping

Zur Erstelllung von lokalen Grammatiken wurde eine „**Bootstrapping**"-Methode von Gross (1999) vorgeschlagen, sodass die Grammatiken mit Hilfe von Beispielen aus den Korpora manuell herangezogen werden können. Der Ansatz läuft wie folgt ab:

(1) Zunächst müssen die Schlüsselbegriffe ausgewählt werden, die von dem Aufgabenziel abhängig sind. Beispielsweise wird das englische Wort „*health*" zur Erstellung der lokalen Grammatiken (z.B. „*healthAndN*"), die dieses Wort enthalten, als Schlüsselbegriff untersucht.

(2) Alle Konkordanzen des Schlüsselbegriffs in dem eingegebenen Text (oder einem Korpus) werden als Kandidaten ermittelt. Die Konkordanzen können entweder nach links oder nach rechts sortiert werden, je nachdem welches Aufgabenziel vorliegt. In dem Beispiel „*healthAndN*" sind die Konkordanzen auf der rechten Seite für die Erstellung der Grammatik hilfreich.

(3) In Schritt (2) werden auch viele andere Schemata erkannt. Die gefundenen Konkordanzen sollen mit der Intuition des Linguisten vereinzelt ausgewertet werden. Ausgehend von den gültigen Konkordanzen werden die Grammatiken und die Graphen manuell ergänzt.

(4) Die Schritte (2) bis (3) werden solange wiederholt, bis alle möglichen Kontexte des Schlüsselwortes untersucht werden, sodass neue Grammatiken (z.B. die Kontexte auf der linken Seite von „*health*") entwickelt werden können,. Diese iterative Vorgehensweise wird „**bootstrapping**" genannt.

(5) (*optional*) Nach dem Ablauf der vorherigen Bootstrapping-Verarbeitung besteht noch die Möglichkeit, eine weitere Bootstrapping-Maßnahme durchzuführen. Die vorher erstellten Graphen zu dem Schlüsselwort „*health*" können dazu genutzt werden, neue Begriffe zu finden, die in dem gleichen Kontext wie „*health*" vorkommen. Dafür muss eine Variable an der Stelle „*health*" eingesetzt werden. Mit

diesen neuen Graphen können funktional ähnliche Wörter gefunden werden, die semantisch zu derselben Wortklasse gehören (beispielsweise Synonyme).

Diese Bootstrapping-Methode liefert bei der Grammatikentwickelung systematisch alle Ergebnisse aus dem eingegebenen Text (oder dem Korpus).

2.3.4.2 Werkzeug: Unitex

Unitex[38] ist eine Kollektion von Programmen, die für die Korpus-Analyse der natürlichen Sprachen entwickelt wurden. Dieses Tool wurde am Anfang im LADL unter der Leitung von Gross erstellt. Heute werden ähnliche Ressourcen für andere Sprachen im Zusammenhang vom RELEX-Labornetz weiter entwickelt (Paumier 2003). Dieses Tool steht im Gegensatz zu INTEX[39] und NooJ[40] frei zur Verfügung und ist mit den gängigen Betriebssystemen kompatibel.

Unitex repräsentiert die Grammatiken auf der Basis von dem erwähnten RTN-Formalismus und beschreibt linguistische Phänomene auf unterschiedlichen Ebenen (Morphologie, Lexikon und Syntax). In dem Unitex-System sind auch mehrere elektronische Lexika verschiedener Sprachen integriert, die umfangreiche grammatische und semantische Informationen enthalten.

Unitex ist in der Lage, die Konkordanzen eines Schlüsselwortes aus den gegebenen Korpora automatisch zu ermitteln und sie in verschiedenen Formaten (HTML oder Text) auszugeben. Die Entwicklung lokaler Grammatiken wird durch solche Unterstützung teilweise automatisiert und vereinfacht.

---

[38] UNITEX ist ein frei verfügbares System. Die Idee ist unter der Leitung von Maurice Gross am LADL entstanden. Die Software wurde von Sebastien Paumier entwickelt. Siehe auch (Paumier 2006)
http://igm.univ-mlv.fr/~unitex

[39] INTEX ist ein UNITEX-ähnliches System, aber nicht frei gegeben.
http://intex.univ-fcomte.fr/

[40] NooJ eine Erweiterung von INTEX.
http://www.nooj4nlp.net/pages/nooj.html

Wie bereits vorgestellt, werden die erstellten lokalen Grammatiken im Unitex-System graphisch repräsentiert. Zum Testen eines Graphen kann Unitex den Graphen direkt auf den ausgewählten Text anwenden.

2.3.5 Anwendungen lokaler Grammatiken

Lokale Grammatiken werden in linguistischen Bereichen eingesetzt und erzielen in der Praxis die ersten Erfolge.

Eine typische Anwendung ist die Informationsextraktion (Senellart 1998), insbesondere die Erkennung von Eigennamen (Friburger/Maurel 2002a/b; Mallchok 2005; Nagel 2008). Ein anderes konkretes Anwendungsbeispiel ist die intelligente Volltextsuche. Bsiri/Geierhos /Ringlstetter (2008) präsentierten eine integrierte Plattform zur Jobsuche über das Internet. Die Kerntechnik ist, dass unstrukturierte Online-Jobanzeigen mit Hilfe von lokalen Grammatiken und elektronischen Lexika in eine semantisch strukturierte Form transformiert werden können. Darüber hinaus wird eine hohe Suchqualität sowie ein effizienter Zugriff auf die Daten ermöglicht.

Auch bei automatischer Übersetzung finden lokale Grammatiken praktische Anwendung. Transduktoren bieten die Möglichkeit an, einen erkannten Satz in einer anderen Form zu schreiben. Das heißt, lokale Grammatiken ermöglichen eine direkte Übersetzung des eingegebenen Satzes in eine andere Sprache. Aufgrund der Komplexität der natürlichen Sprachen kann die Übersetzung allerdings allein auf syntaktischer Basis mit gewissen semantischen Informationen nicht gelingen. Der Vorzug lokaler Grammatiken ist, feste Redewendungen präzise zu übersetzen, was zurzeit bei den meisten Übersetzungstools noch eine Schwäche darstellt.

Ein weiteres Anwendungsgebiet lokaler Grammatiken ist die Disambiguierung. Im Englischen ist ein Wort oft mehrdeutig, wenn es unterschiedlichen Kategorien zugeordnet werden kann. Ohne Kontext verursacht dieses Wort lexikalische Ambiguität. Eine andere Art der Ambiguität wird durch unterschiedliche Zuweisungen syntaktischer Strukturen (z.B. unterschiedliche Anbindungen von PPs) verursacht und wird „strukturelle Ambiguität“ genannt.

Da die Regeln lokaler Grammatiken die Kontexte als Muster sehr präzise beschreiben (Roche 1992), können die Ambiguitäten jedoch aufgelöst werden.

2.3.6 Fazit

In der Grammatikentwickelung gewinnen die korpusbasierten Ansätze immer mehr an Bedeutung. Die Grammatikregeln sollen nicht „ausgedacht“ werden. Stattdessen werden sie durch induktive Methoden aus den vorhandenen Texten extrahiert. Aus syntaktischen und/oder semantischen Analysen realer Korpora entstehen die Grammatiken, die diese natürliche Sprache beschreiben können. Dies ist auch eine Grundidee von lokalen Grammatiken.

Lokale Grammatiken sind eigentlich kontextfreie Grammatiken, die auch rekursive Übergangsnetzwerke nutzen. Sie sind manuell erstellte Regeln, die als endliche Automaten betrachtet werden können. Die Vorgehensweise der Erstellung und ein nützliches Tool wurden in den vorherigen Kapiteln vorgestellt.

Lokale Grammatiken werden normalerweise in einer graphischen Form dargestellt und erzielen in der Praxis die ersten Erfolge:

- Die graphische Darstellung ist sehr übersichtlich und verständlich.
- Durch die Graphendarstellung können sie zu Subgraphen modularisiert werden.
- Lokale Grammatiken erfolgen durch vielseitige praktische Anwendungen. Z.B. die Eigennamenerkennung in der Informationsextraktion, die Qualitätserhöhung in der Volltextsuche, automatische Übersetzung fester Redewendung oder die Disambiguierung etc.

Der Nachteil lokaler Grammatiken erklärt sich folgendermaßen:

- Da jeder Pfad in einer lokalen Grammatik einem Satz entspricht, müssen sehr viele Regeln erstellt werden. Diese Erstellung wird bisher manuell oder semiautomatisch durchgeführt und kann manchmal sehr aufwendig sein.

Die vorliegende Arbeit versucht mit dem neuen Ansatz diese Entwicklungskosten der lokalen Grammatiken zu reduzieren.

3. EDSI-Ansatz

Als Beitrag zur Grammatik-Induktion wird in diesem Kapitel der neue Ansatz der effizienten Destillation Syntaktischer Informationen (EDSI) entwickelt und formal vorgestellt. Die Grundidee der automatischen Ansätze der Grammatik-Induktion fokussiert sich auf die Extraktion der identischen Patterns zwischen Sätzen. Eine sehr verbreitete Lösung ist, dass die Sätze zur Markierung der Position der Patterns wortweise miteinander verglichen werden. Durch verschiedene nachherige Maßnahmen werden die Patterns herausgezogen. Der Nachteil dieser Methode ist, dass die Verarbeitungskompetenz nicht sehr hoch ist. Die hier vorgeschlagene Lösung basiert auf einer speziellen Datenstruktur (sog. Index). Alle Sätze im Korpus werden indexiert, dadurch soll die Suche nach den Patterns beschleunigt werden. Diese Methode zeichnet sich besonders aus, wenn es sich um die Suche in einer großen Datenmenge handelt.

EDSI ist eigentlich ein korpusbasierter, automatischer (oder semi-automatischer, wenn die Induktion der lokalen Grammatiken auch als ein Bestandteil des Systems betrachtet wird) und statistischer Ansatz. Das grundlegende Prinzip hinter diesem EDSI-Ansatz, dass die funktional ähnlichen (gegenseitig ersetzbaren) Konstituenten tendieren in einem ähnlichen Kontext vorzukommen (Siehe Kapitel 2.1.2.2 und 2.2.4), wurde im letzten Kapitel konkret diskutiert. Zunächst werden die Patterns (die gemeinsamen Teile zwischen Sätzen) herausgefunden. Durch die Erkennung der Patterns können die Sätze in Konstituenten segmentiert werden. Das Problem der Extraktion der Konstituenten kann schließlich durch die Suche nach den Patterns und die Selektion der Konstituenten überwunden werden.

Der EDSI-Ansatz versucht die potenziellen syntaktischen Informationen allein aus nicht-strukturierten Korpora, die aus nicht-strukturierten Sätzen bestehen, zu extrahieren. Die gefundenen Informationen werden danach durch eine statistische Methode verfeinert, um so die besten Resultate herauszufinden. Daneben ist weder ein Training-Korpus noch eine linguistische Information (z.B. Annotation) erforderlich. Die eingegebenen Korpora werden anschließend in Form der Baumbank (engl. *treebank*) transformiert.

Die folgende Abbildung stellt eine Übersicht des Ansatzes dar:

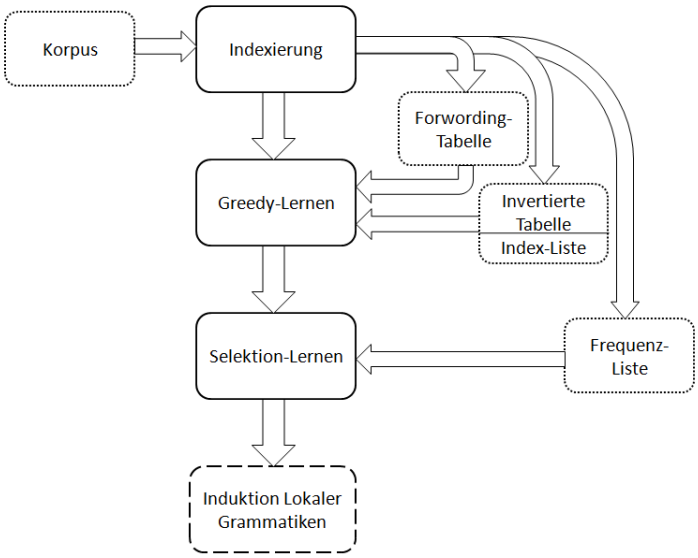


Abbildung 3.1: Gesamter Prozess des Ansatzes EDSI

Das eingegebene Korpus wird durch eine Vorverarbeitung in Sätze zerlegt und normalisiert. Es werden daher allein die Sätze aus dem originalen Korpus eingegeben. Die in der vorliegenden Untersuchung verwendeten Korpora stammen aus dem Tübingen VERBMOBIL Treebanks. Die Sätze sind aus den Baumstrukturen wiederhergestellt. Somit sind die Sätze bereits normalisiert. Die zusätzlichen syntaktischen oder semantischen Informationen aus den Baumstrukturen werden jedoch bei der Wiederherstellung entfernt. Daher stehen nur die Sätze im einfachen Text-Format zur Verfügung.

Der EDSI-Ansatz erfolgt hauptsächlich in vier Schritten:

(1) Indexierung

(2) Greedy-Lernen

(3) Selektion-Lernen

(4) Induktion lokaler Grammatiken

Im ersten Schritt „Indexierung“ wird das eingegebene Korpus indexiert und in ein paar Matrizes umgewandelt, die einen effizienten Datenzugriff gewährleisten können.

Das Greedy-Lernen untersucht jeden Satz und sammelt sämtliche Strukturen und Konstituenten (syntaktische Informationen) als Kandidaten, die dann in dem Pool abgespeichert und im nächsten Schritt verfeinert werden.

Im Schritt des Selektion-Lernens können die wahrscheinlichsten Resultate durch eine statistische Methode aus den zwischengelagerten Informationen ausgewählt werden. Daraus resultieren die Baumstrukturen.

Anschließend können die (lokalen) Grammatiken mit Hilfe der Baumstrukturen induziert werden.

3.1 Relevante Definitionen

Bevor wir weiter auf den Algorithmus eingehen, sollen die wichtigen Begriffe, die mit dem Thema der vorliegenden Arbeit einhergehen, zunächst mit Hilfe von Beispielen (siehe Abbildung 3.2) verdeutlicht werden[41].

> **Satz**[42] (engl. *sentence*): Ein Satz S mit der Länge n ($n > 1$) in dieser Arbeit ist eine Zeichenkette, die aus Termen $[t_1, t_2, \ldots t_n]$ besteht, wobei die Terme die elementaren linguistischen Einheiten[43] sind, die zur Untersuchung gehören, beispielsweise Wörter, Interpunktionen, Ziffern etc.
>
> **Pattern** (engl. *pattern*): Ein Pattern eines Satzes S ist ein Tupel $p^S = < t_1, t_2, \ldots t_i >$, wobei $t_1, t_2, \ldots t_i$ die Terme im Pattern sind.

[41] Fast alle wichtigen Begriffe dieses Forschungsbereiches hat bereits van Zaanen systematisch und formal definiert. Für die konkrete Beschreibung siehe auch van Zaanen (2001). Da die vorliegende Arbeit teilweise identische Begriffe umfasst, werden einige seiner Definitionen (***constituent, hypothesis, fuzzy tree, subsentence or word group & substitutability***) in dieser Arbeit verwendet.

[42] Vergleich zur Definition „sentence“ (van Zaanen 2001, 23)

[43] Die Einheit kann ggf. auch Morpheme oder Phrasen enthalten, wird aber für die Untersuchung in der vorliegenden Arbeit auf Wörter, Interpunktionen und Ziffern beschränkt.

> **Konstituente**[44] (engl. *constituent*): Eine Konstituente in einem Satz S ist eine Liste von Termen $c_{i \ldots j}$, so dass $S = u + c_{i \ldots j} + w$ (das Zeichen + dient hier als ein Operator der Konkatenation in der Liste), wobei u und w die Listen von Termen sind. $c_{i \ldots j}$ ($i \leq j$) ist eine Liste mit $j - i$ Elemente, wobei wenn $1 \leq k \leq j - i$, dann $c_{i \ldots j}[k] = S[i + k]$. Eine Konstituente kann entweder leer (wenn $i = j$) sein oder den ganzen Satz (wenn $i = 0$ und $j = |S|$)[45].

Die Konstituenten sind die Basis zur Transformation eines Satzes in seine Baumstruktur. Entsprechend können sie statt als Sequenz aus Termen als Pseudo-Konstituente mit ihren Positionen im jeweiligen Satz, in dem sie vorkommen, dargestellt werden.

> **Pseudo-Konstituente**[46] (engl. *pseudo-constituent*): Eine Pseudo-Konstituente in einem Satz S ist ein Tupel $c^S = < b, e, n >$, wobei $0 \leq b \leq e \leq |S|$. b und e sind die Positionen in S, die jeweils den Anfang und das Ende einer Konstituente bezeichnen. n ist ein Nonterminal dieser Konstituente.

Das Greedy-Lernen extrahiert aber zunächst, wie bereits der Name verrät, möglichst viele Hypothesen aus dem Korpus.

> **Hypothese**[47] (engl. *hypothesis*): Eine Hypothese ist eine möglich wahre Konstituente. Sie dient dazu, herauszufinden, wo eine Konstituente möglicherweise (aber nicht zwangsläufig) vorkommen kann. Die Struktur einer Hypothese ist genau identisch mit der Struktur einer Konstituente.

---

44 Vergleich zur Definition „constituent“ (van Zaanen 2001, 24)

45 Um die eventuelle Verwirrung zu vermeiden, werden die Definitionen, die aus van Zaanens Arbeit stammen, an das EDSI-System angepasst und neu benannt. Vergleich zur Definition „Subsentence or word group“ (van Zaanen 2001, 25)

46 Vergleich zur Definition „constituent“ (van Zaanen 2001, 24)

47 Vergleich zur Definition „hypothesis“ (van Zaanen 2001, 24)

Pseudo-Hypothese[48] (engl. *pseudo-hypothesis*): Eine Pseudo-Hypothese ist ähnlich wie die Pseudo-Konstituente dargestellt worden. Die Struktur einer Pseudo-Hypothese ist genau identisch mit der Struktur einer Pseudo-Konstituente[49].

Das Hauptziel der vorliegenden Arbeit besteht darin, durch die Entdeckung der funktional ähnlichen Konstituenten das unstrukturierte Korpus in eine Treebank zu transformieren. Darüber hinaus können die Grammatiken auf eine bequeme Weise entwickelt werden.

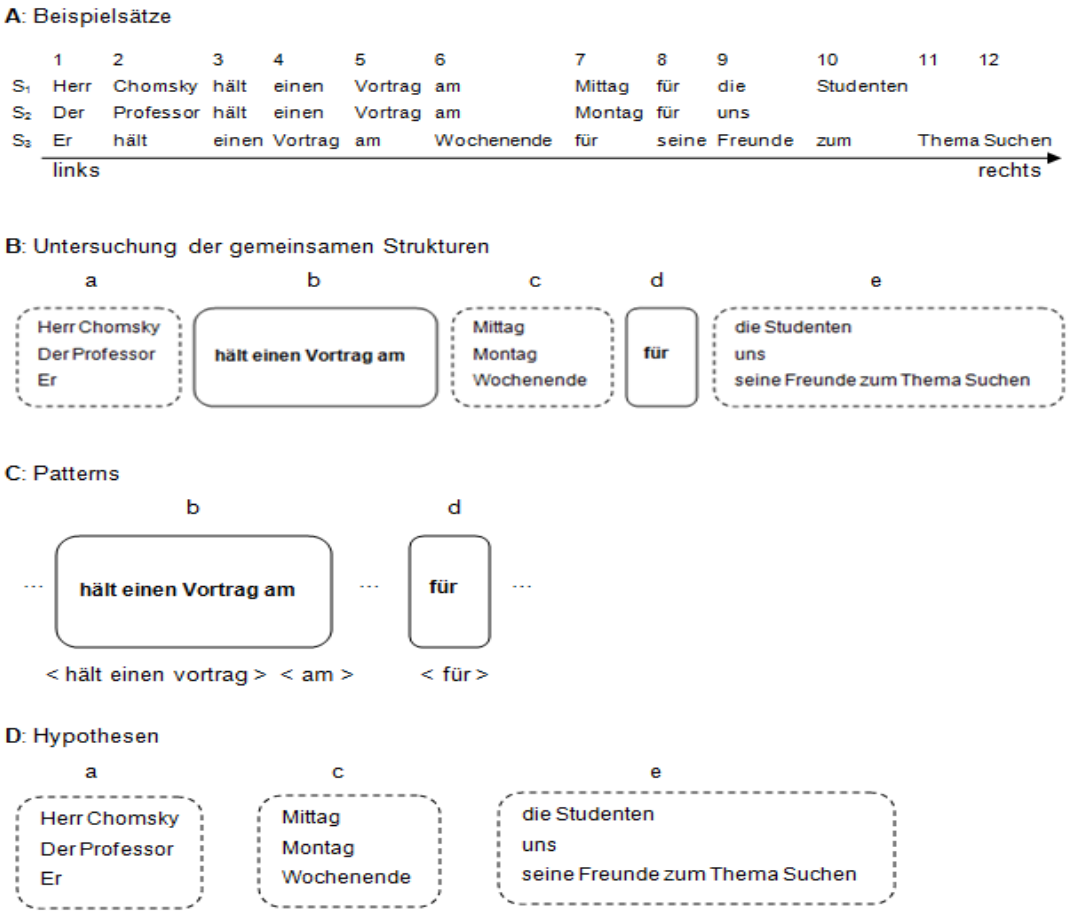


Abbildung 3.2: Extraktion von Patterns und Hypothesen

(A) Die Eingabe der Untersuchung wird durch drei deutsche Sätze, die ähnliche Strukturen enthalten, initialisiert. Normalerweise werden die Sätze termweise (jeder Term im Satz

[48] Vergleich zur Definition „hypothesis“ (van Zaanen 2001, 24)

[49] Vergleich zur Definition „hypothesis“ (van Zaanen 2001, 24)

wird durch seine Position markiert und entspricht einem Knoten im Automaten) von links nach rechts in der Schreibrichtung untersucht.

(B) Die gemeinsame Struktur (oder der Kontext) wird fett geschrieben. Die auf derselben Stelle vorkommenden Sequenzen werden als substituierbare Hypothesen gruppiert.

(C) Die Patterns stellen relativ häufige Redewendungen dar: „*halten*“ und „*Vortrag*“ kommen oft zusammen vor und bilden daher eine syntagmatische Relation. Zusätzlich treten manchmal auch Zeit (*am xxxTag*) und Zuhörer (*für xxx*) auf. Dies kann insbesondere für die Extraktion spezieller Informationen sehr interessant sein (siehe Kapitel 5).

(D) Die gefundenen Hypothesen, die in ein und demselben Kontext vorkommen, sind wahrscheinlich funktional-ähnliche Konstituenten, die sich in einer syntagmatischen Relation befinden. Diese funktionale Ähnlichkeit unterscheidet sich von der traditionellen Grammatik-Theorie. Beispielsweise gehören die Einträge aus der Gruppe a aus klassischer Sicht zu unterschiedlichen Klassen: NP und PP. Aber hier bekommen sie eine gemeinsame Klassen-ID, da sie in diesem Kontext durcheinander substituierbar sind. Allerdings ist dabei das Risiko nicht auszuschließen, dass manche falschen Hypothesen als Konstituenten erkannt werden. Z.B. „*seine Freunde zum Thema Suchen*“ soll durch die Patterns „*zum*“ oder „*zum Thema*“ nochmals zerlegt werden.

3.2 Indexierung

Die Index-Technik realisiert im Allgemeinen eine Suchfunktion und ermöglicht es relevante Dokumente aus der indexierten Dokumentenkollektion nach bestimmten Anforderungen möglichst schnell zu finden.

Diese Technik wurde vor geraumer Zeit zunächst für die bibliothekarische Anwendung entwickelt, z.B. die Aufbereitung von natürlichsprachigen Deskriptoren oder von Inhalt kennzeichnenden/beschreibenden Stich- oder Schlagwörtern (Luckhardt 1996). Die konventionelle Form dieser Erschließung ist manuelle Arbeit und wird als **intellektuelle Indexierung** bezeichnet.

Neben der intellektuellen Indexierung gewinnt **maschinelle** (oder **automatische**) **Indexierung,** die die Inhaltserschließung mit Hilfe von Computern durchführt, immer mehr an Bedeutung. Bei allen Texten einer Dokumentenkollektion werden die entsprechenden Indizes maschinell erzeugt. Dadurch kann die Suchgeschwindigkeit nach dem bestimmten Objekt erheblich erhöht werden.

Unter dem Begriff „Indexieren-Technik" werden verschiedene Methoden entwickelt: invertierte Dateien (engl. *inverted files*), Suffix-Array (engl. *suffix array*), Signatur-Dateien (engl. *signature files*) etc. (Baeza-Yates/Ribeiro-Neto 1999, 191).

Die erste Technik „**Invertierter Index**[50]" ist zurzeit die beste Lösung für die meisten Suchanwendungen. Eines der verschiedenen Verfahren der invertierten Dateien ist die Volltextindexierung (oder Freitextindexierung, engl. *full text indexing*). Dieses Verfahren ist heutzutage als Grundlage für die Volltextsuche (z.B. Web-Suchmaschinen Google und bing etc.) weit verbreitet und wird für den Indexaufbau in der vorliegenden Arbeit genutzt.

3.2.1 Vorgehensschritte des Indexaufbaus

Der Aufbau eines Index ist von dem Aufgabenziel und manchmal von den Sprachen abhängig und muss je nach dem unterschiedlich designt werden. Als Beispiel soll im Folgenden der Index, der in dieser Arbeit benötigt ist, schrittweise beschrieben werden.

3.2.1.1 Vorverarbeitung des Korpus

Die eingegebenen Korpora werden meist vorverarbeitet, z.B. normalisiert. Das heißt, alle zu indexierenden Terme haben ihre entsprechenden kanonischen Formen. Dies ist eine Reduzierung der Datenredundanz sowohl für den Index als auch für die Suche. Zusätzlich werden die Korpora in Sätze zerlegt und mit einer eindeutigen ID belegt, um die nachkommende Adressierung der resultierten Ergebnisse zu ermöglichen.

[50] Die Begriffe „invertierte Dateien" und „invertierter Index" beziehen sich auf das gleiche Objekt und werden als Synonym betrachtet.

Nach dem Aufgabenziel können verschiedene linguistische Maßnahmen eingesetzt werden, z.B. Pluralumlaute, Kompositazerlegung, Stopwortlisten etc. (Zimmermann 1979). Um die große Datenmenge zu verarbeiten, wird der Index oft komprimiert (Ristov 2002, 158). Da in der vorliegenden Untersuchung ein einfacher Index ausreicht, wird keine zusätzliche Verarbeitung benötigt.

3.2.1.2 Bildung der Matrizes

Der Index, der in dieser Arbeit aufgebaut ist, enthält insgesamt vier Matrizes:

- **Die Forward-Tabelle** ist eine zwei-dimensionale Matrix, die mit allen vorverarbeiteten Sätzen und den entsprechenden Satz-IDs gefüllt wird.
- **Die Indexterm[51]-Liste** ist auch eine zwei-dimensionale Matrix, in der alle kanonischen[52] Formen der indexierten Terme abgespeichert werden. Jeder Indexterm wird mit einer eindeutigen ID belegt.
- **Die Invertierte Tabelle** ist eine drei-dimensionale Matrix. Sie wird in den meisten Fällen als das Kernstück der modernen Indexierungstechnik eingesetzt.

  Der invertierte Index stellt eine der wichtigsten Datenstrukturen für effizientes Suchen her. Hierbei werden die Indexterme aus jedem Satz extrahiert und mit der Satz-ID der Tabelle hinzugefügt. Nach der Extraktion entsteht eine Liste aller Terme im Korpus und den Kollektionen der Satz-IDs. Demzufolge wird jeder Term mit Sätzen, in denen er vorkommt, angehängt. So können die Sätze, die den gesuchten Term enthalten, mit Hilfe dieses Terms schnell gefunden werden. Für das Ziel der vorliegenden Arbeit sind vor allem die Vorkommenspositionen aller

[51] Ein Indexterm unterscheidet sich meistens von seinem Term im Korpus. Dies ist abhängig davon, welche Maßnahmen bei der Korpus-Vorverarbeitung vorgenommen wurden. Z.B. die kanonische Form der Groß- oder Kleinschreibung, die Umschreibung der Spezialzeichen, wie „ä“ zu „ae“, ggf. eine Reduzierung auf den Wortstamm jedes Terms (Ortolf 2008) etc..

[52] D.h. alle Wörter werden kleingeschrieben und auch die Umlaute werden zu normalen Buchstaben umgewandelt. Ansonsten gibt es keine weiteren Maßnahmen.

Terme in jedem Satz wichtig. Daher werden die Positionen jedes Terms in jedem Satz mit der entsprechenden Satz-ID verbunden und in der Tabelle abgespeichert. Das heißt, alle Terme werden mit Sätzen positionell genau referenziert.

Der Begriff „*invertiert*“ bezieht sich dabei auf die Tatsache, dass alle indexierten Sätze auf eine invertierte Weise mit einer Liste dargestellt werden. Dadurch besteht die Möglichkeit, dass alle Sätze, die in dieser Liste zu finden sind, wiederhergestellt werden können.

- **Die Frequenz-Liste** dient dazu, die Häufigkeiten aller Indexterme in dem eingegebenen Korpus beizubehalten. Daher ist eine zwei-dimensionale Matrix geeignet.

  Weil die Wortfrequenzen manchmal von der Größe und den Themen des Korpus abhängig sind, insbesondere wenn das Korpus nicht ausreichend groß ist oder die Themen homogen umfasst, ist es notwendig, die Liste durch eine allgemeine Frequenzliste dieser Sprache zu ersetzen. In der vorliegenden Arbeit wird jeweils eine Liste im Englischen[53] und im Deutschen[54] bereitgestellt und zu diesem Zweck verwendet.

Die folgende Abbildung stellt beispielsweise die Strukturen und die Aufbauprinzipien der vier Matrizes dar.

[53] Die englische allgemeine Frequenzliste stammt aus dem Korpus OANC. Siehe Kapitel 3.4.1.1

[54] Die deutsche allgemeine Frequenzliste stammt aus dem IDS-Wikipedia Korpus. Siehe Kapitel 3.4.1.1

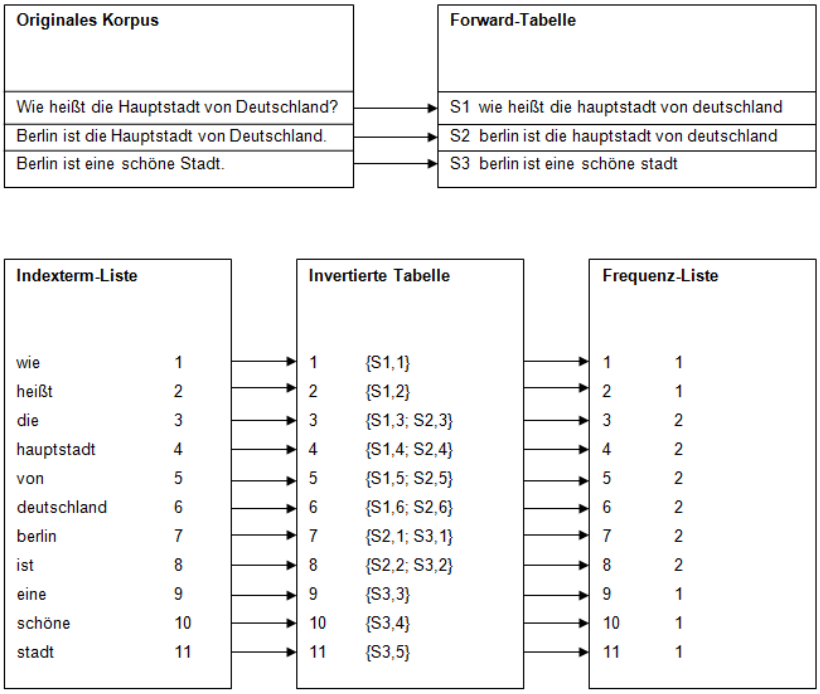


Abbildung 3.3: Matrizes des invertierten Index

Die originalen Korpora in dieser Arbeit werden normalisiert und auf der Wortebene in die Indexterme segmentiert. Weil auch die Interpunktionen zur Extraktion der syntaktischen Strukturen beitragen können, werden sie wie die Wörter als Term betrachtet und indexiert. Nur wenn solche Zeichen am Ende eines Satzes vorkommen, werden sie ausnahmsweise entfernt. Beim Indexieren werden alle Sätze, die im eingegebenen Korpus enthalten sind, vollständig in die Matrizes umgeschrieben.

3.2.2 Algorithmus

Zur konkreteren Beschreibung der Implementierung vom Indexieren in der vorliegenden Arbeit bietet sich ein Pseudo-Algorithmus an.

Angenommen enthält das Eingabe-Korpus m ($m > 0$) Sätze, die unterschiedlich lang sind. Jeder Satz besteht wiederum aus endlichen n ($n > 0$) Termen. Jedes Mal wird ein Satz zum Indexieren eingelesen.

Algorithmus 1: Indexieren

```
var FT:   Matrix der Forwarding-Tabelle
    IL:   Matrix der Indexterm-Liste
    IT:   Matrix der invertierten-Tabelle
    FL:   Matrix der Frequenz-Liste
    S:    ein eingegebener Satz
    NS:   normalisierter Satz
    SID:  Satz-ID
    I:    Indexterm
    IID:  Indexterm-ID
    F:    Häufigkeit des Indexterms
begin
  for m = 1 : M do        # Alle Sätze aus dem Korpus auslesen.
    create a unique id SID(m).      # Eine Satz-ID generieren.
    normalize the sentence S(m) to NS(SID(m)).      # Den eingegebenen Satz normalisieren.
    add (SID(m), NS(SID(m))) to FT.      # Die ID und den neuen Satz in Forwarding-Tabelle abspeichern.
    for n = 1 : N do
      if not exists IID(I(n)) do
        create a unique id IID(I(n)).      # Eine ID für den Term generieren, falls sie noch nicht existiert.
        add (I(n), IID(I(n))) to IL.      # Die ID und den Term in Indexterm-Liste speichern.
      endif
      update F(I(n)).          # Die Frequenz für den Term aktualisieren.
      add SID(m) and n to IT.    # Die Satz-ID und die Termposition in die invertierte Tabelle speichern.
    endfor
  endfor
  add F to FL      # Die Frequenz aller Termen in die Frequenz-Liste speichern.
end
```

Der Satz wird normalisiert und bekommt eine ID. Dadurch entsteht ein Eintrag für die Forward-Tabelle. Der normalisierte Satz wird termweise verarbeitet. Wobei jeder neuer Term auch eine ID erhält. Die wichtigste Matrix „invertierte Tabelle“ wächst durch das Hinzufügen der Term-ID und ihrer Satz-IDs. Bei jeder Ziehung eines Terms wird ebenfalls die Frequenz aktualisiert. Daraus resultiert schließlich die gesamte Frequenz-Liste.

Zeitaufwand:

Da es in der vorliegenden Arbeit um die Entwicklung eines effizienten Ansatzes geht, soll der Faktor des Zeitaufwands nicht übersehen werden. Die Laufzeit des Verfahrens muss in Abhängigkeit von der Datenmenge vorhersagbar bzw. abschätzbar sein.

Die Laufzeitkomplexität $T_{Indexieren(m,\ n)}$ besteht aus den folgenden Faktoren:

m ist die Anzahl der Sätze im Korpus;

n ist die durchschnittliche Länge aller Sätze im Korpus.

Alle Terme im Korpus werden hintereinander bis zu Ende ausgelesen und verarbeitet. Somit kann die Abschätzung der Laufzeit wie folgt formuliert werden:

$$T_{Indexieren}(m, n) = m * n$$

In der Tat ist der Zeitaufwand des Indexierens im Vergleich zu den nachkommenden Lernprozessen so gering, dass er beinahe ignoriert werden kann.

3.2.3 Effizienter Datenzugriff

Eine native Suchmethode, die die Texte im gespeicherten Korpus nach einer bestimmten Anfrage findet, ist der sequenzielle Durchlauf aller darin vorkommenden Texte. Bei einer großen Datenmenge ist ein solcher Ansatz allein aufgrund der Tatsache, dass die Suche durch die komplette Datenmenge zeitlich sehr aufwendig und in der praktischen Anwendung nicht akzeptabel ist, nicht performant.

Für die Analyse großer Datenmengen ist die Effizienz besonders bedeutend. Verschiedene moderne Gestaltungen der effizienten Datenzugriffe weisen unterschiedliche Mechanismen auf, um den Datenzugriff weitreichend zu optimieren. Zur Effizienzsteigerung bei Datenzugriffen wird heutzutage oft die Index-Technik verwendet. Ein Index ist ähnlich wie die Schlüsselwortverzeichnisse von Büchern und besteht aus den Kombinationen von Wörtern und ihren physikalischen Adressen.

Durch das Indexieren werden die Daten in einem homogenen Modell entsprechend der Problemstellung präsentiert und zur nachherigen Verwaltung bereitgestellt. Dabei können die Konflikte innerhalb von Daten, die wegen unterschiedlichen Datenquellen entstehen, behoben werden. Mit Hinblick auf das Aufgabenziel der vorliegenden Arbeit können daraus ein paar grundlegende Statistiken über die gesamte Datenmenge resultieren. Anderseits ermöglicht der Index einen effizienten Datenzugriff auf die potenziell relativ großen Datenbestände.

Der Index ist ein wichtiger Bestandteil des EDSI-Ansatzes und gewährleistet einen schnellen Zugriff auf die Daten. Dies stellt wiederum die Basis zur effizienten Destillation der syntaktischen Informationen dar.

3.3 Greedy-Lernen

Das Ziel des Greedy-Lernens besteht darin, die syntaktischen Informationen (einerseits die Konstituenten und anderseits die Patterns) in unstrukturierten Korpora herauszufinden und die lokalen Grammatiken zu induzieren.

Das Greedy-Lernen ist das Kernstück des EDSI-Ansatzes. Durch diesen unterscheidet sich der EDSI von den anderen statistischen Ansätzen.

Das Lernen der Patterns ist nichts anderes als die Suche nach den Hypothesen. Dabei wird jeder Satz im eingegebenen Korpus termweise[55]durch einen Automaten überprüft. D.h. jeder Satz bildet einen Suchpfad. Die klassischen Ansätze vergleichen jeden Satz mit jedem anderen, um die Patterns zu gewinnen. Dahingegen zieht das Greedy-Lernen alle potenziellen Patterns eines Korpus‘, die dem gezielten Satz entsprechen, durch eine Überprüfung komplett heraus. So wird die Laufzeit vergleichsweise sehr stark reduziert.

Beim Greedy-Lernen ist insbesondere zu beachten, dass möglichst viele aber auch möglichst wahrscheinliche Hypothesen gefunden werden, die dann durch das Selektion-Lernen verfeinert werden. Einerseits sind möglichst viele Hypothesen die Voraussetzung für einen hohen Recall. Anderseits kann die Präzision durch zu viele falsche Hypothesen, die durch das Selektion-Lernen nicht aussortiert werden können, beeinflusst werden.

[55] Weil neben den Wörtern auch die Interpunktionen Bestandteil eines Satzes (außer der Interpunktion am Satzende) sind und zur Extraktion der syntaktischen Informationen beitragen können, wird der Ausdruck „Term“ statt „Wort“ verwendet.

3.3.1 Vorgehensschritte des Greedy-Lernens

Beim Greedy-Lernen werden drei (die Forward-Tabelle, die Indexterm-Liste & die invertierte Tabelle) von den beim Indexieren erzeugten Matrizes benötigt. Auf der logischen Ebene erfolgt das Lernen in drei Prozessen (siehe die Beispiele in der folgenden Abbildung 3.3):

- **Term-Translation**: Hier werden die Terme in Sätzen durch ihre Positionen ersetzt.
- **Pattern-Extraktion**: Hier werden die in Sätzen enthaltenen Patterns extrahiert.
- **Hypothesen-Extraktion**: Mit Hilfe der Patterns, die bei der Pattern-Extraktion gewonnen werden, werden die Hypothesen leicht gefunden. Einerseits werden die Konstituenten, die in einem ähnlichen Kontext vorkommen, zu einer Gruppe klassifiziert. Anderseits wird der Target-Satz dadurch in die möglichen Konstituenten zerlegt und in eine Baumstruktur umgewandelt.

Die in Abbildung 3.3 dargestellten Beispiele präsentieren die typischen Fälle, die beim Lernen auftreten können.

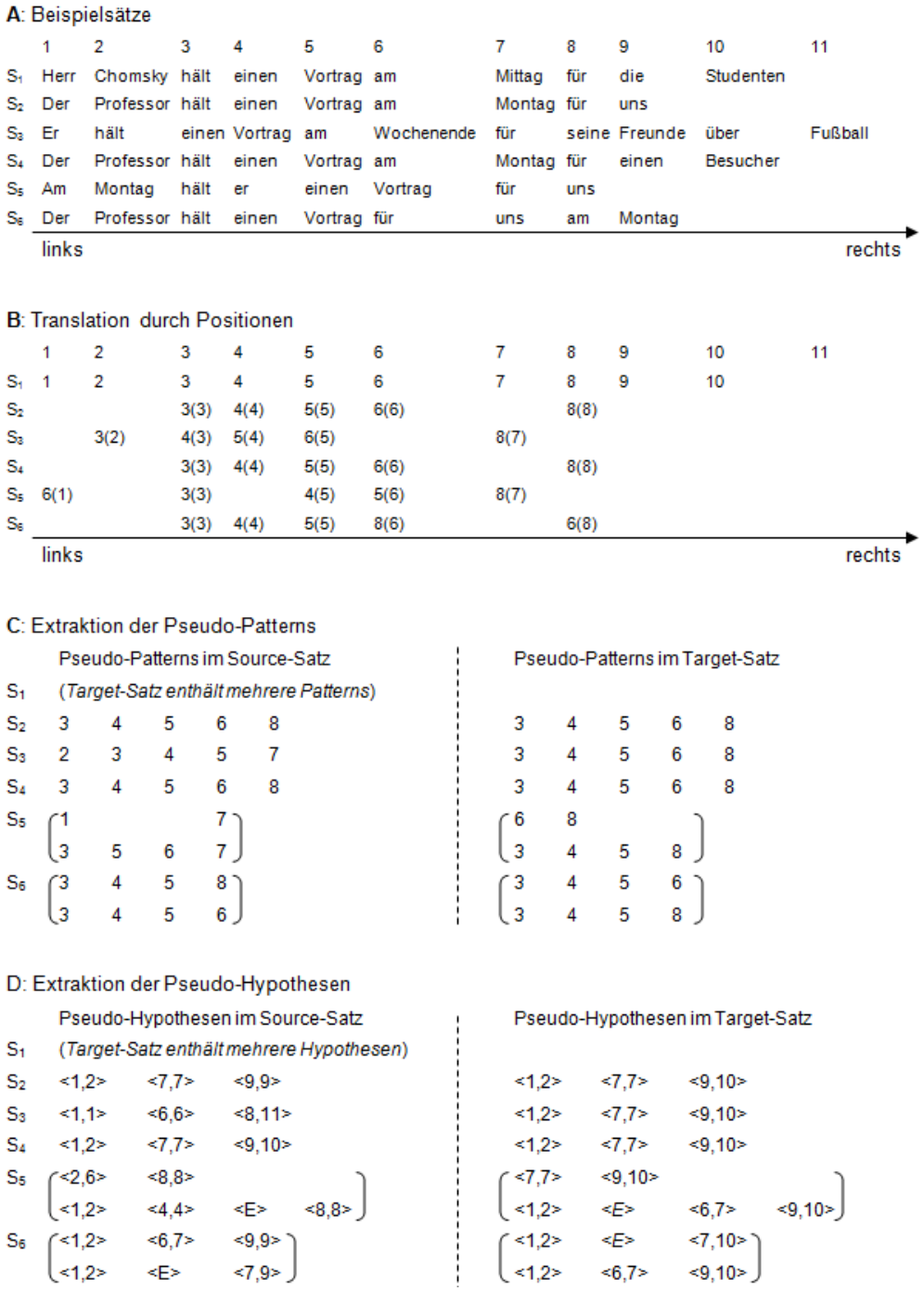


Abbildung 3.4: Greedy-Lernen

(A) Die sechs Sätze repräsentieren die typischen Fälle, die beim Greedy-Lernen auftreten können. Die ersten drei Sätze haben eine nah-lemmatisierte Form, da die Terme in Sätzen eine ganz normale Reihenfolge haben. Somit sind die Patterns und Konstituenten sehr leicht ersichtlich. S_4 enthält den Term „*einen*“ mehrfach. S_5 hat eine abweichende

aber korrekte Form. S_6 stellt eine Überschneidung der Patterns (zwischen „*für*“ und „*am*“) dar. Alle Sonderfälle müssen beim Lernen berücksichtigt werden, um die falsche Extraktion zu vermeiden.

(B) Diese Matrix enthält alle Translationen der Sätze in Referenz zu dem Target-Satz S_1. Weil S_1 nicht mit sich selbst verglichen werden muss, wird er nicht in eine paarweise Position sondern in eine eigene Position umformuliert.

(C) Alle möglichen Patterns werden extrahiert. Die linke Matrix enthält alle Pseudo-Patterns in Source-Sätzen (S_2 bis S_6). Jede Position dieser Patterns entspricht einer Position im Target-Satz S_1. Daher werden die entsprechenden Patterns im Target-Satz gefunden und in der rechten Matrix dargestellt.

Im Vergleich zu anderen ähnlichen Ansätzen ist zu beachten, dass mehrere Patterns aus einem Satz (z.B. *S5* und *S6*) gelernt werden können. Dass mehr Konstituenten gewonnen werden können, ist ein Vorteil des EDSI-Ansatzes.

(D) Durch die Pseudo-Patterns können die möglichen Pseudo-Hypothesen leicht gefunden werden.

Die Hypothesen aus Source-Sätzen, die in der linken Matrix enthalten sind, werden in Referenz zu ihrer Hypothese im Target-Satz zu einer funktionalen Klasse, die mit einer zufälligen eindeutigen ID beleget wird, gruppiert. Dabei ist erlaubt, dass eine Konstituente einen leeren Wert, der durch <*E*> symbolisiert werden kann, enthält (z.B. in S_5 und S_6).

Die Hypothesen aus den Target-Sätzen werden als Basis für die Konstruktion der Baumstrukturen (z.B. die rechte Matrix) genutzt. Ebenso ist es möglich, dass eine Hypothese mit leerem Wert auftritt. Da es nicht vorteilhaft ist, auf einen leeren Wert zu referenzieren, ist es nicht erlaubt, dass eine Hypothese „<*E*>“ im Target-Satz auftritt.

Die deutschen Sätze werden in der Schreibrichtung von links nach rechts analysiert. Bei jeder Untersuchung wird ein Satz als Referenz (Target-Satz) ausgewählt. Alle anderen Sätze sind Source-Sätze. Der **Target-Satz** ist der aktuelle Satz, der durch den Automaten überprüft wird

(siehe Beispielsfall S_1). Der **Source-Satz** bezieht sich auf alle anderen Sätze im Korpus (siehe Beispielsfall *S_2, S_3, S_4, S_5* und *S_6*).

3.3.1.1 Term-Translation

Der erste Prozess „Term-Translation" schreibt die Target- und Source-Sätze in die Sequenzen der Termpositionen um.

Vorgehensweise:

Alle Terme des Satzes aus der Forward-Tabelle werden in seiner Reihenfolge bis zum letzten Term ins System eingegeben. Für jeden Term wird zunächst seine ID in der Indexterm-Liste gesucht, womit alle Positionen dieses Terms in allen Sätzen (beispielsweise *S_2, S_3* oder *S_4*) aus der invertierten Tabelle ausgelesen werden. Dieser Satz muss nicht mit sich selbst verglichen werden[56].

Jeder Term, der nicht im Target-Satz vorkommt, wird in diesem Prozess nicht verarbeitet und bekommt im nachherigen Prozess automatisch einen leeren Wert. (im Beispiel steht ein Leerzeichen auf der Position). Andernfalls wird der Term durch die paarweise Position PositionInTarget (PositionInSource) ersetzt. So entsteht eine Matrix, die alle gemeinsamen Strukturen der Source-Sätze im Korpus zum Target-Satz repräsentiert.

Problemfall:

Ein eventueller Problemfall bei diesem Prozess wäre, dass ein Term im Source-Satz sowie im Target-Satz unterschiedlich oft vorkommt (z.B. kommt der Term „*einen*" in *S_1* auf der Stelle *4* nur einmal, in *S_4* auf den Stellen *4* und *9* zweimal vor). Daher muss entschieden werden, welche Position aus dem Source-Satz mit welcher Position aus dem Target-Satz am besten verlinkt werden soll.

Die Problemlösung bedient sich der Strategie, dass jede solcher Positionen mit der letzten gültigen Position verglichen wird, um so die erste größere Position aufzunehmen (Position *4*

[56] Als statistischer Ansatz ist ein Vergleich mit sich selbst für die Untersuchung nicht von besonderer Bedeutung.

in S_4 ist größer die letzte gültige Position *3* während Position *4* größer als Position *3* in S_1 ist). Damit werden die Positionen aus beiden Sätzen paarweise verbraucht (Position *4* aus S_4 mit Position *4* aus S_1) und die übrigen Positionen im Source-Satz verworfen (Position *9* aus S_4).

Nach der Translation werden die Terme aus den Source-Sätzen mit den paarweisen Positionen referenziert. Die potenziellen Patterns werden in der kommenden Extraktion mit Hilfe eines Vergleichs zwischen den Positionen ausgewählt.

Pseudo-Algorithmus:

Um die technische Realisierung zu verdeutlichen wird ein Pseudo-Algorithmus wie im Folgenden dargestellt.

Algorithmus 2: Term-Translation

```
var   FT:   Matrix der Forward-Tabelle              IL:   Matrix der Indexterm-Liste
      IT:   Matrix der invertierten-Tabelle         TT:   neue Matrix der Term-Translation
      S:    der Target-Satz                         SID:  Satz-ID
      t:    ein Indexterm                           IID:  Indexterm-ID
      It:   die letzte notierte gültige Position des Target-Satzes
      Is:   die letzte notierte gültige Position des Source-Satzes.
begin
  for i = 1 : N do     # Alle Termen aus dem Target-Satz auslesen.
    Indexterm t = S(i)
    read IID(t) from IL      # Die IndextermID aus IL finden.
    read Positions P of every Sentence SID from IT   # Die Positionen jedes Satzes aus IT auslesen.
    for j = 1 : M do   # Alle gefoundenen Source-Sätze durchgehen.
      It(j) = 0      # Die letzte gültige Position des Target-Satzes initialisieren
      Is(j) = 0      # Die letzte gültige Position des Source-Satzes initialisieren
      for k = 1 : K do     # Alle gefoundenen Positionen aus einem Source-Satz durchgehen.
        If i > It(j) & k > Is(j) do
          TT(j)(i) = k    # Diese Translation abspeichern
          It(j) = i       # neue gültige Position
          Is(j) = k       # neue gültige Position
        endif
      endfor
    endfor
  endfor
end
```

Bereits hier ist der wesentliche Vorteil des EDSI-Ansatzes ersichtlich. Im Vergleich zu anderen korpusbasierten Ansätzen (z.B. der klassische Ansatz ABL), die jeden Satz mit jedem

anderen im Korpus termweise alignieren müssen, greift EDSI direkt auf die Sätze zu. So bleiben Zugriffe auf unnötige Daten aus.

3.3.1.2 Pattern-Extraktion

Der zweite Prozess „Pattern-Extraktion“ dient dazu, die kontinuierlichen Positionen aus den Translationen zu extrahieren. Die dahinter stehende Grundidee ist, dass die gefunden Terme aus den Target- und Source-Sätzen mit derselben Reihenfolge ein richtiges Pattern bilden (siehe S_2 und S_3 im Beispiel). Ansonsten kommt es zu einem gekreuzten Überlappungsfall (siehe S_5 im Beispiel). Um diese Überlappung auszuschließen, müssen nur die Terme einer Translation, die mit kontinuierlich aufsteigenden Positionen sowohl im Target-Satz als auch im Source-Satz auftreten, herausgefunden werden.

In diesem Prozess wird nur die bereitgestellte Translation-Matrix benötigt (siehe Block B in Abbildung 3.4). In Referenz zum Target-Satz S_1 wird der fünfte Satz wie folgend übersetzt:

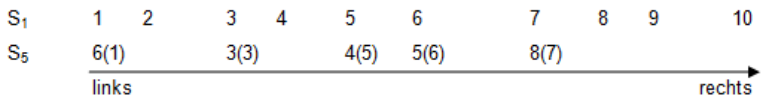


Abbildung 3.5: Gekreuzte Überlappung

Die Position *6(1)* vom S_5 befindet sich in einer Konfliktsituation mit den Postionen *3(3), 4(5)* und *5(6),* weil der Term „*am*“ in S_1 auf der Stelle *6* und in S_5 auf der Stelle *1* steht.

Vorgehensweise:

Die Extraktion geht nach der Reihenfolge des Target-Satzes S_1 vor. Der Term *3(3)* wird als die erste Position gefunden. Dabei wird ein neuer Kandidat erzeugt, der mit der Position „*3*“ (aus S_5) gefüllt und mit der Endposition „*3*“ (auch aus S_5) markiert wird. Zusätzlich wird eine minimale Position mit „*3*“ (aus S_5) initialisiert und geschaut, ob ein neuer Kandidat erzeugt werden muss. Beim nächsten Treffer *4(5)* wird die Position „*5*“ zu den Kandidaten hinzugefügt. Entsprechend steigt die Endposition auf *5*. Die minimale Position bleibt unverändert, weil sie bisher noch immer am kleinsten ist. Ebenso beim Treffer *5(6)*. Bis zur Position „*6*“ des Satzes S_1 kommt *6(1)* in die Verarbeitung. Weil „*1*“ kleiner ist als die aktuelle minimale Position „*3*“, wird ein neuer Kandidat angelegt, dem die Position „*1*“ zugewiesen wird.

Die minimale Position ist jetzt „*1*“. Die zwei Kandidaten haben jeweils die Endpositionen „*6*“ und „*1*“. Der letzte Treffer *8(7)* steigt weiter auf und wird den beiden Kandidaten zugefügt. Soweit werden zwei Patterns aus S_5 gefunden, nämlich:

$$S_5 \begin{pmatrix} 1 & & & 7 \\ 3 & 5 & 6 & 7 \end{pmatrix}$$

Abbildung 3.6: Pseudo-Patterns

Im Vergleich zu den klassischen Ansätzen wie ABL ist dies wiederum ein Vorteil des EDSI-Ansatzes. Der ABL-Ansatz findet lediglich das längste Pattern. Die Übrigen werden für gewöhnlich ignoriert. Eine höhere Anzahl von den Patterns hilft, einen höheren Recall zu erreichen.

Problemfall:

Falls ein nachkommender Treffer eine Position besitzt, die größer als die Anfangsposition aber kleiner als die Endposition eines Kandidaten ist, soll diese Position auf den Kandidaten zurückgreifen, bis ein neuer Kandidat aufgebaut wird.

Beispielsweise ist der letzte Treffer in S_5 statt *8(7)* sondern *8(4)*. Die Position „*4*“ muss in den ersten Kandidaten „*3 5 6*“ zurückrollen bis „*3 4*“ gefunden wird. Anschließend entstehen die Patterns:

$$S_5 \begin{pmatrix} 1 & 4 & \\ 3 & 5 & 6 \\ 3 & 4 & \end{pmatrix}$$

Abbildung 3.7: Extraktion im Falle von Zurückgreifen

Anbei wird gewährleistet, dass alle Patterns im Satz gewonnen werden. Im Extremfall (sehr viele Kandidaten in großer Länge) kann diese Operation jedoch viele Ressourcen verbrauchen.

Pseudo-Algorithmus:

Wie beim letzten Prozess ist ein Pseudo-Algorithmus für die Pattern-Extraktion wie folgt aufgebaut:

Algorithmus 3: Pattern-Extraktion

```
var  TT:  Matrix der Term-Translation;        C:  Pool der Kandidaten der Patterns
     N:   Translation des Target-Satzes;     M:  Source-Sätze
     mp:  Matrix der minimalen Position
begin
  for i = 1 : N do     # Alle Positionen aus dem Target-Satz auslesen.
    foreach j ∈ M do     # jeweils ein Source-Satz
      if exists TT(j)(i) do
        p = TT(j)(i)          # Die aktuelle Position des Source-Satzes
        if not exists mp(j) or p < mp(j) do  # Erster Treffer oder wenn p kleiner als minimale Position
          C(j)(p) = p         # Kandidat in Pool speichern. „+“ ist der Operator fürs Hinzufügen
          mp(j) = p           # Minimale Position
          bp(j)(p) = p        # Begin-Position
          ep(j)(p) = p        # End-Position
        else                  # Bei weitern Treffern
          foreach k ∈ C(j) do
            if p > ep(j)(k) do      # Falls die Position größer als die Endposition
              C(j)(k+p) = k+p       # Kandidat in Pool speichern.
              bp(j)(k+p) = bp(j)(k)
              ep(j)(k+p) = p
              delete C(j)(k)        # Den veralteten Kandidaten weglöschen
            elseif p > bp(j)(k) do  # Falls die Position nur größer als die Beginposition
              rollback to Position r, where r < p     # Hier zurückgreifen
              C(j)(r+p) = r+p       # Kandidat in Pool speichern.
              bp(j)(r+p) = bp(j)(k)
              ep(j)(r+p) = p
            endif
          endforeach
        endif
      endif
    endforeach
  endfor
```

Nach dieser Extraktion werden alle potenziellen Patterns, die sich in einem Source-Satz befinden, herausgefunden. Dabei soll es keine Subsequenzen geben, die zusätzlich in einem längeren Pattern enthalten sind.

3.3.1.3 Hypothesen-Extraktion

Die Slots, die sich in den durch die Pattern-Extraktion gefundenen Pseudo-Patterns befinden, sind potenzielle Hypothesen. Das Ziel der Hypothesen-Extraktion ist, dass die Hypothesen mit Hilfe der vorhandenen Pseudo-Patterns aus dem Korpus gezogen werden.

Einerseits sind die Hypothesen in den Target-Sätzen eine wichtige Basis zur Transformation des Korpus‘ in eine Treebank. Dies entspricht dem Hauptziel des EDSI-Ansatzes. Durch die Konstruktion der Baumstrukturen wird das System quantitativ evaluiert. Andererseits helfen die gruppierten Hypothesen aus den Source-Sätzen, die Grammatikregeln zu induzieren.

Die Hypothesen-Extraktion basiert auf der Annahme, dass die Hypothesen in demselben Kontext eine paradigmatische Relation (siehe Kapitel 2.1.2.2) besitzen und daher funktional substituierbar sind.

> **Substituierbarkeit**[57] (engl. *substitutability*): Zwei Konstituenten *u* und *v* sind gegenseitig substituierbar, wenn:
>
> 1. Die Sätze $S1 = t + u + w$ und $S2 = t + v + w$ (*t* und *w* sind eine Sequenz von Termen) sind beide gültig.
>
> 2. Es gilt immer die folgende Regel: Wenn $1 \leq i \leq |u|$, dann $u[i] \notin v$, ebenfalls wenn $1 \leq j \leq |v|$, dann $v[j] \notin u$.

Es gilt zu beachten, dass diese Definition eine Substitution der leeren Konstituente (eine leere Konstituente wird mit <*E*> bezeichnet) zulässt. Die Substitution zwischen zwei leeren Konstituenten ist jedoch nicht ausgeschlossen, d.h. mindestens eine Konstituente muss nicht leer sein[58].

Die Hypothesen „*Herr Chomsky*“ und „*Er*“ im unteren Beispiel (siehe Abbildung 3.2) sind in diesem Kontext austauschbar und werden einer funktionalen Klasse zugeordnet. Wenn „*Herr Chomsky*“ durch „*Er*“ ersetzt wird, entsteht ein neuer Satz, der grammatisch ebenfalls korrekt ist. Ebenso erfüllen die paarweisen Hypothesen „*am Mittag*“ und „<*E*>“ die Substituierbarkeit.

Die aus dem Korpus gewonnenen Pseudo-Hypothesen werden zunächst in zwei getrennten Pools (Fuzzy-Treebanks) abgespeichert.

[57] Vergleich zur Deffinition „substitutability“ (van Zaanen 2001, 25)

[58] Vergleich zur Definition „substitutability“ (van Zaanen 2001, 25)

Fuzzy-Baum[59] (engl. *fuzzy tree*): Ein Fuzzy-Baum ist ein Tupel $F = < S, H >$, wobei S ein Satz ist und H eine Menge von Pseudo-Hypothesen $\{h_1^S, h_2^S, \ldots\}$.

Fuzzy-Treebank[60] (engl *fuzzy treebank*): Eine Fuzzy-Treebank ist eine Menge von Fuzzy-Baumstrukturen.

Zwei Fuzzy-Treebanks, die für verschiedene Aufgabenzwecke zuständig sind, werden unterschiedlich konstruiert. Wie bereits erwähnt, ist die **Fuzzy-Treebank T** (siehe Beispiel in Anhang 7.2.1), die nur die Pseudo-Hypothesen aus den Target-Sätzen enthält, eine wichtige Voraussetzung für den Aufbau der Baumstruktur. Die andere **Fuzzy-Treebank S** (siehe Anhang 7.2.2), die alle Konstituenten-Klassen und ihre entsprechenden Hypothesen umfasst, wird für die Induktion der lokalen Grammatiken genutzt.

Vorgehensweise:

Die Patterns im Target- und Source-Satz sind miteinander paarweise verlinkt, d.h. jede Position im Target-Pattern hat auch (genau) eine verlinkte Position im Source-Pattern (siehe Beispiele A und B in Abbildung 3.4).

Da zwei unterschiedliche Fuzzy-Treebanks hergestellt werden müssen, wird jeweils ein Modus im Programm entwickelt.

- **Modus 1**: Jeder Target-Satz kann als ein Suchpfad betrachtet werden. Eine Durchsuchung des Target-Satzes S_1 findet mit Hilfe von Pseudo-Patterns (z.B. *<3, 4, 5, 6, 8>*) drei Slots (*<1,2>, <6,7>, <9,10>*) in S_1. Alle gefundenen Slots werden im Fuzzy-Baum T abgespeichert.

- **Modus 2**: Jeder Target-Satz kann als ein Suchpfad betrachtet werden. Eine Durchsuchung des Target-Satzes S_1 findet mit Hilfe von Pseudo-Patterns (z.B. *<3(2), 4(3), 5(4), 6(5), 8(7)>*) alle Slots (*<1,2>, <6,7>, <9,10>*) in S_1. Durch die verlinkten Positionen erscheinen auch die Slots (*<1,1>, <E>, <6,9>*) in Satz S_{10}. Falls

[59] Vergleich zur Definition „fuzzy tree“ (van Zaanen 2001, 24)

[60] Vergleich zur Definition „fuzzy treebank“ (van Zaanen 2001, 24)

die Slots aus den Source-Sätzen die Überprüfung ihrer Länge bestehen, werden sie in ihren jeweiligen Gruppen dem Fuzzy-Baum S hinzugefügt.

Die zwei Modi unterscheiden sich darin, dass die gelernten Hypothesen zur Konstruktion der Fuzzy-Treebank S verfeinert werden. Die Fuzzy-Treebank S ist nur bei der Induktion lokaler Grammatiken notwendig. Daher wird Modus 1 in dem hier angeführten Ansatz als Default-Wert eingesetzt.

Problemfälle:

Mit Hilfe der Pseudo-Patterns ist es wesentlich angenehmer die Pseudo-Hypothesen zu finden. Bei der Extraktion können jedoch zwei Probleme auftreten, die im Folgenden analysiert werden. Entsprechend werden ihre Lösungsstrategien angeboten:

- Im Target-Satz kann möglicherweise eine leere Hypothese „<*E*>“ auftreten. D.h. eine Menge der Hypothesen aus den Source-Sätzen werden in Referenz zu einem leeren Wert gruppiert (z.B. S_5 und S_6 in Abbildung 3.4). Da die Kontexte, in denen die Hypothesen vorkommen, nicht mehr eindeutig, sondern verschieden sein könnten, werden die Hypothesen in diesem Fall (beispielsweise <*E*> im S_1 mit <*4,4*> im S_5) nicht aufgenommen.

  Dadurch, dass jeder Satz des Korpus‘ einmal die Rolle des Target-Satzes spielt, wird dieses Problem gelöst. Wenn *S5* als der Target-Satz betrachtet wird, werden die Hypothesen „<*4,4*>“ in S_5 und „<*E*>“ in S_1 durch einen eindeutigen Kontext gefunden.

- Die Hypothesen werden, obwohl sie sich in einem ähnlichen Kontext befinden, den Gruppen nicht immer richtig zugeordnet. Insbesondere in den Sprachen, die eine flexible Satzstellung haben.

  Der Satz S_5 (in Abbildung 3.4) hat eine abweichendere Satzstellung als S_1. Dadurch werden „<*2,6*>“ in S_5 und „<*7,7*>“ in S_1 falsch als äquivalente Hypothesen erkannt. Eine Problemlösung wäre die Lemmatisierung (Zimmermann 1974) des ganzen Korpus. Die Lemmatisierung ist jedoch meist mit aufwendiger manueller Arbeit verknüpft und widerspricht den Zielsetzungen der vorliegenden Arbeit. Eine Alternative ist die statisti-

sche Methode, die die Länge der Hypothesen vergleicht. Variiert die Länge der Hypothesen stark, gehören sie nur sehr unwahrscheinlich zur selben Gruppe.

Von dieser Maßnahme könnte auch ein anderer Fall profitieren. Die Klassifizierung (siehe Abbildung 3.4) des letzten Slots *„die Studenten"* in S_1 und *„seine Freunde über Fußball"* in S_3 ist zwar korrekt, aber fragwürdig. Sie können miteinander ausgetauscht werden, ohne die Grammatikalität der neu produzierten Sätze zu verletzen. Der zweite Slot könnte jedoch weiter in *„seine Freunde"* und *„über Fußball"* zerlegt werden. Vergleichsweise ist *„seine Freunde"* die bessere äquivalente Hypothese zu *„die Studenten"*.

Eine bestimmte Größe des Unterschieds kann jedoch nicht festgelegt wird, weil diese von verschiedenen externen Faktoren (z.B. die Korpora, die Sprachen usw.) beeinflusst werden kann. Der Wert in der vorliegenden Untersuchung liegt zwischen *3* und *7*.

Das zweite Problem kann durch die statistische Methode nicht komplett verhindert, sondern nur verringert werden. In Bezug auf das Aufgabenziel, dass die Gruppen nur für die Induktion der lokalen Grammatiken genutzt werden, entsteht dennoch keine Problematik.

Pseudo-Algorithmus:

Jeder Modus wird im Folgenden mit einem Algorithmus dargestellt und so die technische Realisierung beschrieben.

Algorithmus 4: Hypothese-Extraktion (Modus 1)

```
var   P:    Pattern-Pool;                    HT:   Fuzzy-Baum T
      FL:   Matrix der Frequenz-Liste
begin
  for i = 1 : P do      # Alle Patterns des Target-Satzes auslesen.
    find slot <Bt, Et> at beginning of P(i)     # Falls ein Slot am Anfang des Satzes existiert
    find slot <Bt, Et> at end of P(i)       # Falls ein Slot am Ende des Satzes existiert
    for j = 0 : |S| do    # Alle Knoten im Pattern durchgehen
      if S(j+1) > S(j) + 1 do    # Ein Slot gefunden
        Bt = S(j)+1
        Et = S(j+1)-1
        Et gets the frequcy from FL    # Et bekommt seine Frequenz aus der Frequenz-Tabelle.
        add <Bt, Et> to HT      # Die Hypothese in Fuzzy-Baum T hinzufügen
      endif
    endfor
  endfor
end
```

Algorithmus 5: Hypothese-Extraktion (Modus 2)

```
var   P:   Pattern-Pool;                    HT:   Fuzzy-Baum T
      N:   Alle Positionen des Patterns;    M:    Source-Sätze
begin
  for i = 1 : P do      # Alle Patterns des Target-Satzes auslesen.
    find slot <Bt, Et> at beginning of P(i)     # Falls ein Slot am Anfang des Satzes existiert
    find slot <Bt, Et> at end of P(i)       # Falls ein Slot am Ende des Satzes existiert
    for j = 0 : |S| do    # Alle Knoten im Pattern durchgehen
      if S(j+1) > S(j) + 1 do     # Ein Slot gefunden
        Bt = S(j)+1
        Et = S(j+1)-1
        add <Bt, Et> to HS      # Die Hypothese in Fuzzy-Baum S hinzufügen
      endif
      for k = 1 : M do
        find Bs with Bt for M(k)      # Jede Position im Target-Satz entspricht eine Position im
        find Es with Es for M(k)      # Source-Satz
        if <Bs,Es> und <Bt,Et> ähnlich lang sind do
          add <Bs, Es> to HS      # Die Hypothese in Fuzzy-Baum S hinzufügen
        endif
      endfor
    endfor
  endfor
end
```

Die Hypothesen in der Fuzzy-Treebank T werden mit der Häufigkeit des letzten Wortes markiert, um die nachkommende statistische Methode im Selektion-Lernen zu ermöglichen. Da die zwei Modi unterschiedliche Aufgaben haben, nehmen sie einen ungleichen Zeitaufwand in Anspruch. Beim Effizienzvergleich (siehe Kapitel 4.3.1.1 und Kapitel 4.3.2.1) wird nur der Modus 1 evaluiert, weil er der äquivalenten Lernphase des referenzierten klassischen Ansatzes entspricht.

3.3.2 Algorithmus

Kompakter Pseudo-Algorithmus:

Die ersten zwei getrennten Prozesse (Term-Translation und Pattern-Extraktion) können in der Praxis einheitlich implementiert werden, was eine kompakte Lösung bietet.

Algorithmus 6: Greedy-Lernen

```
var  IT:  Matrix der invertierten-Tabelle;      C:  Pool der Kandidaten der Patterns
     N:   Target-Sätze;                         M:  Source-Sätze
     mp:  Matrix der minimalen Position
begin
  for i = 1 : N do     # Alle Termen aus dem Target-Satz auslesen.
    Indexterm t = S(i)       # Ein Term aus Target-Satz
    read IID(t) from IL      # Die IndextermID aus Indexterm-Liste finden.
    read Positions P of SID from IT    # Die Positionen jedes Satzes aus IT auslesen.
    foreach j ∈ M do  # Alle gefoundenen Source-Sätze durchgehen.
      foreach p ∈ P(j) do
        if not exists mp(j) or p < mp(j) do    # Erster Treffer oder p kleier als minimale Position
          C(j)(p) = p      # Kandidat in Pool speichern. „+" ist der Operator fürs Hinzufügen
          mp(j) = p        # Minimale Position
          bp(j)(p) = p     # Begin-Position
          ep(j)(p) = p     # End-Position
        else               # Bei weitern Treffern
          foreach k ∈ C(j) do
            if p > ep(j)(k) do      # Falls die Position größer als die Endposition
              C(j)(k+p) = k+p       # Kandidat in Pool speichern.
              bp(j)(k+p) = bp(j)(k)
              ep(j)(k+p) = p
              delete C(j)(k)        # Den veralteten Kandidaten weglöschen
            elseif p > bp(j)(k) do  # Falls die Position nur größer als die Beginposition
              rollback to Position r, where r < p    # Hier zurückgreifen
              C(j)(r+p) = r+p       # Kandidat in Pool speichern
              bp(j)(r+p) = bp(j)(k)
              ep(j)(r+p) = p
            endif
          endforeach
        endif
      endforeach
    endforeach
  endfor
  Hypothes-Extraktion      # Die Hypothesen extrahieren
end
```

Die „Term-Translation" und die „Pattern-Extraktion" sind daher nur auf der logischen Ebene voneinander getrennt. Während der Translation werden jegliche Kandidaten der Patterns generiert. Dadurch soll die Effizienz verstärkt werden.

Aus den folgenden Gründen ist EDSI im Vergleich zum klassischen Ansatz (z.B. ABL) besonders effizient:

1) Nur die Terme, die tatsächlich im Target-Satz enthalten sind, werden untersucht. Die Terme in Source-Sätzen, die nicht im Target-Satz vorkommen, erhalten beim Greedy-Lernen automatisch einen leeren Wert und werden nicht weiter verarbeitet.

2) Alle Source-Sätze, die keinen Term des Target-Satzes enthalten, werden beim Suchen nicht berücksichtigt. Das heißt, dass nur wenige Sätze Teil der Untersuchung sind.

3) Nur der Target-Satz muss durch den Automaten analysiert werden. Alle beteiligten Source-Sätze werden in der invertierten Tabelle gefunden.

So wird die Laufzeit der Pattern-Extraktion erheblich reduziert. Je umfangreicher das Korpus ist, desto größer ist dieser Vorteil. Die Effizienz der gesamten Extraktion wird also wesentlich erhöht.

Zeitaufwand:

Die gesamte Verarbeitungszeit des Greedy-Lernens besteht hauptsächlich aus zwei hintereinander durchgeführten Prozessen:

$$T_{\textit{Term-Translation \& Pattern-Extraktion}} \text{ und } T_{\textit{Hypothesen-Extraktion}}$$

Der erste Teil $T_{\textit{Term-Translation \& Pattern-Extraktion}}$ macht den wesentlichen Unterschied zu den klassischen Ansätzen aus.

- $\boldsymbol{T_{\textit{Term-Translation \& Pattern-Extraktion}}}$ (m, n, i, j): Der Zeitaufwand des ersten Prozesses wird durch die folgenden Faktoren bestimmt:

  m: die Anzahl der Sätze im Korpus;

  n: die durchschnittliche Länge aller Sätze im Korpus;

  i: die durchschnittliche Satzanzahl, die die Terme im Source-Satz enthalten, wobei $(1 \leq i \leq m)$;

j: die durchschnittliche Anzahl der Patterns, die in einem Satz gefunden werden, wobei ($1 \leq j \leq n$).

Der Zeitaufwand kann wie folgt formuliert werden:

$$T_{\text{Term-Translation \& Pattern-Extraktion}}(m, n, i, j) = i\,j * m\,n$$

D.h. im schlimmsten Fall ist die Laufzeit $m^2\,n^2$. Im besten Fall beträgt die Laufzeit $m\,n$ (wenn $i = 1$ und $j = 1$).

- $\boldsymbol{T_{Hypothesen\text{-}Extraktion}}(r, s, t)$: Die Faktoren der Verarbeitungszeit des zweiten Prozesses sind folgende:

  r: die durchschnittliche Anzahl der unterschiedlichen Patterns, die ein Target-Satz enthält, wobei ($1 \leq r \leq 0{,}5n^2$);

  s: die durchschnittliche Länge aller Patterns, wobei ($1 \leq s \leq n$).

Für den Modus 2 wird noch ein zusätzlicher Faktor benötigt:

t: die durchschnittliche Anzahl der Source-Sätze in Referenz zu jedem Pattern ($1 \leq t \leq m$).

Wie bereits erwähnt, verursacht jeder Modus der Hypothesen-Extraktion einen anderen Zeitaufwand. Entsprechend wird die Abschätzung der Laufzeit formuliert:

Modus 1: $T_{Hypothesen\text{-}Extraktion}(r, s) = m * r * s$

Modus 2: $T_{Hypothesen\text{-}Extraktion}(r, s, t) = m * r * s * t$

Im schlimmsten Fall beträgt der zeitliche Aufwand jeweils $0{,}5n^3$ bis $0{,}5n^3m$. D.h. der Unterschied zwischen den beiden Modi im Hinblick auf den Zeitaufwand kann sehr groß sein.

3.3.3 Lernen durch Suche

Das Lernen der Patterns ist im Wesentlichen nichts anderes als die Aufgabe, die potentiellen Patterns in einem Suchraum zu finden. Auch Mitchell hat das Vorgehen des Lernens aus Bei-

spielen als Suchproblem betrachtet (Mitchell 1982). Eine Lernaufgabe aus Beispielen hat er wie folgt beschrieben:

> *Given:* *(1)* *A language in which to describe instances.*
> *(2)* *A language in which to describe generalizations.*
> *(3)* *A Matching predicate that matches generalizations to instances.*
> *(4)* *A set of positive and negative training instances of a target generalization to be learned.*
>
> *Determine:* *Generalizations within the provided language that are consistent with the presented training instances.* (Mitchell 1982, 204)

Anbei wird das Matching-Prädikat sowohl durch positive als auch negative Beispiele trainiert, sodass es in der Lage ist, die generierten Hypothesen zu identifizieren.

Die statistischen Lernmethoden der Pattern-Erkennung unterscheiden sich von der obigen Beschreibung darin, dass das Bewertungsprädikat durch die Wahrheitsbedingungen statt die Beispiele befähigt wird. Eine formale Definition ist geboten:

> Gegeben: (1) Hypothesensprache *HS* für das Lernobjekt,
> (2) Wahrheitsbedingungen *W* für die Identifikation eines Objektes,
> (3) Bewertungsprädikat *P* zur Identifizierung eines Lernobjekts.
>
> Ziel: Wenn ein Lernobjekt $o \in HS$, wobei *W(o)* wahr ist, dann ist *P(o)* wahr; ansonsten ist *P(o)* falsch.

Der Suchraum ist die Menge aller in der Hypothesensprache gebildeten möglichen Objekte (oder Ausdrücke in diesem Fall). Daher kann das Problem des Pattern-Lernens durch unterschiedliche Suchmethoden gelöst werden.

Wie in Kapitel 3.1.3 erwähnt, ist die einfachste Suchstrategie der Durchlauf durch die ganze Datenmenge, der auf dem Aufzählungsalgorithmus basiert. Diese Strategie zählt alle Lernobjekte auf und prüft so, ob sie die Wahrheitsbedingungen erfüllen können. Da die in der Untersuchung genutzten Korpora keine unendlichen Terme enthalten, kann der Aufzählungsalgo-

rithmus (mehr oder weniger) funktionieren (z.B. der Einsatz der klassischen Methoden). Er ist jedoch nicht besonders effizient.

Die Möglichkeit, einen effizienten Algorithmus zu entwickeln, besteht unter folgenden Bedingungen:

- Einerseits kann der Suchraum komprimiert oder reduziert werden. Existierten die überflüssigen Hypothesen von Anfang an nicht im Suchraum, entsteht auch nicht der Aufwand, sie zu identifizieren.
- Andererseits kann der Suchraum strukturiert werden. Dies ermöglicht die Suche nach den Hypothesen in einem begrenzten, relativ kleinen Raum.

Der im EDSI-Ansatz verwendete Index entspricht den obigen Vorteilen und ist darüber hinaus eine wesentliche Verbesserung zur Erhöhung der Effizienz (siehe die Resultate in Kapitel 4).

3.4 Selektion-Lernen

Beim Selektion-Lernen geht es hauptsächlich um die Verfeinerung der gefundenen Hypothesen. Daher werden die Konflikte zwischen den Hypothesen hinsichtlich einer Problemlösung analysiert. Anschließend wird eine Treebank durch die extrahierten Konstituenten konstruiert. D.h. alle Sätze des eingegebenen Korpus‘ werden in Baumstrukturen transformiert, die wiederum dabei helfen, die lokalen Grammatiken zu induzieren.

Die beim Greedy-Lernen gesammelten Hypothesen befinden sich oft in einer Konkurrenzsituation; sie überschneiden sich. In der vorliegenden Arbeit wird nur die Überlappung der Hypothesen im selben Satz berücksichtigt und untersucht. Die Lösung für das Überlappungsproblem ist eigentlich die Entscheidung zwischen den überlappenden Hypothesen. Mit Hinblick auf die Vorteile der statistischen Methoden wird eine Entscheidungsstrategie durch die Wortfrequenz entworfen, sodass die besten Hypothesen gefunden werden.

3.4.1 Vorgehensschritte des Selektion-Lernens

Die Vorgehensweise des Selektion-Lernens ist im Prinzip ähnlich wie der zweite Schritt des ABL-Ansatzes (siehe Kapitel 2.2.4.1). Das Selektion-Lernen kann grob in zwei zeitlich aufeinander folgende Schritte geteilt werden:

- **Konstituente-Selektion**. Die gefundenen Hypothesen müssen verfeinert werden, da sie oft im Überlappungsfall miteinander konkurrieren. Eine Entscheidungsstrategie muss etabliert werden, damit die besten Konstituenten gefunden werden können. Die neu gelernten Hypothesen aus dem Fuzzy-Baum sind die Konstituenten, die die Baumstrukturen bilden können. Damit ergibt sich ein wichtiges Ziel der vorliegenden Arbeit. Nämlich, dass das eingegebene Korpus in eine Treebank transformiert wird.

- **Treebank-Konstruktion**. Dies ist ein zusätzlicher Schritt zur graphischen Darstellung der resultierten Treebank. Heutzutage werden verschiedene Formen (z.B. Penn-Treebank oder XML-Format etc.) entwickelt, um die Treebank zu präsentieren. In der vorliegenden Arbeit wird eine der Penn-Treebank ähnliche Darstellungsform verwendet.

Hauptsächlich wird hier der erste Schritt des Lernens diskutiert. Im zweiten Schritt geht es einerseits um die Erkennung der Subknoten und anderseits um die Präsentation der Baumstruktur.

A: Beispielsätze

	1	2	3	4	5	6	7	8	9	10	11
S_1	Herr	Chomsky	hält	einen	Vortrag	am	Mittag	für	die	Studenten	
S_2	Der	Professor	hält	einen	Vortrag	am	Montag	für	uns		
S_3	Er	hält	einen	Vortrag	am	Wochenende	für	seine	Freunde	über	Fußball
S_4	Der	Professor	hält	einen	Vortrag	am	Montag	für	einen	Besucher	
S_5	Am	Montag	hält	er	einen	Vortrag	für	uns			
S_6	Der	Professor	hält	einen	Vortrag	für	uns	am	Montag		

links → rechts

B: Alle Pseudo-Hypothesen im Target-Satz

ID	C1	C2	C3	C4	C5
S_2	<1,2>		<7,7>		<9,10>
S_3	<1,2>		<7,7>		<9,10>
S_4	<1,2>		<7,7>		<9,10>
S_5			<7,7>		<9,10>
	<1,2>	<6,7>			<9,10>
S_6	<1,2>			<7,10>	
	<1,2>	<6,7>			<9,10>

C: Überlappung

S_7 Er erhält den Brief am Montag.

S_8 Herr Chomsky hält einen Vortrag am Montag.

S_9 Er erhält die Geschenke von senen Freunden.

S_8 (Herr Chomsky hält einen Vortrag) am Montag.

S_7 Er erhält (den Brief am Montag.)

S_9 Er erhält (die Geschenke von senen Freunden.)

Abbildung 3.8: Selektion-Lernen

(A) Die Beispielsätze aus dem Greedy-Lernen werden hier wieder verwendet, um die Vorgehensweise des Selektion-Lernens zu präsentieren. Die Sätze S_1 bis S_4 haben eine normale Satzstellung. Die Satzstellung von S_5 und S_6 ist im Vergleich zum Target-Satz S_1 etwas abweichend und verursacht eine gekreuzte Überlappung der Satzstellung.

(B) Diese Matrix enthält alle möglichen Hypothesen im Target-Satz, die beim Greedy-Lernen gefunden und gruppiert wurden. Jede Gruppe gehört zu einer funktionalen Klasse, die mit einer zufälligen aber eindeutigen Konstituenten-ID belegt wird. Dabei ist es nicht erlaubt, dass eine Hypothese im Target-Satz einen leeren Wert erhält.

(C) Das häufigste bei diesem Lernen auftretende Problem ist die Überlappung zwischen den Hypothesen im Target-Satz. Im Überlappungsfall kann der Satz nicht richtig in eine

Baumstruktur konvertiert werden. Die in der vorliegenden Arbeit vorgeschlagene Problemlösung wird mit Hilfe einer statistischen Methode entworfen.

Beim Selektion-Lernen geht es hauptsächlich um die Verfeinerung der Hypothesen. Jeder Target-Satz wird durch seine enthaltenen Konstituenten in Form einer Baumstruktur präsentiert.

3.4.1.1 Konstituente-Selektion

Dieser Selektionsprozess stellt wiederum ein Suchproblem dar, welches sich allerdings von dem Greedy-Lernen unterscheidet. In diesem Prozess werden die möglichst korrekten Konstituenten aus dem Fuzzy-Baum gesucht (gefunden).

Überlappung zwischen den Hypothesen:

Weil ein Target-Satz in Bezug auf jeden Source-Satz mindestens ein Pattern enthält, werden möglichweise viele unterschiedliche Patterns in diesem Target-Satz gefunden. In jedem Pattern des Target-Satzes befinden sich meist mehrere Hypothesen. Die Hypothesen aus einem Pattern überschneiden sich oft mit den Hypothesen aus den anderen Patterns. Dies führt zur Konkurrenz zwischen den Hypothesen (siehe den Satz S_7 in Abbildung 3.9) innerhalb eines Satzes:

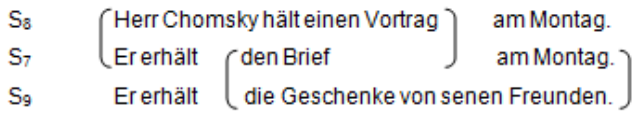


Abbildung 3.9: Überlappung zwischen den Hypothesen

Der Target-Satz S_7 hat das gemeinsame Pattern „*am Montag*“ mit S_8. Zugleich enthalten S_7 und S_9 das Pattern „*Er erhält*“. Im Satz S_7 werden die zwei Hypothesen „*Er erhält den Brief*“ und *„den Brief am Montag“* gefunden, die sich in einem sogenannten Überlappungsfall befinden. Diese Überlappung verhindert die Konstruktion der Baumstruktur des Satzes S_7. Daher muss eine Entscheidung für die (bessere) Hypothese (oder die Entfernung der anderen [schlechteren] Hypothesen) getroffen werden.

Statistische Strategie:

Die einfachste Lösung beim Suchen nach der besten Hypothese im Fall einer Überlappung basiert auf der „*first-come, first served*" Strategie. Diese Methode geht davon aus, dass die erste gelernte Hypothese der beste Kandidat ist. Alle danach gelernten überlappenden Hypothesen werden als falsche Kandidaten betrachtet und werden nicht mehr berücksichtigt. Diese Strategie ist leicht zu implementieren. Ihr Nachteil ist jedoch offensichtlich: die Resultate sind von der Reihenfolge der Sätze im Korpus abhängig. Durch die Veränderung der Reihenfolge können möglicherweise jedesmal unterschiedliche Ergebnisse erzielt werden.

Um den Nachteil der obigen Strategie zu vermeiden, werden statistische Strategien eingesetzt, die die Wahrscheinlichkeiten aller überlappenden Hypothesen mit Hilfe von verschiedenen statistischen Methoden berechnen. So kann die wahrscheinlichste Hypothese ausgewählt werden. Auch die in der vorliegenden Arbeit vorgeschlagenen Strategien beruhen auf diesem Prinzip, damit die beste Entscheidung getroffen werden kann.

In der folgenden Untersuchung werden zwei Strategien entworfen, die unterschiedliche statistische Methoden anwenden.

- **Strategie 1**: Die beim Indexieren erzeugte Frequenz-Liste der Terme wird zur Berechnung der Wahrscheinlichkeit verwendet. Es ist zu bemerken, dass häufige Wörter (z.B. Artikel oder Konjunktionen) meist nicht am Ende einer Phrase stehen. Das heißt, das letzte Wort in einer Konstituenten trägt normalerweise eine relativ kleine Frequenz wie im folgenden Beispiel ersichtlich:

Herr Chomsky hält einen Vortrag am Mittag für die Studenten			
Phrasen	**Wortfrequenz**		
Herr Chomsky	947		64
einen Vortrag	71689		165
hält einen Vortrag	1950	71689	165
am Mittag	82866		102
die Studenten	832976		1877
für die Studenten	165741	832976	1877

Abbildung 3.10: Wortfrequenzen aus dem Korpus IDS

Die verwendeten Wortsequenzen werden aus dem IDS-Korpus, das sehr umfangreiche Informationen aus verschiedenen Themenbereichen enthält, berechnet. Die obigen sechs

Beispiele zeigen deutlich, dass die Wortfrequenzen des letzten Wortes vergleichsweise klein sind. Dies ist die Grundidee hinter der ersten Strategie. Die Wahrscheinlichkeit *P* einer bestimmten Hypothese *h* kann durch die Frequenz *l* seines letzten Wortes im IDS-Korpus *T* berechnet werden.

$$P(h) = \frac{l(h)}{T}$$

- **Strategie 2**: Es wird angenommen, dass die häufigeren Hypothesen im Vergleich zu den selten vorkommenden Hypothesen mit höherer Wahrscheinlichkeit die richtigen Konstituenten sind. Die Wahrscheinlichkeit *P* einer bestimmten Hypothese *h* kann durch ihre Frequenz *f* im untersuchten Korpus *K* berechnet werden.

$$P(h) = \frac{f(h)}{K}$$

Die beiden Strategien bedienen sich der statistischen Methode, die die qualitativeren Ergebnisse liefert.

Problemfälle:

Bei dem Selektion-Lernen spielt die Wahrscheinlichkeiten der Hypothesen eine entscheidende Rolle. Die Berechnung der Wahrscheinlichkeiten stößt auf zwei mögliche Probleme.

- **Die Plausibilität der Wahrscheinlichkeit**. Durch die Hypothesen- und Wortsequenzen werden die Hypothesen bewertet. Aber die Distribution der Sequenzen ist bis zu einem gewissen Grad vom Korpus beeinflusst. Je größer das Korpus ist und je umfangreichere Themen es umfasst, desto glaubwürdiger sind die Frequenzen in statistischer Hinsicht.

- **Die Gleichheit der Frequenzen**. Die Frequenzen der unterschiedlichen Hypothesen können gleich sein, insbesondere wenn ein kleines Korpus verwendet wird. Die gleichen Frequenzen sind in der Tat wirkungslos für die Auswahl der Hypothesen. Durch ein externes, ausreichend umfangreiches Korpus (z.B. IDS-Korpus) kann dieses Problem deutlich verringert werden.

In der vorliegenden Untersuchung werden die Korpora „IDS[61]“ und „OANC[62]“ jeweils für die deutsche und englische Sprache eingesetzt.

Pseudo-Algorithmus:

Die benötigten Wahrscheinlichkeiten können während der Extraktion der Hypothesen vorbereitet werden.

Algorithmus 7: Konstituente-Selektion

```
var   HT:   Fuzzy-Baum T                    C:   Pool für ausgewählte Konstituenten
      N:    Alle Positionen des Patterns;   M:   Source-Sätze
begin
  calculate probability HT with freqency   # Die Wahrscheinlichkeiten vom HT berechnen
  sort with probability HT    # sortieren, damit die wahrscheinliche Hypothesen zuerst vorkommen
  for i = 1 : HT do      # Alle Hypothesen aus Fuzzy-Baum T sortiert auslesen
    find the position B(i) at beginning of HT(i)      # Den Anfang finden
    find the position E(i) at end of HT(i)            # Das Ende finden
    init checker = 1      # Falls die Wahrheitsbedingung erfüllt.
    if B(i) = E(i) do
      checker = 1
    else if
      for j = 1 : C do     # Alle Konstituenten im Pool durchgehen
        find the position B(j) at beginning of C(j)     # Den Anfang finden
        find the position E(j) at end of C(j)           # Das Ende finden
        if (B(i) < B(j) && E(i) > E(j)) or (B(i) > B(j) && E(i) < E(j)) or (B(i) > E(j)) or (E(i) < B(j)) do
        else
          checker = 0
        endif
      endfor
    endif
    if checker = 1 do
      add HT(i) to C
    endif
  endfor
end
```

Die beiden Strategien unterscheiden sich allein darin, dass sie auf unterschiedlichen Frequenzen basieren. Daher kann die Konstituente-Selektion durch einen einheitlichen Pseudo-Algorithmus wie folgt beschrieben werden.

---

[61] Das „German Wikipedia IDS-XCES Research Corpus“ ist eine Sammlung der deutschen Web-Seiten aus Wikipedia im April 2005. Das Korpus besteht aus umfangreichen Texten aus verschiedenen Themen und enthält über 3 Million Sätze. Dieses Korpus ist beim Institut für Deutsche Sprache erhältlich. Mehr Informationen gibt es unter http://www.ids-mannheim.de/kl/projekte/korpora/, (Stand: 07.12.2012).

[62] OANC (Open American National Corpus) ist ein frei verfügbares Korpus, das Texte aus Nachrichten, Briefen etc. enthält. Siehe auch http://www.anc.org/OANC/, (Stand: 07.12.2012).

3.4.1.2 Treebank-Konstruktion

Die verfeinerten Fuzzy-Bäume können in einer graphischen Form der Baumstruktur dargestellt werden. Eine Menge von Baumstrukturen ist eine Treebank. Das heißt, die Treebank enthält die syntaktischen Strukturen, die als Baumstruktur repräsentiert werden. Zusätzlich kann die Treebank mit semantischen oder anderen linguistischen Informationen erweitert werden. Mit Hilfe von Treebanks werden zahlreiche computerlinguistische Anwendungen ermöglicht (Lezius 2001, 414).

Je nach Bedarf der Aufgaben werden bereits verschiedene Präsentationsformen entwickelt. Beispielsweise ist die Penn-Treebank[63] eines der beliebtesten Formate, das für verschiedene Sprachen verwendet wird. Für die deutsche Sprache sind heute ebenfalls einige Treebanks (NEGRA[64], TIGER[65], TueBa-D/Z[66], TueBa-D/S und TueBa-E/S[67] etc.) vorhanden.

Die in diesem Schritt konstruierten Baumstrukturen werden ähnlich wie die Penn-Treebank (siehe Abbildung 4.13) präsentiert, wodurch zwei Aufgabenziele erreicht werden:

- **ID-Belegung**: Da die Konstituenten aus den nicht annotierten Korpora automatisch extrahiert und funktional klassifiziert werden, können sie nicht mit den POS-Tags der Penn-Treebank markiert werden. Stattdessen erhält jede Klasse eine automatisch beleg-

[63] Penn-Treebank ist ein Projekt von der Universität Pennsylvania, siehe http://www.cis.upenn.edu/~treebank/, (Stand: 07.12.2012) .

[64] NEGRA ist ein Projekt von der Uni des Saarlandes, das im gleichen Format wie die Penn-Treebank vorliegt. Siehe: http://www.coli.uni-saarland.de/projects/sfb378/negra-corpus/negra-corpus.html,(Stand: 07.12.2012).

[65] TIGER ist ein gemeinsames Projekt von der Universität Stuttgart, Universität Saarbrücken und Universität Potsdam. Siehe: http://www.ims.uni-stuttgart.de/projekte/TIGER/, (Stand: 07.12.2012) .

[66] TueBa-D/Z ist eine von der Universität Tübingen erstellte Treebank, die die Nachrichten aus der Tageszeitung erfasst.

[67] TueBa-D/S und TueBa-E/S Treebanks für deutsche und englische Sprachen wurden im Rahmen des Verbmobil Projekts von der Universität Tübingen entwickelt und ähneln der Penn-Treebank. Die zwei Korpora werden in der vorliegenden Untersuchung für die Evaluation der Lernergebnisse eingesetzt. Siehe: http://www.sfs.uni-tuebingen.de/resources/stylebook_vm_ger.pdf, (Stand: 07.12.2012) .

te ID. Falls eine Konstituente im Satz nicht identifiziert werden kann, findet für sie keine Belegung statt.

- **Knoten-Kombination**: Jede Konstituente bildet einen Knoten in der Baumstruktur. Daher müssen die Relationen zwischen den Knoten ermittelt werden.

Pseudo-Algorithmus:

Algorithmus 8: Treebank-Konstruktion

```
var   C:    Pool für ausgewählte Konstituenten      S:    Sätze im Korpus
begin
  for i = 1 : C do      # Alle Konstituenten im Pool durchgehen
    create ID for C(i)   # Eine ID belegen
    insert ID in S(i)    # ID in Satz hinzufügen
  endfor
end
```

3.4.2 Algorithmus

Der zweite Schritt „Treebank-Konstruktion“ dient dazu, die verfeinerten Fuzzy-Bäume graphisch darzustellen. Daher ist die „Treebank-Konstruktion“ ein optionaler Prozess, der von dem Selektion-Lernen getrennt werden kann.

Algorithmus 9: Selektion-Lernen

```
begin
  Konstituente-Selektion     # Die Hypothesen im Fuzzy-Baum verfeinern
  Treebank-Konstruktion (optional)     # Das Korpus in Treebank präsentieren
end
```

Zeitaufwand:

Da das Selektion-Lernen aus zwei hintereinander ablaufenden Prozessen besteht, kann der gesamte Zeitaufwand entsprechend zweigeteilt werden:

$$T_{Konstituente\text{-}Selektion} \text{ und/oder } T_{Treebank\text{-}Konstruktion}$$

- $\boldsymbol{T_{Konstituente\text{-}Selektion}}(u, v)$: Die Laufzeit ist von den folgenden zwei Faktoren abhängig:

  u: die durchschnittliche Anzahl der unterschiedlichen Hypothesen, die ein Fuzzy-Baum enthält, wobei ($1 \leq u \leq 0{,}5n^2$);

  v: die durchschnittliche Anzahl der ausgewählten Hypothesen in einem Fuzzy-Baum, wobei ($1 \leq v \leq u$);

Der Zeitaufwand wird nach dem Algorithmus wie folgt formuliert:

$$T_{Konstituente\text{-}Selektion}(u, v) = m * u * v$$

- $\boldsymbol{T_{Treebank\text{-}Konstruktion}}(v)$: Die Konstruktion ist allein von der Anzahl der verfeinerten Hypothesen im Fuzzy-Baum bestimmt:

$$T_{Treebank\text{-}Konstruktion}(v) = m * v$$

3.5 Induktion der Grammatikregeln

Aus den konstruierten Baumstrukturen können verschiedene Grammatikregeln induziert werden. Die Induktion der vorliegenden Arbeit basiert auf den kontextfreien Grammatiken (CFG, siehe Kapitel 2.3.3.1). Weitergehend wird die Induktion der lokalen Grammatiken (LG, siehe Kapitel 2.3.4) beschrieben.

Extraktion der kontextfreien Grammatiken

Die Grammatiken in den verfeinerten Treebanks werden im Allgemeinen als kontextfrei betrachtet und können durch einen Tree-Parser ausgelesen werden. Anbei wird eine Baumstruktur aus der TüBa-E/S als Beispiel gegeben (siehe Abbildung 3.11). Jeder Knoten[68] im Baum entspricht einer Konstituente. Das Symbol „*S*“ bezeichnet den ganzen Satz und ist der Startpunkt.

[68] Die TüBa-E/S Treebank wird semi-automatisch annotiert. Eine genaue Beschreibung der im Baum verwendeten POS-Tags ist unter http://www.sfs.uni-tuebingen.de/resources/stylebook_vm_eng.pdf (Stand: 07.12.2012) zu finden.

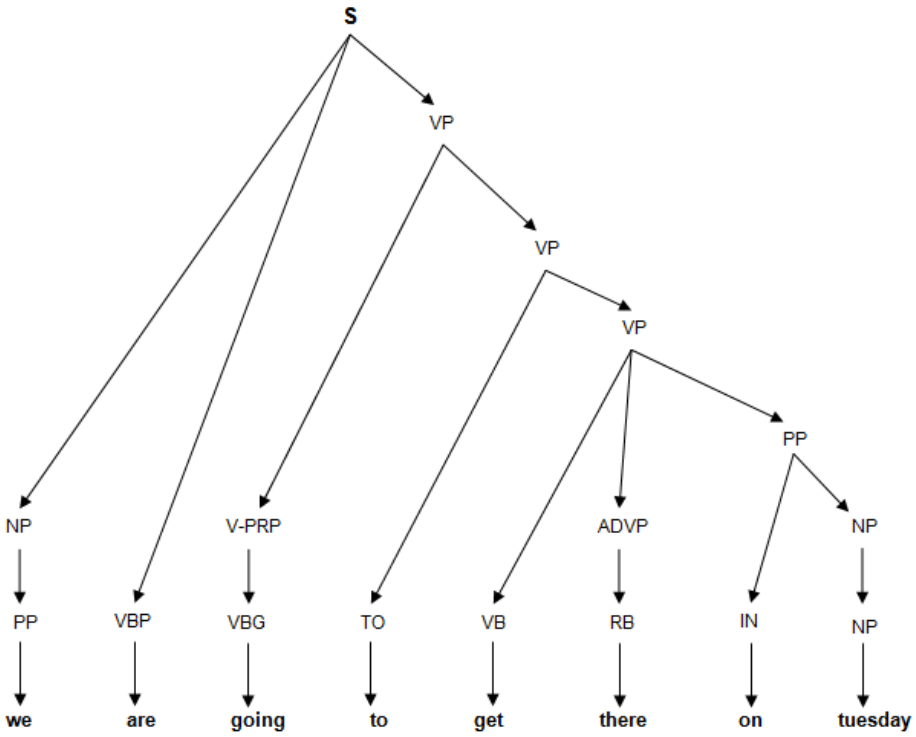


Abbildung 3.11: Originale Baumstruktur des Satzes 490

S	→	NP	VBP	VP
NP	→	PP		
VP	→	V-PRP	VP	
V-PRP	→	VBG		
VP	→	TO	VP	
VP	→	VB	ADVP	PP
ADVP	→	RB		
PP	→	IN	NP	
NP	→	NP		
PP	→	we		
VBP	→	are		
VBG	→	going		
TO	→	to		
VP	→	get		
RB	→	there		
IN	→	on		
NP	→	tuesday		

Abbildung 3.12: Produktionsregeln des Satzes 490

Jede kontextfreie Grammatik wird durch eine Menge ihrer Produktionsregeln präsentiert. Die Regeln können durch die Kollektion zwischen Eltern- und Tochterknoten von oben nach unten und von links nach rechts geparst werden. Beispielsweise werden die Produktionsregeln der Abbildung 3.11 wie oben extrahiert und dargestellt. Jeder Knoten in der Baumstruktur

wird durch ein nicht-terminales Symbol (die ID der entsprechenden Konstituente oder eine Bezeichnung „*X*“ der unbekannten Konstituenten, falls diese Konstituenten beim vorherigen Lernen nicht erkannt werden können) markiert und steht auf der linken Seite. Auf der rechten Seite befinden sich alle seine Subknoten, die ggf. terminal sind.

Die extrahierten kontextfreien Grammatiken sind in der Lage, ihre originalen Baumstrukturen wiederherzustellen, wenn die Extraktion rückwärts durchgeführt wird. Darüber hinaus können durch die Regeln neue Strukturen produziert werden, weil die funktional äquivalenten Konstituenten (die erkannten Knoten) beim vorherigen Lernen klassifiziert werden und eine identische Klassen-ID besitzen.

Diese Produktivität kann durch stochastische Methoden verstärkt werden. Nachdem alle kontextfreien Grammatiken einer Treebank gesammelt wurden, kann die Wahrscheinlichkeit jeder Grammatikregel durch ihre Frequenz berechnet werden. Daraus entstehen die stochastischen kontextfreien Grammatiken (SCFG, engl. *stochastic context-free grammar*). Die Auftrittswahrscheinlichkeit *P* einer kontextfreien Grammatik *g,* die einen linken Teil *gl* enthält, wird wie folgt formuliert:

$$P(g) = \frac{Freq(g)}{Freq(gl)}$$

Wobei *Freq(g)* die Häufigkeit der Grammatik *g* und *Freq(gl)* die Häufigkeit aller Grammatiken, die den linken Teil von *g* enthalten, ist.

Damit kann die wahrscheinlichste Regel in einer kontextfreien Grammatik, wo mehrere mögliche Produktionsregeln eingesetzt werden dürfen, berechnet werden.

Induktion der lokalen Grammatiken

Auch lokale Grammatiken sind im Wesentlichen kontextfrei und ähnlich wie stochastische kontextfreie Grammatiken. Der Unterschied besteht darin, dass lokale Grammatiken keine Wahrscheinlichkeiten berechnen. Alle Produktionsregeln der lokalen Grammatiken werden semi-automatisch erstellt und sind daher absolut (mindestens theoretisch) korrekt.

Um die lokalen Grammatiken zu induzieren, werden die im Greedy-Lernen erstellten Gruppen der äquivalenten Konstituenten in die entsprechenden Knoten der gewonnenen kontextfreien Grammatiken eingefügt (siehe das Beispiel in Kapitel 4.4.2.1). Allerdings müssen die Gruppen vorher manuell aufgeräumt werden, um zu gewährleisten, dass alle nachher produzierten Sätze korrekt sind. Durch das manuelle Aufräumen entsteht zwar ein Ressourcenaufwand, insbesondere der Zeit. Im Vergleich zu der normalen manuellen Erstellung der Grammatiken ist dieser Aufwand jedoch sehr gering. Dies macht das Ziel dieser Arbeit aus.

3.6 Fazit

In den vorstehenden Kapiteln wurden alle wichtigen Bestandteile sowie ihre Prinzipien des EDSI-Ansatzes detailiert dargestellt. Nebenbei wurde über die Probleme, die möglicherweise auftreten können, und ihre Lösungen diskutiert. So sind zunächst drei wichtige Punkte auf der theoretischen Ebene geklärt:

- Die enorme Effizienzerhöhung des EDSI-Ansatzes basiert auf der Konvertierung des Lernens zum Problem des Suchens. Die gesamte Verarbeitung wird durch den Index des Korpus vergleichsweise stark reduziert, wodurch der überflüssige Zeitaufwand gespart wird.

- Außerdem werden die extrahierten Treebanks durch eine neue Berechnungsmethode der Wahrscheinlichkeiten der Hypothesen verbessert.

- Die manuelle Induktion der lokalen Grammatiken kann dadurch semiautomatisiert werden. Somit wird der Zeitaufwand der Induktion entsprechend reduziert.

Der erste Punkt stellt das Hauptziel dieser Arbeit dar. Im folgenden Kapitel wird auf eine quantitative Evaluierung der Untersuchung eingegangen. Außerdem werden einige ausgewählte Beispiele für die Erstellung der lokalen Grammatiken präsentiert.

4. Evaluation der Systeme

In diesem Kapitel wird der EDSI-Ansatz in zwei natürlichen Sprachen instanziiert und durch quantitative und qualitative Methoden evaluiert. Weil es in der vorliegenden Arbeit um Effizienzerhöhung geht, wird die Evaluation im Vergleich zu dem klassischen Ansatz des ABL unter identischen Voraussetzungen durchgeführt.

Dabei werden einerseits der wichtigste Faktor (Laufzeit) sowie die anderen quantitativen Faktoren der beiden Ansätze miteinander verglichen. Anderseits wird eine qualitative Untersuchung für einen Teil der Testergebnisse durchgeführt und dargestellt.

Um die Plausibilität des Vergleichs zwischen den zwei Ansätzen zu gewährleisten, werden die Tests unter denselben Bedingungen implementiert. D.h. die Hardware und die bezogenen externen Ressourcen (z.B. die zur Evaluation genutzten Korpora) sind identisch. Die Software verwendet auch möglichst die gleichen Methoden und Module, falls es Gemeinsamkeiten gibt (z.B. die Evaluationsmethoden, das Modul der Treebank-Konstruktion etc.). Da die zwei Ansätze jeweils ein naives und ein erweitertes Lernen enthalten, werden die Resultate entsprechend in zwei Phrasen quantitativ evaluiert. Anschließend wird mit Hilfe von Beispielen die Vorgehensweise der Induktion der lokalen Grammatiken vorgestellt. Ebenso werden noch ein paar interessante linguistische Phänomene beschrieben.

4.1 Methode der Evaluation

Um einen Ansatz der Grammatik-Induktion zu evaluieren, gibt es heutzutage verschiedene Methoden (siehe Kapitel 2.2.5), die jeweils ihre eigenen Stärken in Bezug auf die Aufgabenziele haben.

4.1.1 Quantitative Evaluation

Eine quantitative Evaluation liefert klare und genaue Bewertungen eines Systems und ist daher viel überzeugender als eine abstrakte Betrachtung. Vor allem, wenn es sich um einen Per-

formancetest handelt, ist diese Evaluationsart notwendig. Mit Hilfe der Methode GST (Gold-Standard-Treebank) resultieren aus einer Evaluation quantitative Ergebnisse. Daher eignet sich diese Methode besonders für die vorliegende Untersuchung.

Die Voraussetzung der GST-Methode ist mindestens eine bereitgestellte und ausreichend umfangreiche Treebank. Alle Baumstrukturen in dieser Treebank werden als Evaluationsstandards betrachtet, die angenommen immer korrekt sind. Die in der vorliegenden Arbeit verwendeten Treebanks „EN_TueBa“ und „DE_TueBa“ sind jeweils eine Teilmenge[69] aus den TüBa-E/S und TüBa-D/S Treebanks. Die Treebank „EN_TueBa“ enthält 2275 Baumstrukturen während die Treebank „DE_TueBa“ aus 5559 Baumstrukturen besteht.

Vorgehensweise

Die untere Abbildung stellt die gesamte Vorgehensweise der quantitativen Evaluation sowie die Wanderung der originalen Treebanks in der vorliegenden Arbeit dar. In der Abbildung werden als Beispiel die englische Treebank und der Ansatz EDSI dargestellt. Diese Vorgehensweise gilt auch für die deutsche Treebank und den ABL-Ansatz.

1) Aus der originalen Treebank „TüBa-E/S“ werden die nicht wohlgeformten Baumstrukturen aussortiert. Dadurch entsteht die Treebank „EN_TueBa“, die eine Teilmenge der originalen Treebank ist und identisch der vorherigen konstruiert wird.

2) Aus den Baumstrukturen der neuen Treebank „EN_TueBa“ werden die Sätze extrahiert und als Plaintext wiederhergestellt. Die Sätze werden in einem gleichnamigen Korpus abgespeichert und für den Test bereitgestellt.

3) Auf dem Korpus „EN_TueBa“ wird der Ansatz EDSI durchgeführt. Durch den Test werden zwei Korpora produziert, die als Basis zur Induktion lokaler Grammatiken genutzt werden.

[69] Da die originalen Korpora nur aus den Umgangssprachen stammen, enthalten sie manche ungrammatische Sätze, die für diese Untersuchung eigentlich uninteressant sind. Dies ist im Sinne eines Vergleichs harmlos, weil die entstehende Störung für beide Ansätze gilt. Die Korpora werden, um bessere Resultate erzielen zu können, dennoch aufgeräumt. Die originalen Baumstrukturen werden keinesfalls verändert, sondern lediglich ausgewählt.

4) Zur Evaluation des Systems wird die Fuzzy-Treebank T mit der Treebank EN_TueBa verglichen. Wenn die Konstituenten in Fuzzy-Treebank T auch in EN_TueBa enthalten sind, werden sie als „richtig“ bewertet, ansonsten als „falsch“.

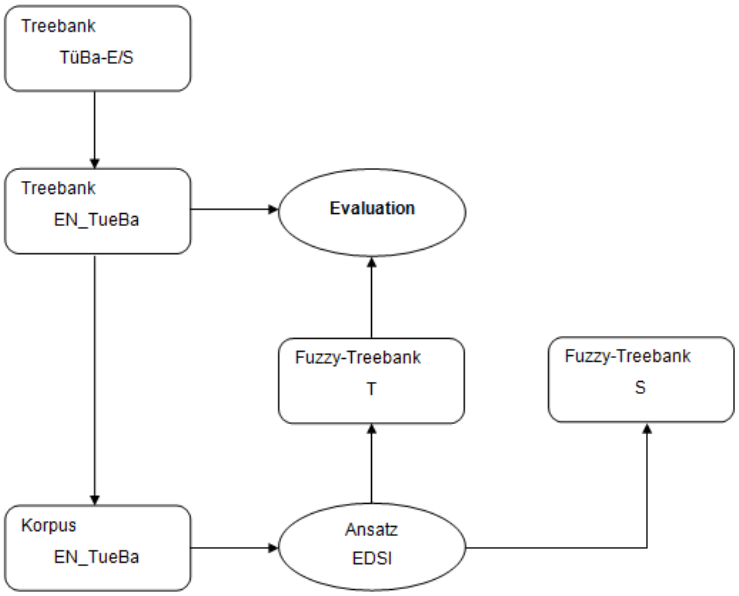


Abbildung 4.1: Vorgehensweise der quantitativen Evaluation

Das Korpus EN_TueBa besteht allein aus Sätzen, ohne jegliche dazugehörige syntaktische oder semantische Informationen aus der Treebank. Die durchschnittliche Satzlänge in den EN_TueBa und DE_TueBa Korpora beträgt 11,7 und 11,8 Terme. Die zwei Korpora werden als die nicht-annotierten Korpora in die beiden Ansätze eingegeben und analysiert. Anschließend sollen die Korpora in ihre entsprechenden Treebanks umgewandelt werden. Durch den Vergleich zwischen den originalen und den erzeugten Baumstrukturen werden die getesteten Systeme quantitativ evaluiert.

Performancemessung

Um die Performance der beiden Ansätze zu bewerten, werden in der Evaluation die folgenden Maße benötigt:

- **Laufzeit** *T* für den Ansatz *A*. Der Prozess startet zum Zeitpunkt *a* und endet zum Zeitpunkt *b*:

$$T_A = T_A(b) - T_A(a), wobei\ a < b$$

- **Präzision** *P* (engl. *Precision*) für die gesamte Menge der positiven Objekte *TP* (richtige und positive Objekte) + *FP* (falsche und positive Objekte) (Goutte/Gaussier 2005):

$$P = \frac{TP}{TP + FP}$$

- **Recall** *R* (engl. *recall*) für die gesamte Menge der richtigen Objekte *TP* (wie oben) + *FN* (falsche und negative Objekte) (Goutte/Gaussier 2005):

$$R = \frac{TP}{TP + FN}$$

Weil Präzision und Recall sich gegenseitig beeinflussen können (z.B. führt eine Erhöhung der Präzision normalerweise zur Senkung des Recall.), wird das F-Maß oft zur Bewertung eingesetzt. **F-Maß** *F* kombiniert Präzision und Recall, um einen harmonisch gewichteten Wert der beiden zu ermitteln (Goutte/Gaussier 2005).

$$F_\beta = \frac{\left(1 + \beta^2\right) * PR}{R + \beta^2 P}$$

Je größer β ist, desto mehr wird die Präzision gewichtet. Bei der Evaluation dieser Arbeit wird β auf „1“ gesetzt. Das heißt, Präzision und Recall haben den gleichen Beitrag zu diesem F-Wert.

Die oben vorgestellten Evaluationsmaße werden in den verschiedenen Testkontexten berechnet und ausgegeben. Damit kann die Performance eines Systems deutlich präsentiert und bewertet werden.

4.1.2 Qualitative Evaluation

Die andere beliebte Evaluationsmethode LGtm (*Looks-Good-to-me*) wird für die Einschätzung der gruppierten Hypothesen in der Fuzzy-Treebank S eingesetzt.

Beim Lernen können die Hypothesen der falschen Gruppe zugeordnet werden. Daher werden die Hypothesen in einer Gruppe nach den definierten Kriterien verfeinert. Aufgrund der Vielzahl der Hypothesen kann diese Verfeinerung nicht manuell durchgeführt werden. Die Qualität der resultierten Treebank maschinell zu evaluieren, ist auch ausgeschlossen, da die Hypothesen keine zusätzlich syntaktischen oder semantischen Informationen enthalten.

Mit Hinblick auf das Aufgabenziel ist in diesem Fall eine LGtm-Evaluation völlig ausreichend und geeignet. Durch die Bewertung von einem Teil der Fuzzy-Treebank wird das beste Verfeinerungskriterium festgelegt.

4.2 Testumgebung

In diesem Kapitel werden die relevanten Testumgebungen vorgestellt, die sich auf die Voraussetzungen der Implementierung und die internen Einstellungen der beiden zu vergleichenden Ansätze beziehen.

Zunächst wird die Umgebung durch die wichtigen Faktoren der Implementierung beschrieben. Danach werden die Tochtersysteme des von dieser Arbeit referenzierten Ansatzes ABL grob dargestellt. Ebenfalls wird eine solche Darstellung für den EDSI-Ansatz gegeben, da die verschiedenen Tochtersysteme zu unterschiedlichen Aufgabenzwecken dienen und unterschiedliche Resultate erzielen.

4.2.1 Umgebung der Implementierung

Auf der logischen Ebene kann die Umgebung der Implementierung in drei Bereiche geteilt werden:

- **Hardware**

Die kompletten Tests werden hintereinander auf ein und derselben Linux Maschine (**CPU**: Quad-Core AMD Opteron Processor; **Memory**: 2 GB) durchgeführt.

- **Implementierungstools**

Die Programme werden in PERL[70] 5.10.0 erstellt (Source-Code siehe Anhang 7.1). Für die speziellen Funktionen jedes Ansatzes werden die entsprechenden Module getrennt entwickelt, sodass sich die beiden Ansätze nicht gegenseitig beeinflussen.

Die entwickelten lokalen Grammatiken werden mit dem speziellen Tool Unitex (siehe Kapitel 2.3.4.2) erstellt und graphisch dargestellt.

- **Algorithmen**

Die zwei zu vergleichenden Ansätze haben jedoch auch ein paar gemeinsame oder zumindest sehr ähnliche Prozesse (Konstituente-Extraktion, Treebank-Konstruktion & Evaluation etc.), die dieselben Module oder ähnliche Algorithmen verwenden. Diese Maßnahme gewährleistet, dass die Ansatzlaufzeit nicht von den Programmen, sondern von den Prinzipien der Ansätze bestimmt wird. Daher wird die Plausibilität der Effizienztests nicht verletzt.

- **Relevante externe Ressourcen**

Die Ansätze werden auf identischen Korpora, die auf den im letzten Kapitel vorgestellten Tübingen-Treebanks basieren, evaluiert. Außer den Tübingen-Treebanks werden noch zwei weitere Korpora in der vorliegenden Arbeit verwendet. Die IDS- und OANC Korpora[71] werden jeweils für die Ermittlung der allgemeinen Wortfrequenzen im Deutschen und im Englischen genutzt.

---

[70] PERL ist die Abkürzung von Practical Extraction and Report Language. PERL ist eine gängige Programmierungssprache (Skriptsprache), die für verschiedene Aufgaben verwendet werden kann. Diese Sprache ist besonders für die Textverarbeitung geeignet.

[71] Siehe Kapitel 3.4.1.1

Die Implementierungsumgebung ist daher für die beiden zu vergleichenden Ansätze identisch. In diesem Sinne wird die Störung der externen Faktoren minimiert.

4.2.2 Umgebung des referenzierten Systems

Der referenzierte Ansatz ABL besteht aus mehreren Tochtersystemen. Grundsätzlich besteht ABL aus zwei obligatorischen Phasen und einer optionalen Phase:

(1) Alignment-Lernen;

(2) Selektion-Lernen und

(3) Grammatiken-Extraktion (zusätzlich)

Für jede Lernphrase werden drei unterschiedliche Instanzen entwickelt. Durch die Kombination der Instanzen entstehen verschiedene Tochtersysteme des ABL-Ansatzes wie im Folgenden ersichtlich (van Zaanen 2001, 66):

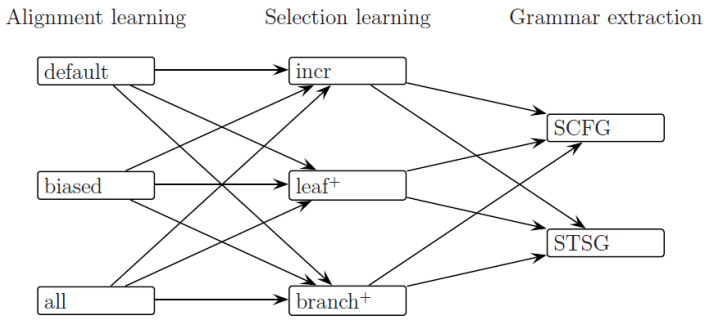


Abbildung 4.2: ABL Systeme

Für den Vergleichstest zwischen den ABL- und EDSI-Ansätzen sind nicht alle Tochtersysteme notwendig. Es sollte also ein typisches System ausgewählt werden, das in der Lage ist, den ABL-Ansatz in diesem Test zu vertreten. Hinsichtlich der Untersuchungsziele, dass die Effizienzerhöhung und die nebenbei erzielte Verbesserung der Resultate des neuen Ansatzes durch den Vergleichstest bewiesen werden müssen, soll das gewählte Tochtersystem die folgenden Bedingungen erfüllen:

1) Das System muss unter allen Tochtersystemen eines der schnellsten Systeme sein.

2) Das System soll im Vergleich zu den anderen Tochtersystemen gute Resultate ausliefern können.

Aus diesem Grund werden die Instanzen „default[72]“ aus dem „Alignment learning“ und „incr[73]“ aus dem „Selection learning“ im Test implementiert und zum Vergleich verwendet. Die Resultate aus der Instanz „incr“ sind von der Reihenfolge des Korpus abhängig und variieren daher etwas. Eine vorgeschlagene Stabilisierungsmethode[74] ist, ein Durchschnittswert aus mehreren Tests zu berechnen. D.h. das Korpus, das jedes Mal in einer zufälligen Reihenfolge dieselben Sätze enthält, wird zehnfach in das System eingegeben. Anschließend resultiert aus diesen zehn Ergebnissen ein durchschnittlicher Wert. Weil die Laufzeit dieser Vorgehensweise entsprechend verlängert wird, wird im Vergleichstest nur die Laufzeit einer Durchführung genutzt.

Beim ABL-Ansatz endet die Korpusverarbeitung mit den konstruierten Treebanks, durch deren Hilfe der Vergleichstest und die Evaluation erfolgen, nach dem Selektion-Lernen.

Weil die in der vorliegenden Arbeit zu induzierten Grammatiken anders als die beim ABL extrahierten Grammatiken sind, wird die Phase „Grammatiken-Induktion“ nur beim EDSI-Ansatz implementiert. Daher gibt es weder einen Vergleichstest noch eine quantitative Evaluation für diese Phase.

4.2.3 Umgebung des EDSI-Ansatzes

Der in dieser Arbeit entwickelte EDSI-Ansatz ist im Wesentlichen ähnlich wie der ABL-Ansatz designt und besteht aus vier Phasen. Im Gegensatz zum ABL-Ansatz werden beim EDSI-Ansatz insgesamt vier Tochtersysteme geboten.

---

[72] Instanz „default“ verwendet die Standardwerte der Operationskosten im Algorithmus: Einfügen und Lösen kosten 1, Ersetzen hat die Kosten 2.

[73] Instanz „incr“ bedeutet, dass die früher gefundenen Hypothesen als die richtigen Konstituenten aufgenommen werden. Die nachher gelernten Hypothesen werden ignoriert und nicht mehr in die Fuzzy-Treebank hinzugefügt.

[74] Siehe van Zaanen (2001, 67). Diese Methode wird „random“ genannt.

(1) Indexieren

(2) Greedy-Lernen;

(3) Selektion-Lernen und

(4) *Grammatiken-Extraktion* (zusätzlich)

Das Indexieren ist eigentlich eine Vorbereitungsphase, die keine Baumstrukturen produziert, sondern nur die benötigen Matrizes erstellt. Der Zeitaufwand dieser Phase ist im Vergleich zu den anderen drei Phasen so gering dass er beinahe ignoriert werden darf.

Für das Greedy-Lernen werden zwei Modi entwickelt, die sich auf die unterschiedlichen Aufgabenziele beziehen. Das „*Modus 1*“ ist der Standardmodus zur Erstellung der Fuzzy-Treebanks im Greedy-Lernen. Das „*Modus 2*“ dient dazu, die gefundenen äquivalenten Hypothesen, die weiterhin für die Induktion der lokalen Grammatiken eingesetzt werden, möglichst korrekt zu gruppieren. Da zwei statistische Strategien im Selektion-Lernen integriert sind, entstehen hier wiederum zwei unterschiedliche Tochtersysteme (siehe Abbildung 4.3).

In der letzten Phase werden die lokalen Grammatiken auf der Basis der Treebanks automatisch vorgeschlagen und manuell erstellt.

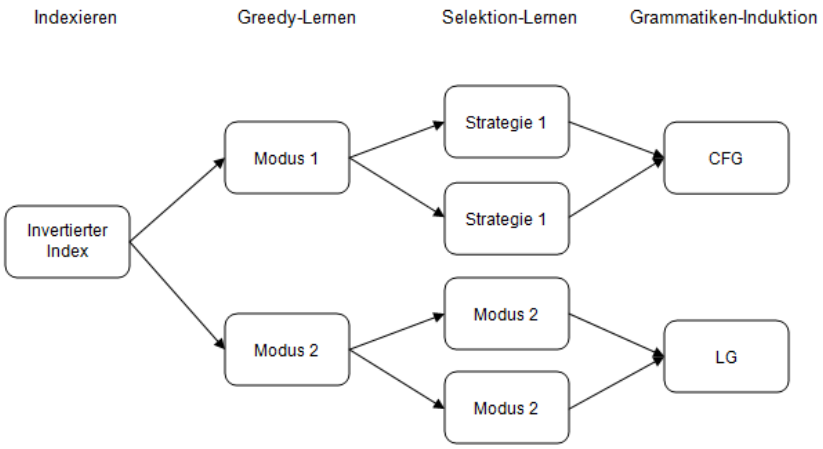


Abbildung 4.3: EDSI Systeme

Die ersten Instanzen jeder einzelnen Phase (*Modus 1 und Strategie 1*) werden aufgenommen und ergänzen das Standardsystem. Des Weiteren werden auch die Subsysteme (die anderen drei Pfaden) durch die Instanzen gebildet.

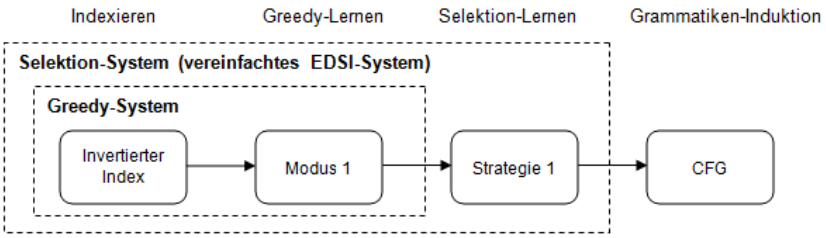


Abbildung 4.4: EDSI Subsysteme

Die Kombination des „Indexierens“ mit dem „*Modus 1*“ im Greedy-Lernen bildet das „Greedy-System“, das äquivalent zum „Alignment-Lernen“ im ABL-Ansatz ist. Durch die Ergänzung des „Selektion-Lernens“ entsteht das „Selektion-System“, das als ein EDSI-System ohne die Grammatik-Induktion betrachtet werden kann. Die beiden Subsysteme werden in der Untersuchung getestet und evaluiert.

Die im Selektion-Lernen entwickelten zwei Auswahlstrategien basieren auf der Wahrscheinlichkeit des Auftritts, die jeweils durch die Konstituentenfrequenzen im Korpus und die Wortsequenzen in den untersuchten Korpora berechnet wird. Da die Frequenzen ggf. gleich sein können, besteht die Gefahr, dass die Resultate bei mehrfacher Durchführung bis zu einem gewissen Grad variieren können. Diese Gleichheit der Frequenzen kann jedoch durch zwei externe allgemeine Frequenzlisten reduziert werden. Die Stabilität des EDSI-Systems wird ebenfalls auf den beiden Korpora getestet und bewertet (siehe Kapitel 4.3.2.3).

Nach dem Selektion-Lernen werden die kontextfreien Grammatikregeln automatisch durch das System vorgeschlagen, die danach durch die Ergänzung von äquivalenten Hypothesen manuell in die lokalen Grammatiken überführt werden.

4.3 Resultate und Evaluation

Der EDSI-Ansatz wird auf den Korpora der Tübingen-Treebanks in verschiedenen Kontexten getestet und quantitativ evaluiert, sodass der Ansatz unter verschiedenen Bedingungen bewertet werden kann. Zunächst findet eine Analyse für die Zwischenresultate aus dem Greedy-Lernen statt. Dabei werden die Stärke und auch die Schwäche dieses Lernens durch den Vergleichstest deutlich dargestellt. Nach dem Selektion-Lernen ergeben sich die finalen Treebanks, die eine zentrale Rolle bei der Grammatik-Induktion spielen. Daraus resultiert die end-

gültige quantitative Bewertung des gesamten EDSI-Ansatzes. Weil die Flexibilität der Wortstellung sich auf den Ansatz negativ auswirken kann, wird die Evaluation in zwei unterschiedlichen Sprachen durchgeführt, um ihre Resultate zu vergleichen.

4.3.1 Evaluation: Greedy-System

Das Greedy-Lernen versucht möglichst viele (korrekte) Hypothesen aus dem Korpus zu finden und den Fuzzy-Treebanks hinzuzufügen. Durch die Erhöhung der Anzahl der gefunden Hypothesen kann der Recall steigen. Es besteht jedoch das Risiko, dass auch manche falsche Hypothesen in den Treebanks aufgenommen werden, die die gesamte Präzision negativ beeinflussen. In diesem Fall muss entweder eine bessere Extraktionsmethode entwickelt oder eine geeignete nachkommende Verfeinerungsmethode gefunden werden. Das Greedy-Lernen wird auf den Tübingen-Korpora, die jeweils aus Sätzen im Englischen und im Deutschen bestehen, getestet.

4.3.1.1 Testergebnis: Vergleich der Effizienz

Das Greedy-Lernen hat den größten Beitrag zur Effizienzerhöhung des EDSI-Ansatzes geleistet. Weil das Greedy-System die Kombination der Phasen „Indexieren“ und „Greedy-Lernen“ ist, besteht entsprechend auch der Zeitaufwand des Systems aus zwei Phasen. Wie in den Kapiteln 3.2.1 und 3.3.2 analysiert, wird die Laufzeit des Greedy-Systems durch die folgenden Faktoren beeinflusst:

$$
\begin{aligned}
T_{Greedy\text{-}System} &= T_{Indexieren} + T_{Greedy\text{-}Lernen} \\
&= T_{Indexieren} + T_{Term\text{-}Extaktion\ \&\ Patternextraktion} + T_{Hypothesenextraktion} \\
&= m * n + i\,j * m\,n + m * r * s
\end{aligned}
$$

(wobei: $1 \le i \le m;$

$1 \le j \le n;$

$1 \le r \le 0{,}5n^2$ und

$1 \le s \le n$

m ist die Satzanzahl, n ist die durchschnittliche Satzlänge.).

Die Einschätzungsformulierung der Verarbeitungszeit im schlimmsten Fall kann wie folgt vereinfacht werden:

$$T_{Greedy\text{-}System} = mn + m^2n^2 + 0{,}5mn^3$$

In diesem Fall ist $i = m$. Dies bedeutet, dass die im Target-Satz vorkommenden Terme auch in allen Source-Sätzen auftreten. Gleichzeitig muss $j = n$ sein. Das heißt, jeder Source-Satz enthält alle Terme des Target-Satzes, die genau so viele Patterns wie ihre Anzahl konstruieren. Genauer gesagt, sind die Terme entweder in jedem Source-Satz identisch oder stehen in der umgekehrten Reihenfolge des Target-Satzes.

Im besten Fall (wenn $i = 1$ und $j = 1$) kann die Abschätzung noch einfacher formuliert werden:

$$T_{Greedy\text{-}System} = mn + mn + m$$

Im Vergleich zum schlechtesten Fall wird der Zeitaufwand mn Male reduziert. Z.B. wenn das eingegebene Korpus aus 1000 Sätzen besteht, deren durchschnittliche Länge 10 Wörter beträgt, macht die Laufzeit im besten Falle ungefähr 0,02% der Zeit aus, die der schlimmste Fall benötigt.

Hier ist zu erkennen, dass die gebrauchte Zeit des Indexierens ($T_{Indexieren} = mn$) relativ gering ist. Insbesondere wenn das Korpus sehr umfangreiche Sätze enthält, ist der Zeitaufwand des Indexierens in der gesamten Verarbeitungszeit verhältnismäßig sehr kurz.

Im Vergleich zum Greedy-System besteht die äquivalente Phrase des klassischen ABL-Ansatzes aus zwei Teilen:

$$\begin{aligned} T_{Alignment\text{-}Lernen} &= T_{Patternextraktion} + T_{Hypothesenextraktion} \\ &= m^2n^2 + m * r * s \end{aligned}$$

(wobei: $1 \leq r \leq 0{,}5n^2$ und $1 \leq s \leq n$;
m ist die Satzanzahl, n ist die durchschnittliche Satzlänge.).

Der zweite Bestandteil „$T_{Hypothesenextraktion}$“ ist in beiden Systemen identisch. Die normale Laufzeit der Pattern-Extraktion im klassischen Ansatz ist relativ konstant und ist quasi gleich dem schlimmsten Fall beim Greedy-Lernen.

Die durchschnittliche Satzlänge (*n*) der natürlichen Sprachen ist im Allgemeinen limitiert (z.B. ist die Durchschnittslänge der in dieser Untersuchung verwendeten Tübingen-Korpora im Englischen und im Deutschen jeweils 11,6 und 11,7). Vergleichsweise ist die Satzanzahl (*m*) stärker gewichtet als die Satzlänge (*n*), wenn es sich um ein großes Korpus handelt. D.h. der Zeitaufwand $T_{Patternextraktion}$ lässt sich stärker von der Satzanzahl beeinflussen. Eine Verkürzung der Verarbeitungszeit in dieser Phase wird zu einer Erhöhung der gesamten Systemeffizienz führen. Basierend auf dieser Grundidee wird im weiteren Verlauf der EDSI-Ansatz entwickelt.

Test auf dem englischen Korpus

Das englische Korpus besteht aus 2275 Sätzen, die der Tübingen-Treebank entnommen werden. Um den Einfluss der Korpusgröße zu untersuchen, werden die Systeme mit verschiedenen Mengen von Sätzen getestet. Das Greedy-System und sein referenziertes Alignment-System wurden an den Korpora mehrfach durchgeführt[75]. Aus den Resultaten wurde die jeweils durchschnittliche Laufzeit berechnet.

Satzanzahl	Laufzeit T_{EDSI}(s)	Laufzeit T_{ABL}(s)	T_{ABL}/T_{EDSI}-Verhältnis
200	4,63	31,26	7
600	23,57	284,05	12
1000	64,17	847,17	13
1400	134,04	1755,84	13
1800	217,17	2866,31	13
2200	319,80	4308,36	13

Tabelle 4.1: Vergleich der Laufzeit des Greedy-Systems (EN)

Der Vergleich zwischen der Laufzeit in der obigen Tabelle zeigt einen sehr deutlichen Unterschied zwischen den beiden Systemen. Der Zeitaufwand des Alignment-Lernens steigt rasant an. Nachdem die Satzanzahl des getesteten Korpus von 200 auf 2200 um das 10-Fache vergrößert wird, steigt die Laufzeit des Alignment-Systems über 140-mal. Dahingegen verlängert sich die gebrauchte Verarbeitungszeit des Greedy-Systems weniger als 60-mal. Dieser enor-

[75] Weil die laufenden Systemprozesse auf einem Rechner auch Ressourcen verbrauchen, kann die echte Laufzeit eines Programms schwer zu ermitteln. Um die Störung der externen Faktoren auszuschließen, wird der Test mehrfach durchgeführt, um die Durchschnittslaufzeit zu berechnet.

me Unterschied zeigt an, dass die Steigerung des Zeitaufwands beim Greedy-System wesentlich geringer als beim Alignment-System ist.

Das T_{ABL}/T_{EDSI}-*Verhältnis* ist relativ konstant und variiert meistens zwischen 12 bis 13 (abgesehen von den ersten zwei Punkten). Es liegt womöglich daran, dass die nicht festgelegten Parameter (z.B. *i* und *j* etc.) tendenziell stabil werden, wenn die Satzanzahl eine ausreichende Größe erreicht.

Test auf dem deutschen Korpus

Da die Satzstellung eine wichtige Rolle in der Grammatik-Induktion spielt, ist es notwendig das System mit einer flexiblen natürlichen Sprache (in der vorliegenden Untersuchung wird die deutsche Sprache als Beispiel aufgenommen) zu testen. Das deutsche Korpus stammt ebenfalls aus der Tübingen-Treebank und enthält insgesamt 5559 Sätze. Die zwei Systeme wurden wieder mehrfach durchgeführt, um die durchschnittliche Laufzeit des jeweiligen Systems zu berechnen.

Satzanzahl	Laufzeit T_{EDSI}(s)	Laufzeit T_{ABL}(s)	T_{ABL}/T_{EDSI}-Verhältnis
500	24,45	225,65	9
1500	134,98	2012,68	15
2500	362,44	5571,28	15
3500	700,15	11152,15	16
4500	1160,53	18170,07	16
5500	1708,92	27879,23	16

Tabelle 4.2: Vergleich der Laufzeit des Greedy-Systems (DE)

Diese Laufzeit-Tabelle zeigt ebenfalls einen deutlichen Unterschied des Zeitaufwands zwischen den Systemen. Das T_{ABL}/T_{EDSI}-*Verhältnis* liegt meistens zwischen 15 und 16 (abgesehen von den ersten zwei Punkten).

Die Vergleichstests in zwei Sprachen geben zu erkennen, dass das neu entwickelte Greedy-System eine wesentliche Verringerung der Verarbeitungszeit für die natürlichen Sprachen erzielt. Darüber hinaus wird ermöglicht, eine flach (oder tief) syntaktische Analyse auf den großen Korpora durchzuführen.

4.3.1.2 Testergebnis: Bewertung der Treebanks

Durch das Greedy-Lernen werden alle möglichen Hypothesen im Korpus gefunden und in der Fuzzy-Treebank abgespeichert. D.h. die Fuzzy-Treebank enthält nicht nur die richtigen Konstituenten, die in den originalen Treebanks vorkommen, sondern auch falsche Hypothesen, die beim nachfolgenden Selektion-Lernen entfernt werden sollen. Da es sich beim Selektion-Lernen nur um die Auswahl der gelernten Hypothesen handelt, können keine neuen Hypothesen gefunden werden. Das bedeutet, dass die beim Greedy-Lernen nicht gelernten Konstituenten nicht in den endgültigen Fuzzy-Treebanks existieren können. Daher versucht der EDSI-Ansatz von Anfang an, möglichst viele Hypothesen zu extrahieren und so mehr Konstituenten abzudecken. Ein möglicher Nachteil dieser Idee ist, dass der Fuzzy-Treebank sehr viele falsche Hypothesen hinzugefügt und beim Selektion-Lernen nicht aussortiert werden können. Dies wird beim Selektion-Lernen untersucht.

Die durch das Greedy-Lernen auf den zwei Tübingen-Korpora erstellten Fuzzy-Treebanks werden quantitativ evaluiert. Deren Resultate werden im Folgenden präsentiert. Unter „Gold-Standard“ ist die Anzahl der Konstituenten in der originalen Tübingen-Treebank zu finden. Die genaue Anzahl der durch das Greedy-Lernen korrekt und falsch gelernten Hypothesen werden im weiteren Verlauf aufgelistet. Recall bezeichnet, wie viele korrekte Hypothesen im Verhältnis zu der Gold-Standard-Treebank gelernt werden.

Test auf dem englischen Korpus

Das Greedy-System wird mit unterschiedlichen Datenmengen getestet.

Satzanzahl	Gold-Standard	Korrekt Gelernte	Falsch Gelernte	Recall
200	3359	1641	3942	48,85%
600	10151	5832	14727	57,45%
1000	17356	10778	27774	62,10%
1400	25062	16444	45349	65,61%
1800	32196	21607	59946	67,11%
2200	39459	26974	75218	68,36%

Tabelle 4.3: Gelernte Hypothesen durch das Greedy-System (EN)

Während die Datenmenge wächst, steigt entsprechend der Recall. Dies erfüllt die Erwartung des statistischen Ansatzes, dass ein umfangreiches Korpus zur Entdeckungsquote der Konstituenten beitragen kann.

Allerdings ist nicht zu übersehen, dass dadurch immer mehr falsche Hypothesen gelernt und in der Fuzzy-Treebank abgespeichert werden. Diese Hypothesen müssen beim nachkommenden Selektion-Lernen möglichst aussortiert werden, damit eine hohe Präzision erreicht werden kann. Die Steigerung der falschen Hypothesen beträgt 19. Sie ist leicht höher als die Steigerung der korrekten Hypothesen (16) und wird in der vorliegenden Untersuchung als akzeptabel betrachtet.

Test auf dem deutschen Korpus

Die Arbeitsweise der Tests auf dem deutschen Korpus und die Arbeitsweise auf dem englischen Korpus sind identisch.

Satzanzahl	Gold-Standard	Korrekt Gelernte	Falsch Gelernte	Recall
500	8580	4108	12301	47,87%
1500	26089	15125	44557	57,97%
2500	43557	26999	81827	61,99%
3500	61428	39802	121887	64,79%
4500	78472	52055	159472	66,34%
5500	97427	65855	207549	67,59%

Tabelle 4.4: Gelernte Hypothesen durch das Greedy-System (DE)

Die Evaluation auf dem deutschen Korpus präsentiert ähnliche Ergebnisse wie auf dem englischen Korpus. Das beweist, dass ein solcher statischer Ansatz wie EDSI in der Lage ist, die Grammatiken aus einer Sprache mit flexibler Wortstellung automatisch zu extrahieren.

4.3.2 Evaluation: Selektion-System

Die Evaluation des Greedy-Systems zeigt, dass das Greedy-Lernen nur eine relativ niedrige Präzision erzielen kann. Außerdem können die Baumstrukturen wegen den Überlappungen der Hypothesen in den Fuzzy-Treebanks noch nicht ordentlich konstruiert werden. Aufgrund dessen ist der Einsatz des Selektion-Lernens zur Erzielung besserer Resultate notwendig.

Zusammen mit dem Greedy-System bildet das Selektion-Lernen das Selektion-System. Beim Selektion-Lernen handelt es sich nicht mehr um die Effizienzerhöhung, sondern um die Verfeinerung der Fuzzy-Treebanks. In dieser Phase werden zwei Strategien entwickelt, die auf statistischen Methoden basieren und die dabei helfen, die besseren Hypothesen auszuwählen.

Wenn die nachkommende Verarbeitungsphase „Grammatik-Induktion“ nur als zusätzlicher Teil betrachtet wird, gleicht das Selektion-System einem vereinfachten EDSI-Ansatz. In diesem Sinne ist die Evaluation des Selektion-Systems nichts anderes als die quantitative Bewertung für das EDSI-System.

Das Selektion-System wurde wie beim Greedy-System auf den zwei Tübingen-Korpora getestet. Die dadurch erstellten Fuzzy-Treebanks T werden im Folgenden quantitativ evaluiert und dargestellt.

4.3.2.1 Testergebnis: Vergleich der Effizienz

Die Effizienz der Grammatik-Induktion zu erhöhen, ist das Hauptziel der vorliegenden Arbeit. Die durch den EDSI-Ansatz gewonnenen Ergebnisse werden in diesem Kapitel quantitativ analysiert und mit dem klassischen Ansatz ABL verglichen.

Das Selektion-System (oder das EDSI-System) besteht aus dem Greedy-System und dem Selektion-Lernen, die zusammen den gesamten Zeitaufwand verursachen. Eine Einschätzung der Verarbeitungszeit kann daher wie folgt formuliert werden:

$$
\begin{aligned}
T_{EDSI} &= T_{Greedy\text{-}System} + T_{Selektion\text{-}Lernen} \\
&= T_{Indexieren} + T_{Greedy\text{-}Lernen} + T_{Selektion\text{-}Lernen} \\
&= T_{Indexieren} + T_{Term\text{-}Extaktion\ \&\ Patternextraktion} + T_{Hypothesenextraktion} + \\
&\quad T_{Konstituente\text{-}Selektion} \\
&= m * n + i\,j * m\,n + m * r * s + m * u * v
\end{aligned}
$$

(wobei: $1 \le i \le n;\ 1 \le j \le n;$

$1 \le r \le 0{,}5n^2$ und $1 \le s \le n;$

$1 \le u \le 0{,}5n^2$ und $1 \le v \le u;$

m ist die Satzanzahl, n ist die durchschnittliche Satzlänge.).

Der Zeitaufwand des schlimmsten Falls kann mit der folgenden Formulierung eingeschätzt werden:

$$T_{Greedy\text{-}System} = mn + m^2n^2 + 0{,}5mn^3 + 0.25mn^4$$

Wobei die variierbaren Parameter ihre größten Werte bekommen: $i = m$, $j = n$, $r = 0{,}5n^2$, $s = n$ und $u = v = 0{,}5n^2$.

Die Einschätzung des besten Falls kann folgendermaßen angegeben werden:

$$T_{Greedy\text{-}System} = mn + mn + m + m$$

Das Kernstück des Zeitaufwands (das Greedy-Lernen) wird auf *2mn* reduziert. Wie in Kapitel 4.3.1.1 diskutiert, ermöglicht diese starke Reduzierung eine enorme Erhöhung der Effizienz des gesamten Systems.

Der Zeitaufwand des referenzierten klassischen ABL-Systems besteht aus den folgenden Teilen:

$$\begin{aligned} T_{ABL} &= T_{Alignment\text{-}Lernen} + T_{Selektion\text{-}Lernen} \\ &= T_{Patternextraktion} + T_{Hypothesenextraktion} + T_{Konstituente\text{-}Selektion} \\ &= m^2n^2 + m * r * s + m * u * v \end{aligned}$$

(wobei: $1 \le r \le 0{,}5n^2$ und $1 \le s \le n$;

$1 \le u \le 0{,}5n^2$ und $1 \le v \le u$;

m ist die Satzanzahl, n ist die durchschnittliche Satzlänge).

Die zwei Ansätze und das Selektion-Lernen beruhen auf unterschiedlichen Entscheidungsstrategien, die jeder Hypothese in der Fuzzy-Treebank einen Wahrscheinlichkeitswert zuschreiben. Abgesehen von diesem Unterschied wird derselbe Algorithmus verwendet. Daher ist die Formulierung der Zeitaufwandeinschätzung des Selektion-Lernens identisch.

Wenn das Korpus ausreichend groß ist (d.h. m ist viel größer als n), bestimmt das Greedy-Lernen die Effizienz des gesamten Systems. Das heißt, die Effizienzerhöhung ist vom Greedy-Lernen abhängig. Die entsprechende Phase (das Alignment-Lernen) im ABL-System ist dem schlimmsten Fall im EDSI-System quasi identisch. In Anbetracht der Abschätzung des

besten Falls kann theoretisch und potenziell eine Erhöhung zwischen dem Greedy-Lernen und Alignment-Lernen berechnet werden; der Wert liegt im Bereich zwischen *0* und *0,5mn*. Allerdings sind die extremen Werte in der Praxis nur sehr unwahrscheinlich zu erreichen.

Die in den Kurven verwendeten Werte stellen die durchschnittliche Laufzeit mehrerer Durchführungen dar, wobei eine mögliche externe Störung ausgeschlossen werden kann.

a) Test auf dem englischen Korpus

Der Test wurde mit verschiedenen Datenmengen (mit einem Intervall von 200 Sätzen) auf dem englischen Korpus durchgeführt. Die Kurven der Laufzeit werden in der folgenden Abbildung dargestellt:

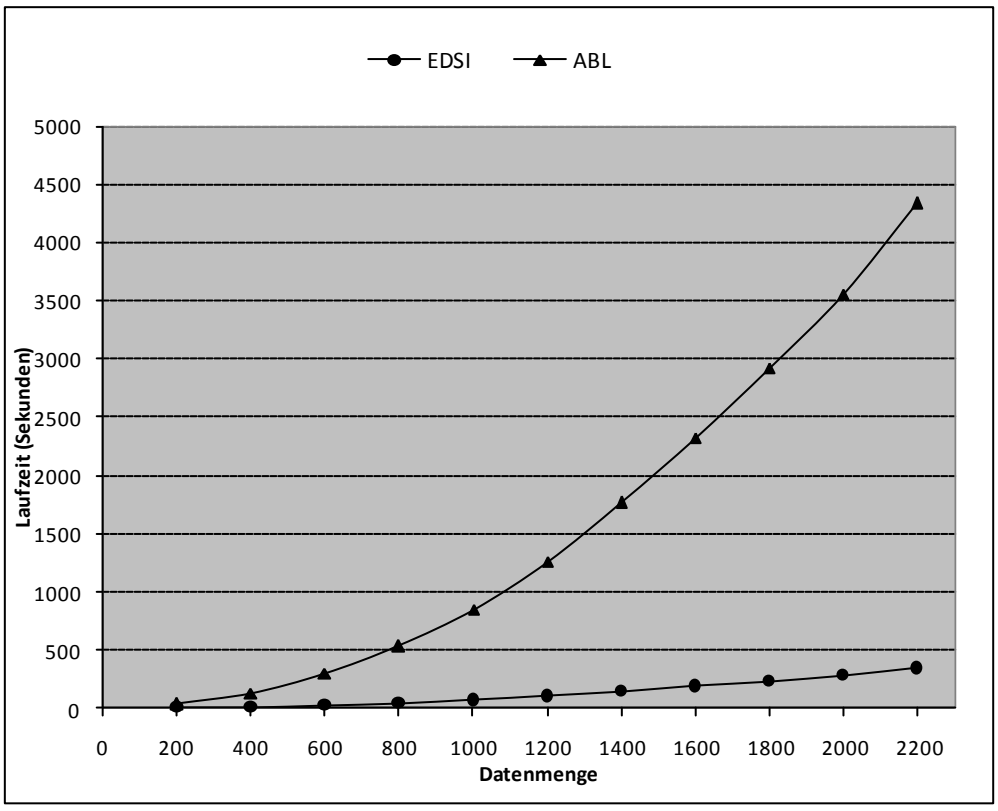


Abbildung 4.5: Zeitaufwand-Kurve (EN)

Hier ist der Unterschied zwischen den beiden Systemen sehr deutlich zu erkennen. Der Wert vom T_{ABL}/T_{EDSI}-*Verhältnis* liegt zwischen 6 und 13. Wie beim Greedy-System tendiert dieser Wert ab dem vierten Punkt (Datenmenge 800) 12 oder 13 zu sein, da die variierbaren Faktoren ab dieser Datengröße relativ konstant werden. Aber je mehr Sätze das eingegebene Kor-

pus enthält, desto größer ist der absolute Unterschied. D.h. der EDSI-Ansatz ist vorteilhafter bei der Verarbeitung von großen Korpora.

b) Test auf dem deutschen Korpus

Der Test wurde auch auf dem deutschen Korpus mit verschiedenen Datenmengen (mit einem Intervall von 500 Sätzen) durchgeführt. Die folgende Abbildung zeigt die Zeitaufwand-Kurven von EDSI- und ABL-Ansätzen.

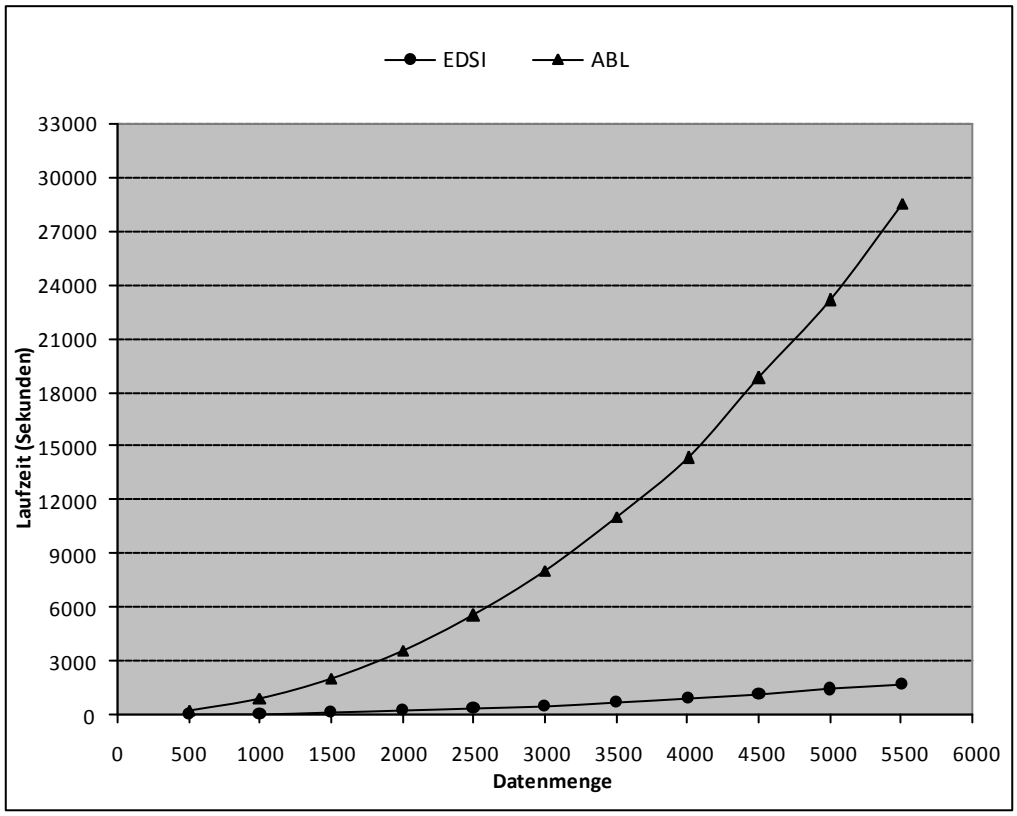


Abbildung 4.6: Zeitaufwand-Kurve (DE)

Die Zeitaufwand-Kurven zeigen ein ähnliches Phänomen wie die des englischen Korpus: die Effizienzerhöhung des EDSI-Ansatzes ist sehr deutlich. Der Wert vom T_{ABL}/T_{EDSI}-*Verhältnis* liegt zwischen 9 und 17, tendiert aber erst ab der Datenmenge 1500 dazu, konstant zu bleiben. Dieses Verhältnis ist generell etwas größer als auf dem englischen Korpus. D.h. der EDSI-Ansatz ist von den Eigenschaften der Sprachen abhängig. Ansonsten spielt eventuell auch die Datenmenge eine Rolle, da der EDSI-Ansatz gerade bei der Verarbeitung großer Korpora vorteilhaft ist.

4.3.2.2 Testergebnis: Bewertung der Resultate

Der Test wurde auf unterschiedlichen Satzmengen durchgeführt, um den Einfluss der Korpusgröße zu untersuchen. Um die Performance zu bewerten, werden dabei die drei Systeme *EDSI-S1* (Strategie 1), *EDSI-S2* (Strategie 2) und *ABL* getestet und miteinander verglichen. Da die Resultate des ABL-Systems von der Reihenfolge der Sätze abhängig sein können, werden die eingegebenen Korpora bei jedem Test mit einer zufälligen Reihenfolge neu sortiert. So lässt sich der Effekt leicht an den Ergebnissen erkennen.

Wie in Kapitel 4.1.1 vorgestellt, wurden drei wichtige Messungsmaße beim Test benötigt. Recall bezeichnet, wie viele korrekte Hypothesen im Verhältnis zu der Gold-Standard-Treebank gelernt werden. Die Präzision deutet an, wie viele Hypothesen in der gesamten Treebank korrekt sind. Der F1-Wert ist der harmonische Wert der beiden Maße, in dem der Recall und die Präzision gleich gewichtet werden.

a) Auf dem englischen Korpus

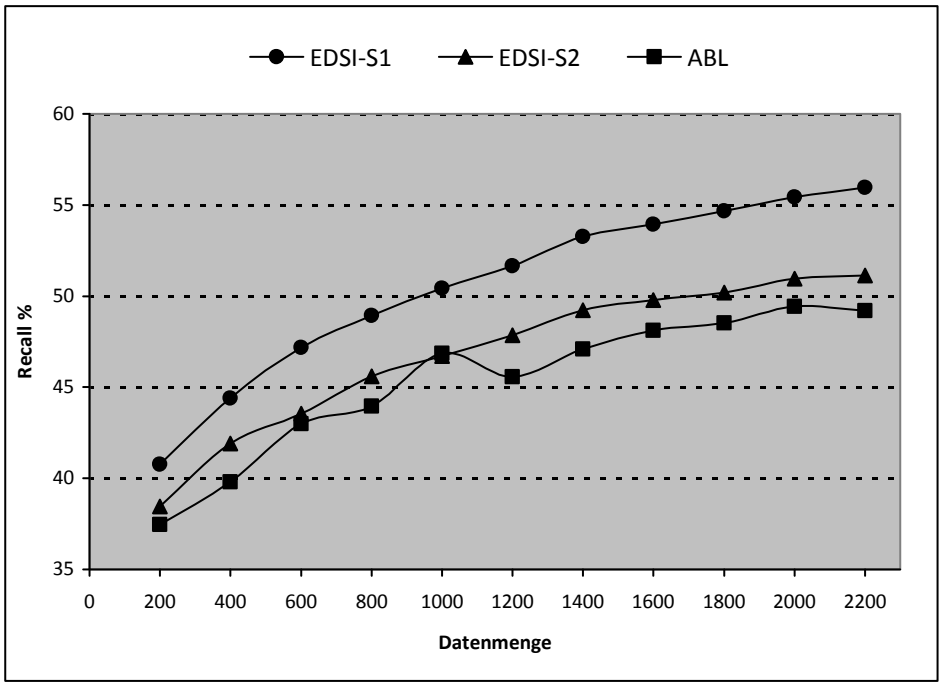


Abbildung 4.7: Recall (EN)

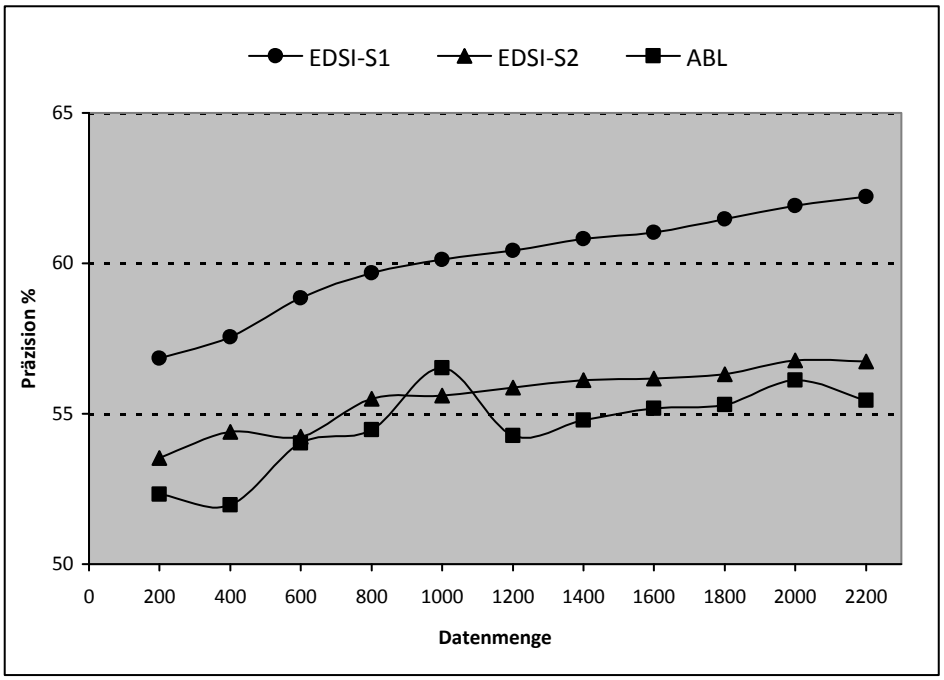


Abbildung 4.8: Präzision (EN)

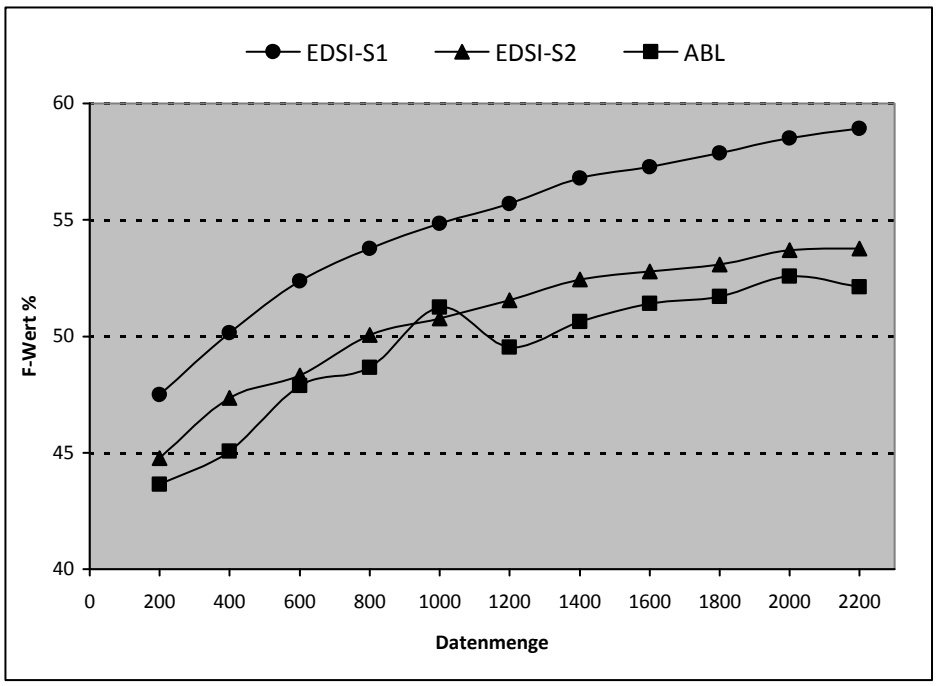


Abbildung 4.9: F1-Wert (EN)

Hier zeigt sich, dass alle Kurven mit Zunahme der Datenmenge steigen. Dies erfüllt die Erwartungen des statistischen Ansatzes. Allerdings verlangsamt sich diese Steigerung und bleibt

bis zu einer bestimmten Datenmenge konstant. Interessant ist, dass der Recall und die entsprechende Präzision der drei Kurven sich sehr ähnlich verhalten. Wo ein hoher Recall erreicht wird, erzielt auch die Präzision einen relativ hohen Wert.

Das System EDSI-S1 liefert die besten Ergebnisse und wird als das Default-System des EDSI-Ansatzes erkannt. Das System EDSI-S2 kann im Allgemeinen die besseren Ergebnisse im Vergleich zum ABL-System liefern, die jedoch nicht so signifikant wie die Ergebnisse aus Strategie 1 sind.

Da die Satzreihenfolge beim Default-ABL System eine wichtige Rolle spielt, sind die Ergebnisse aus unterschiedlichen, zufällig sortierten Korpora ziemlich instabil. Im besten Fall kann das ABL-System bessere Ergebnisse als das System EDSI-S2 (z.B. bei Datenmenge 1000) liefern.

b) Auf dem deutschen Korpus

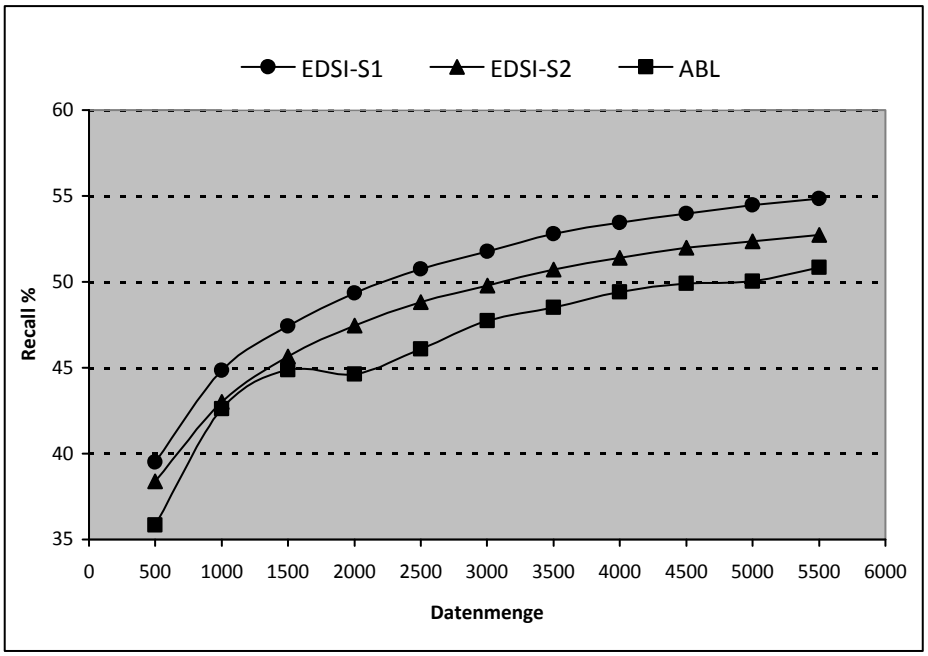


Abbildung 4.10: Recall (DE)

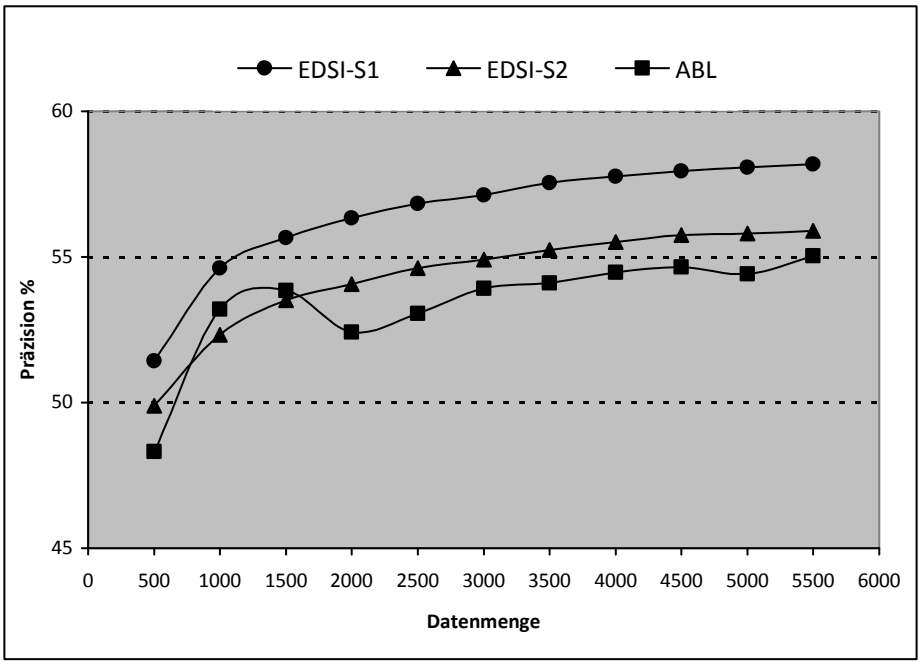


Abbildung 4.11: Präzision (DE)

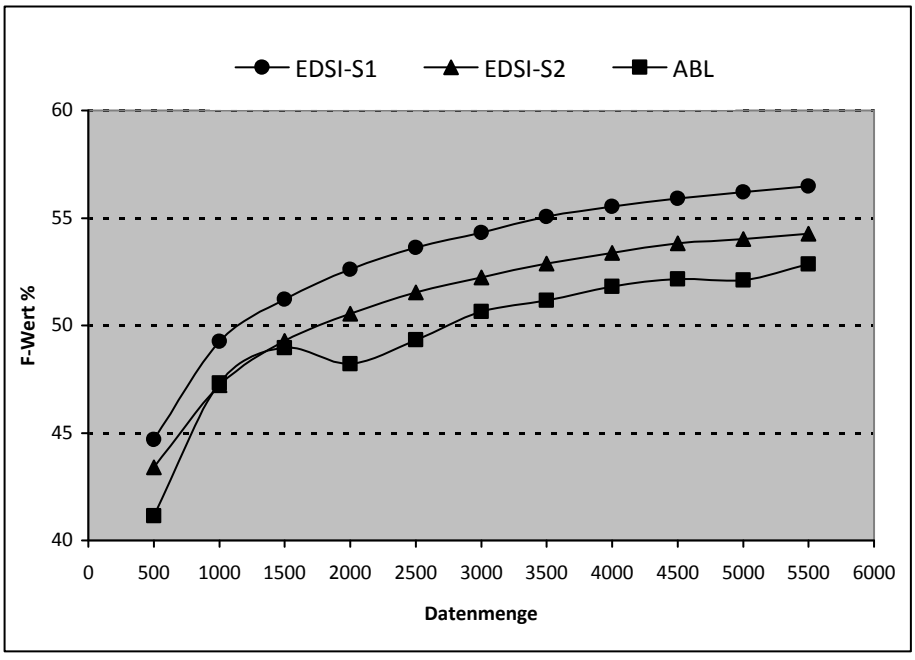


Abbildung 4.12: F1-Wert (DE)

Die obigen Kurven der drei Systeme ähneln den Testkurven des englischen Korpus‘. Das EDSI-S1 System erreicht wie bereits zuvor die beste Bewertung. Die EDSI-S2 und ABL Kurven überschneiden sich wieder teilweise.

Allerdings ist zu beachten, dass die Flexibilität der Wortstellung ebenfalls einen gewissen Einfluss auf die Resultate hat. Die allgemeine Präzision auf dem deutschen Korpus ist etwas niedriger (ungefähr 5%) als die Präzision auf dem englischen Korpus, obwohl das deutsche Korpus wesentlich mehr Sätze enthält.

Außerdem zeigt ein Vergleich zwischen den Greedy und Selektion Systemen, dass der Recall vom Selektion-System auf beiden Korpora ungefähr 10% kleiner als der des Greedy-Systems ist. Während das Selektion-Lernen viele falsche Hypothesen ausfiltert, werden teilweise auch richtige Hypothesen verworfen.

4.3.2.3 Testergebnis: Stabilität des Systems

Die Auftrittswahrscheinlichkeit der Hypothesen ist die Basis der zwei im EDSI-System intergerierten Strategien. Diese Wahrscheinlichkeit wird jeweils durch die Wort- (S1) und Hypothesen-Frequenz (S2) kalkuliert. Da die Frequenzen der jeweiligen Objekte ggf. identisch sind, kommt ihren Hypothesen entsprechend dieselbe Wahrscheinlichkeit zu. Ohne den Unterschied zwischen den Wahrscheinlichkeiten ist es für das EDSI-System nicht mehr möglich eine klar bessere Entscheidung zu treffen. In diesem Fall wird durch das Programm zufällig eine der konkurrierenden Hypothesen als der beste Kandidat ausgewählt. Dies könnte zu dem Problem führen, dass die Resultate wegen der zufälligen Auswahl ihre Stabilität verlieren.

Das Default-System EDSI-S1 wurde evaluiert, um dessen Stabilität zu bewerten. Die in der vorliegenden Arbeit vorgeschlagene Lösung zu diesem Problem basiert auf der statistischen Methode, dass es in jeder natürlichen Sprache eine allgemeine Wortfrequenzliste gibt. Wenn ein Korpus ausreichend umfangreich (viele Texte aus verschiedenen Themen) ist, kann daraus die Liste der jeweiligen Sprache resultieren. Je höher die Frequenz ist, desto weniger Wörter existieren, die eine identische Frequenz besitzen.

Die Frequenzliste aus dem Tübingen-Korpus wird mit der allgemeinen Liste kombiniert, sodass eine neue Liste erstellt werden kann, die einerseits das Wortgewicht im allgemeinen Kontext und anderseits das Gewicht im Korpus beschreibt. Eine Analyse stellt die Verteilung der neuen Frequenzliste (d.h. die Frequenzen der Wortfrequenzen in der neuen Liste) wie folgt dar:

Bereich	≥ 1	≥ 2	≥ 5	≥ 10	≥ 20	≥ 40
Frequenzverteilung (EN)	725	220	35	7	1	1
Frequenzverteilung (DE)	1450	807	92	36	15	6

Tabelle 4.5: Verteilung der verallgemeinerten Wortfrequenzen

Die Frequenzverteilung der Wortfrequenz zeigt, dass sich sehr wenige Wörter in dem Bereich befinden, in dem die Frequenzen hoch (z.B. ≥ 5) und identisch sind. Da in diesem Fall im Englischen nur 4,83% und im Deutschen 6,34% der Wortfrequenzen sind, ist der Einfluss des genannten Problems auf die Resultate als sehr gering einzuschätzen.

EN

Test-Nummer	Gelernte Hypothesen	Korrekte Hypothesen	Recall (%)	Präzision (%)
1	36944	23011	56,19	62,28
2	36943	23011	56,19	62,28
3	36943	23016	56,20	62,30
4	36943	23012	56,19	62,29
5	36941	23018	56,20	62,31
6	36945	23023	56,21	62,31
7	36945	23010	56,18	62,28
8	36943	23019	56,20	62,30
9	36944	23028	56,23	62,33
10	36947	23024	56,22	62,31

DE

Test-Nummer	Gelernte Hypothesen	Korrekte Hypothesen	Recall (%)	Präzision (%)
1	93186	54460	55,18	58,44
2	93183	54455	55,18	58,43
3	93188	54463	55,18	58,44
4	93192	54456	55,18	58,43
5	93193	54458	55,18	58,43
6	93190	54460	55,18	58,43
7	93185	54456	56,18	58,43
8	93189	54458	55,18	58,43
9	93189	54464	55,19	58,44
10	93187	54453	55,17	58,43

Tabelle 4.6: Stabilitätstest beim EDSI-System

Das EDSI-System wird auf den zwei Korpora zum Vergleich der Ergebnisse jeweils 10-mal durchgeführt. Die Vergleiche zeigen, dass die Ergebnisse jeder Durchführung von denen der anderen abweichen können. Allerdings ist diese Abweichung so gering, dass sie ignoriert werden darf. Das EDSI-System ist daher stabil.

4.4 Induktion der lokalen Grammatiken

Die in der vorliegenden Arbeit induzierten lokalen Grammatiken beschreiben einerseits syntaktische Strukturen und andererseits Wortgruppen. Entsprechend kann neben der Fuzzy-Treebank T, in der sich alle Baumstrukturen befinden, auch eine Fuzzy-Treebank S, die alle funktional äquivalenten Hypothesen enthält, durch das EDSI-System erstellt werden. Somit bilden die beiden Treebanks die Grundlage für den Aufbau lokaler Grammatiken. Im Gegensatz zur quantitativen Evaluation für die Treebank T in den vorstehenden Kapiteln wird die Fuzzy-Treebank S in diesem Kapitel qualitativ evaluiert.

4.4.1 Eine „looks good“ Evaluation

Eine grundlegende Annahme der vorliegenden Arbeit ist, dass die äquivalenten Konstituenten oft in einem ähnlichen Kontext vorkommen. Durch die Extraktion der Slots in den Patterns können die Konstituenten gefunden und klassifiziert werden. Allerdings werden die Konstituenten durch diese Maßnahme manchmal falsch identifiziert oder in falsche Gruppen eingeordnet.

Die Hauptursachen der Problemfälle sind:

- **Abweichende Wortstellung**

  Die Abweichung der Wortstellung ist von der Satzart und der Flexibilität der Sprache abhängig. Diese Abweichung führt – wie das folgende Beispiel zeigt – die Extraktion meistens zu falschen Resultaten:

We	**are**	going to get there on Tuesday.
How	**are**	you coming to Hanover?
On Tuesday we	**are**	going to get there.

  Durch das Pattern “*are*” werden die drei Hypothesen, die offensichtlich nicht funktional äquivalent sind, zu einer Gruppe klassifiziert. Dieses Problem kann möglicherweise durch eine getrennte Verarbeitung der jeweiligen Satzarten gelöst werden, da jede Satzart eine eigene (Standard-) Satzform besitzt. Die Problemlösung zur flexiblen Wortstellung ist die Lemmatisierung aller Sätze. Das Ziel aller Maßnahmen ist, das Korpus

zu optimieren und so die Gültigkeit der Extraktion zu gewährleisten. Da diese zwei Themen jedoch nicht Teil der vorliegenden Arbeit sind, soll hier nicht näher darauf eingegangen werden.

- **Konstituenten am Satzende**

  Die Slots am Ende der Patterns werden automatisch als äquivalente Hypothesen betrachtet. Dies führt manchmal zum Problem der „*übergroßen Konstituente*“:

  We are going to get there **on** [Tuesday.]
  They have a meeting **on** [the twenty fifth in Berlin.]

  Die Zweite Konstituente ist zwar nicht falsch, dennoch ziemlich fragwürdig. Die beste äquivalente Konstituente ist „*the twenty fifth*“. In diesem Sinne ist der Teil „*in Berlin*“ überflüssig.

- **Konstituenten am Satzanfang**

  Hier findet sich ein ähnliches Problem: die Slots am Anfang der Patterns werden in derselben Gruppe eingeordnet.

  [We] **are** going to get there on Tuesday.
  [I think maybe you] **are** coming to Hanover?

  Diese übergroße Hypothese „*I think maybe you*“ entsteht durch die Komplexität eines Nebensatzes. Es ist zu bemerken, dass die zwei falsch gruppierten Hypothesen hinsichtlich ihrer Länge sehr stark variieren. Dies betrifft ebenfalls die Hypothesen im letzten Problemfall. Daher wird durch den Vergleich der Länge der Hypothesen eine Lösung für diese zwei Probleme entworfen.

Die Fuzzy-Treebanks S der vorliegenden Untersuchung dienen hauptsächlich dazu, die Wortgruppen, die meistens aus Wörtern oder kurzen Phrasen bestehen, vorzuschlagen und so lokale Grammatiken zu konstruieren. Kleine Konstituenten sind daher wichtiger und werden bevorzugt analysiert. Der Unterschied der Länge wird in der Evaluierung mit verschiedenen Schwellenwerten (engl. *threshold*) getestet. So werden die „guten“ Hypothesen einer Gruppe ausgewählt.

Zur bestmöglichen Klassifizierung der in den Fuzzy-Treebanks S enthaltenen Hypothesen wurden in dieser Untersuchung zwei Auswahlkriterien festgelegt:

- Wenn die Konstituente häufig in einer Gruppe auftritt, gehört sie sehr wahrscheinlich zu dieser.
- Wenn eine Konstituente selten in einer Gruppe auftritt, wird sie mit ihrer referenzierten Konstituente verglichen. Falls der Unterschied ihrer Länge den festgelegten Schwellenwert nicht übersteigt, gehört sie auch zu dieser Gruppe. Ansonsten wird sie als falsch klassifizierte Hypothese aus den Fuzzy-Treebanks S entfernt. Um möglichst viele falsche Hypothesen zu entfernen, aber gleichzeitig möglichst viele richtige Hypothesen auszuwählen, werden die Hypothesen in den Fuzzy-Treebanks S mit harmonischen Werten (Häufigkeit = 2 und Schwellenwert = 5) verfeinert.

4.4.2 Ein Beispiel der Grammatik-Induktion

Aus den bereits verfeinerten Treebanks (Beispiel siehe Anhang 7.2) können kontextfreie Grammatiken mit den zugehörigen Wortgruppen extrahiert werden. Darüber hinaus werden ihre lokalen Grammatiken induziert. Beispielsweise wird der Satz 490 aus dem englischen originalen Korpus verwendet, um die Induktion der lokalen Grammatiken in der vorliegenden Arbeit zu präsentieren.

4.4.2.1 Kontextfreie Grammatiken

Die Baumstruktur in die originalen Treebank wird in Kapitel 3.5 (siehe Abbildung 3.11) dargestellt. Das EDSI-System rekonstruiert eine neue Baumstruktur aus der Fuzzy-Treebank T (siehe Anhang 7.2.1).

In der originalen Treebank werden die Konstituenten durch die standardisierten POS-Tags bezeichnet, die in dem unstrukturierten Korpus nicht vorhanden sind. Stattdessen bekommen die funktionalen Gruppen im EDSI-System nur die zufällig belegten IDs (siehe Abbildung

4.14), weil die extrahierten Hypothesen direkt aus unstrukturierten Korpora stammen und keine zusätzlichen syntaktischen oder semantischen Informationen erhalten.

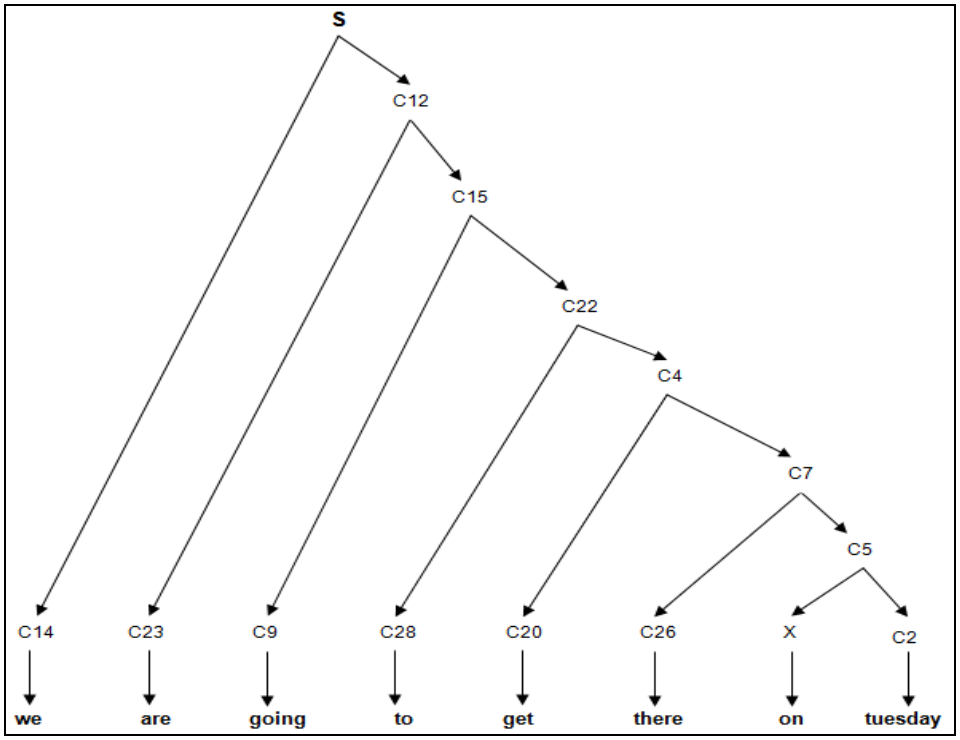


Abbildung 4.13: Baumstruktur im EDSI-System

S	→	NP	VBP	VP	S	→	C14	C12
					C12	→	C23	C15
NP	→	PP						
VP	→	V-PRP	VP		C15	→	C9	C22
V-PRP	→	VBG						
VP	→	TO	VP		C22	→	C28	C4
VP	→	VB	ADVP	PP	C4	→	C20	C7
					C7	→	C26	C5
ADVP	→	RB						
PP	→	IN	NP		C5	→	X	C2
NP	→	NP						
PP	→	we			C14	→	we	
VBP	→	are			C23	→	are	
VBG	→	going			C9	→	going	
TO	→	to			C28	→	to	
VP	→	get			C20	→	get	
RB	→	there			C26	→	there	
IN	→	on			X	→	on	
NP	→	tuesday			C2	→	tuesday	

Abbildung 4.14: Vergleich der kontextfreien Grammatiken

Im Vergleich zu den originalen Grammatiken des Satzes 490 werden die durch das EDSI-System gebildeten kontextfreien Grammatikregeln wie oben dargestellt. Der Vergleich zeigt, dass sich sowohl die rekonstruierte Baumstruktur als auch die extrahierten Grammatiken von den entsprechenden originalen Daten unterscheiden. Obwohl die gefundenen syntaktischen Informationen in diesem Sinne nicht hundertprozentig richtig sind, tragen sie dennoch dazu bei, die Entwicklungskosten der lokalen Grammatiken zu reduzieren.

4.4.2.2 Gruppen

Bevor die verfeinerten Gruppen in die lokalen Grammatiken intergeriert werden, müssen sie zunächst manuell aufgeräumt werden. So wird gewährleistet, dass ausnahmslos jeder Pfad in der Grammatikregel stets richtige Sätze produziert.

Die nach der Aufräumung noch in der Treebank gebliebenen Hypothesen sind die Ausdrücke, die nicht ausgedacht sondern direkt aus dem Korpus extrahiert werden. Daher müssen sie korrekt und in der Lage sein, ihre Sprache zu beschreiben. Je umfangreicher das Korpus ist, desto produktiver sind die Gruppen. Die Wortgruppen des Satzes 490 werden wie in der folgenden Abbildung 4.15 dargestellt (die automatisch erstellte Gruppe siehe Anhang 7.2.2).

NP	C14	we you you and i they all three of them
VBP	C23	are were aren't
VBG	C9	going lying trying willing starting wanting ready free supposed
TO	C28	to
V	C20	get

		go arrive be stay

ADVP	C26	there back here back

IN	X	on

Abbildung 4.15: Wortgruppen im Satz 490

Die von der Treebank vorgeschlagenen Gruppen enthalten verschiedene Formen von Ausdrücken, die funktional äquivalent und in einem Satz austauschbar sind. Beispielsweise besteht die Gruppe C14 aus fünf Ausdrücken, die in den Sätzen dieselbe Funktion tragen. Die Gruppe C28 enthält nur die eigene Konstituente des Satzes 490. Dies entspricht auch der Erwartung der Untersuchung, da „*to*“ in diesem Fall keine äquivalenten Konstituenten haben soll. Das Wort „*on*“ kann vom EDSI-System nicht identifiziert werden und wird daher mit dem Zeichen „*X*“ markiert.

NP	C2	tuesday monday tuesday wednesday thursday friday saturday sunday fridays saturdays monday the eighth thursday the first thursday the second friday the ninth friday the eleventh friday the twelfth friday the twenty first saturday march fifth that friday that wednesday a monday a sunday the day that day the same day

		the first the third the fourth the sixth the nineteenth the fifteenth the twenty ninth the seventeenth the seventh the twenty first the eleventh the twenty fourth the twenty eighth the eighteenth the twenty second the eighth the sixteenth the twenty seventh the twenty fifth the tenth the twenty sixth the thirty first the ninth the twentieth the twelfth the thirteenth the twenty seven etc.

Abbildung 4.16: Wortgruppe "*tuesday*" im Satz 490

Die letzte Gruppe ist beeindruckend erfolgreich. In dieser Gruppe werden über 50 äquivalente Konstituenten eingeordnet, die verschiedenste Datumsformen präsentieren und hier nur teilweise dargestellt werden können. Der besondere Erfolg dieser Gruppe liegt wahrscheinlich daran, dass das vorstehende Wort „*on*“ sehr häufig in den Sätzen vorkommt und als ein Pattern (oder ein Teilpattern) gefunden wird.

Die Einträge in der Gruppe können nach ihrem Format weiter zu Subgruppen klassifiziert werden. Z.B. werden alle Konstituenten im Format „*Wochentag*“ aus den 2275 Sätzen gefunden. Die Konstituenten im Format „*Wochentag* + *Kalendertag*“ sind nicht komplett vorhanden, weil das Korpus wahrscheinlich nicht alle Varianten enthält. Trotzdem ist die Liste sehr hilfreich für den Aufbau lokaler Grammatiken. Denn die fehlenden Konstituenten können durch die Kombination zwischen den ergänzten Wochentagen und Kalendertagen vollständig abgedeckt werden. Dieses Prinzip gilt auch für andere Subgruppen.

Im Vergleich zu einer manuellen Erstellung ist diese automatische Extraktion viel effizienter und mit weniger manuellem Aufwand verbunden. Einerseits werden die Gruppen nicht ausgedacht sondern echten Kontexten entnommen und müssen somit richtig sein. Anderseits werden sie automatisch oder semi-automatisch gruppiert.

4.4.2.3 Lokale Grammatik

Des Weiteren kann für den Satz 490 eine lokale Grammatik induziert werden, die nicht nur den Satz 490 parst, sondern auch ähnliche Sätze korrekt produzieren kann (siehe die Abbildung in Anhang 7.3.2). Lokale Grammatiken sind eigentlich endliche Automaten, die einerseits syntaktische Strukturen und anderseits die Wortgruppen beschreiben können.

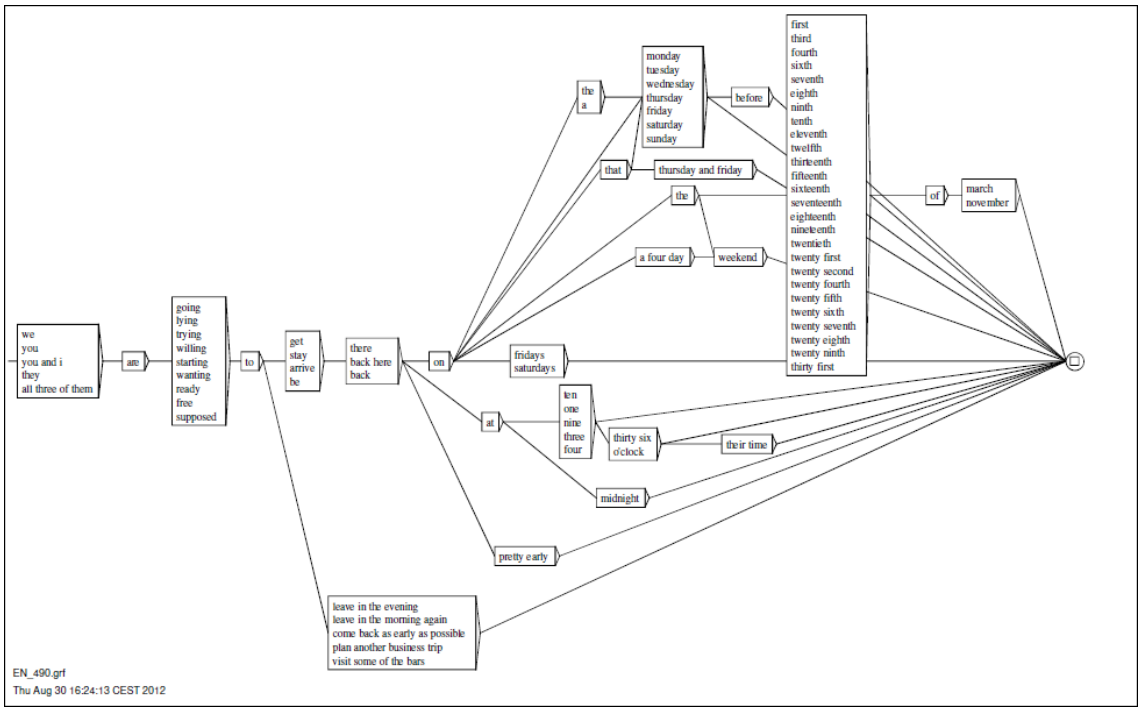


Abbildung 4.17: Lokale Grammatik

Durch die breitgestellte Baumstruktur wird eine flache (oder tiefe) Struktur wie oben konstruiert, die die aufgeräumten Wortgruppen als Knoten enthält. Jeder Pfad der lokalen Grammatik ist ein endlicher Automat und kann einen Satz (oder mehrere Sätze) korrekt produzieren.

we are going to get there on tuesday
we are going to get there on tuesday before

we are trying to arrive there on the tenth of november
we are trying to arrive there on the eleventh
we are ready to arrive back here at nine
we are ready to arrive back here at nine thirty six
you are ready to be back here at four
…

Durch diese produktive Grammatikregel können zahlreiche Sätze automatisch generiert werden, die nicht im originalen Korpus existieren, aber dennoch grammatisch korrekt sind.

Die Eltern- und Tochter-Beziehung in der Baumstruktur kann durch die Subgraphen eines großen Graphen implementiert werden. Dies ermöglicht eine tiefe Analyse komplexer Strukturen.

Eine der Anwendungsmöglichkeiten der lokalen Grammatiken ist, die gegebenen Sätze syntaktisch (und semantisch mit Hilfe von externen Lexika) zu prüfen. Dadurch können beispielsweise die speziellen Informationen gezielt extrahiert werden (siehe Kapitel 5).

4.5 Fazit

Die Performance des in der vorliegenden Arbeit entwickelten EDSI-Ansatzes wird im Vergleich zum klassischen Ansatz evaluiert. Die drei Punkte (siehe Kapitel 3.6), die zunächst auf der theoretischen Ebene geklärt wurden, werden durch die Testergebnisse quantitativ und qualitativ erläutert.

- Die Testergebnisse der quantitativen Evaluation im Englischen und im Deutschen sind sehr überzeugend. Die Werte liegen im Bereich der theoretischen Prognose. Die gravierende Erhöhung der Verarbeitungseffizienz ist im Gegensatz zu den klassischen Ansätzen eine wesentliche Verbesserung.

- Außerdem wurde eine neue Lernstrategie zur Verfeinerung der Fuzzy-Treebanks entwickelt, wodurch das gesamte Ergebnis etwas verbessert werden kann.

- In der vorliegenden Arbeit wurden die kontextfreien Grammatiken automatisch extrahiert. Auf Basis der gewonnenen syntaktischen Informationen erfolgt die Induktion der lokalen Grammatiken mit Hilfe manueller Unterstützung.

Das Hauptziel dieser Arbeit, die Verarbeitungseffizienz zu erhöhen, wurde durch den EDSI-Ansatz erreicht.

5. Anwendung der LG im Information-Retrieval

Wie bereits erwähnt, finden lokale Grammatiken in den linguistischen Untersuchungen, insbesondere im Information-Retrieval, vielseitige Anwendung. Dieses Kapitel dient als zusätzlicher Teil der vorliegenden Arbeit, der eine typische Anwendung der lokalen Grammatiken zur Erkennung der speziellen Informationen vorstellt. Hierbei werden der Arbeitsmechanismus und die Produktivität der lokalen Grammatiken präsentiert.

Das Information-Retrieval[76] befasst sich mit dem Suchen nach benutzerrelevanter Information innerhalb einer Dokumentenkollektion. Mit Hilfe der induzierten lokalen Grammatiken können die für Benutzer interessanten Informationen gezielt extrahiert werden. In diesem Anwendungsbeispiel handelt es sich um die Extraktion der Firmeninformation in der Industriebranche vor allem im Bereich Maschinenbau. Die Anwendung besteht dabei aus drei Teilaufgaben:

- Erkennung der Firmennamen
- Erkennung der Adressen
- Erkennung der Firmenprofile

Die Gewinnung solcher Informationen ist besonders hilfreich bei der Verbesserung der Suchqualität vertikaler Suchmaschinen[77].

Die in dieser Arbeit darzustellende Beispielanwendung der lokalen Grammatiken befasst sich mit einem eingeschränkten Themenbereich. In Anbetracht der Heterogenität der Webseiten ist das Lernen von Regeln sehr schwierig, wenn kein ausreichend annotiertes Korpus verfügbar

[76] Siehe auch Modern Information Retrieval (Baeza-Yates & Ribeiro-Neto 1999).

[77] Vertikale Suchmaschinen unterscheiden sich von den allgemeinen Suchmaschinen in dem Punkt, dass sie sich auf bestimmte Themen oder spezielle Zielgruppen konzentrieren. Damit können die Informationen „tiefer“ verarbeitet werden und qualitativere Suchresultate liefern. Beispielsweise wird ein Prototyp der vertikalen Suchmaschine für die Industriebranchen www.is-ma.com in der vorliegenden Arbeit entwickelt.

ist. Durch die Themeneinschränkung können die relativ homogenen (strukturell und inhaltlich) Webseiten zusammengestellt werden, da die Seiten aus diesem Themenbereich die jeweilige Terminologie oder ähnliche syntaktische Strukturen verwenden. So wird das automatische Lernen von Regeln und speziellen Informationen erleichtert (oder ermöglicht).

Die Grammatikregeln werden zunächst aus den Trainingskorpora durch den EDSI-Ansatz analysiert und manuell erstellt. Aus einem Testkorpus extrahieren diese Regeln die Informationen, die evaluiert werden.

5.1 Erkennung der Firmennamen

Das Hauptaufgabenziel der Extraktion der Firmeninformation ist die Erkennung von Firmennamen, die eigentlich zu einer Unterkategorie der Eigennamenerkennung (engl. *Named Entity Recognition*) gehört. Unter diese Unterkategorie fallen nur die Firmen, die kommerziell betrieben werden und eigene Produkte oder Dienstleistungen in der Industriebranche anbieten.

5.1.1 Interner und externer Kontext

Da die Identifikation eines Eigennamens nicht über die grammatikalische Analyse des Textes erfolgt, müssen die linken und rechten Kontexte der Eigennamen untersucht werden, sodass Eigennamen im Text automatisch erkannt werden können.

Wie bereits in Kapitel 2.2.3.1 beschrieben, berücksichtigt ein Erkennungssystem der Eigennamen nicht nur den internen sondern auch den externen Kontext. Der interne Kontext versucht die möglichen Eigennamen zu lernen, die dann durch den externen Kontext weiter identifiziert werden. So kann eine hohe Erkennungsrate erreicht werden. Die in der Untersuchung genutzten lokalen Grammatiken der Firmennamen basieren überwiegend auf dem internen Kontext.

Interner Kontext

Firmennamen haben im Allgemeinen ihre eigene Struktur, die zu Grammatikregeln abgeleitet werden können.

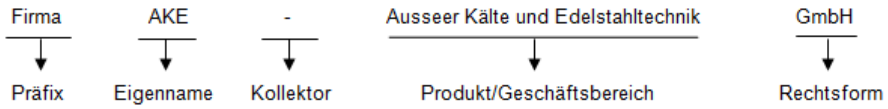


Abbildung 5.1: Struktur des Firmennamens

Das obige Beispiel stellt eine typische Struktur eines Firmennamens dar. Der Name fängt mit dem Wort „*Firma*“ an. Danach steht „*AKE*“ als die Abkürzung oder der Eigenname. Dieser Teil besteht meistens aus den großgeschrieben Buchstaben, die manchmal durch „ . “, „ - “ und „UND“[78] modifiziert werden. Der nächste Teil „*Ausseer Kälte und Edelstahltechnik“* bezeichnet die Produkte/Dienstleistung oder die Geschäftsbereiche. Weil die Webseiten hauptsächlich aus dem Maschinenbau stammen, enthält dieser Teil oft Schlüsselwörter wie „*Anlagenbau*“ oder „*Maschinenbau*“ etc. Anschließend wird die standardisierte Rechtsform „*GmbH*“ der Firma angegeben. In einigen Fällen werden aus verschiedenen Gründen auch englische Rechtsformen gefunden.

Für jeden Knoten in der Grammatik werden mit Hilfe des EDSI-Ansatzes verschiedene Konstituenten (Phasen und reguläre Ausdrücke) extrahiert und/oder induziert, die funktional gruppiert werden.

Knoten	Anzahl der Varianten	Beispiele
Präfix	6	Firma, Fa.
Eigenname	7308	<MAJ>
Inhaber	60	<PRE>
Produkt/Geschäftsbereich	1289 mit Subgraphen	
Rechtsform	360	GmbH, oHG
Kollektor	8	&, <E>

Tabelle 5.1: Variantenzahl des Firmennamens

Jede in der Tabelle angegebene Anzahl bezeichnet die direkten Pfade ihres Knotens, die durch rekursive Subgraphen rasant vermehrt werden können. (siehe detailierte Graphen in Anhang 7.3.3.1)

[78] UND ist eine in dieser Arbeit definierte Subklasse, die verschiedene UND-Relationen in den Firmennamen formuliert (z.B. &, +, und etc.).

Externer Kontext

Bei der Eigennamenerkennung spielt auch der externe Kontext eine wichtige Rolle. Die Firmennamen auf den Webseiten kommen oft mit anderen Informationen (z.B. Impressum oder Kontakt) zusammen vor, die durch externe Kontexte entfernt werden müssen. Dabei werden die negativen Wörter, die die anderen Informationen von den Firmennamen trennen können, aufgelistet (siehe auch Anhang 7.4.1):

der die das des
ihr ihre
home email
kontakt impressum profil

Da solche Wörter häufig vor einem Firmennamen auftreten, ist die Wortsequenz aller nachstehenden Wörter ein möglicher Firmenname. Um die Präzision der Namenserkennung weiter zu erhöhen, soll diese Liste entsprechend erweitert werden.

Ein anderer wichtiger Kontext ist der Domainname der Firmen, der oft teilweise (oder sogar komplett) den Firmennamen enthält.

abend-maschinenbau.de
Abend Maschinenbau e.K.

Wie im obigen Beispiel gezeigt wurde, sind die Bestandteile „*abend*“ und „*maschinenbau*“ sehr hilfreich dabei, den Firmennamen zu identifizieren.

Da es in diesem Test um die Präsentation der lokalen Grammatiken geht, die hauptsächlich auf dem internen Kontext basieren, soll auf den externen Kontext nicht näher eingegangen werden.

5.1.2 Grammatiken der Firmennamen

Durch die Kombination der Knoten entstehen unterschiedliche Grammatikregeln zur Erkennung der Firmennamen. Die Regeln werden im Folgenden als kontextfreie Grammatiken dargestellt.

Firmenname → (<E>|PF) EN c PG c (<E>|IH) c RF
Firmenname → PG c (<E>|RF) EN c (<E>|IH) c RF
Firmenname → (<E>|RF) EN c RF
Firmenname → PG c (<E>|IH) c RF
Firmenname → PF EN
Firmenname → PF c PG
Firmenname → <PRE> (<E>|<PRE>) c RF
...
PF → Firma, Fa., Ingenieurbüro, Ing. Büro etc.
EN → <MAJ>, <MAJ> <PRE> etc.
PG → <merkmale>, <merkmale> <PRE> etc.
IH → Gebr. <PRE>, Inh. von <PRE> etc.
RF → GmbH, mbH, Ltd, oHG etc.
c → <E>, -, in, im, für etc.
<merkmale> → Anlagen, Apparatebau, Automation, CNC, CAD etc.[79]

[79] Beschreibung der Elemente:

PF: Präfix

EN: Eigennamen

PG: Produkt, Dienstleistung oder Geschäftsbereich

IH: Inhaber

RF: Rechtsform

c: Kollektor zwischen den Knoten

<E>: Leerzeichen oder leerer Wert

<PRE>: ein Wort mit einem großgeschriebenen Buchstaben am Anfang

<MAJ>: ein großgeschriebenes Wort

<merkmale>: ein Subgraph mit allen Schlüsselwörtern vom Knoten PG

In den Regeln spielt der Knoten RF (Rechtsform) eine sehr wichtige Rolle, weil er fast der Bestandteil jeder „Firmenname“-Regel ist. Die Grammatiken der Firmennamen werden in der vorliegenden Arbeit mit dem Unitex-Tool erstellt (siehe detailierte Präsentation in Form der lokalen Grammatiken in Anhang 7.3.3.1).

Diese Grammatik kann zahlreiche Firmennamen erkennen. Neben den richtigen Namen werden auch manche Namen falsch erkannt. Allerdings kommt eine komplette falsche Erkennung relativ selten vor. Dahingegen werden die meisten falschen Resultate entweder zu lang oder zu kurz identifiziert. Im Falle einer übergroßen Identifizierung können die überflüssigen Wörter mit Hilfe eines externen Kontexts entfernt werden.

5.2 Erkennung der Adressen

Die Erkennung der Adressen konzentriert sich auf die Straßennamen, weil die PLZ und der Ort relativ einfach mit einer Liste (engl. *Gazetteer*) zu identifizieren sind. Genau wie die Firmennamen haben auch Straßennamen ihre eigene Struktur.

5.2.1 Interner und externer Kontext

Die Vorgehensweise zur Erkennung der Straßennamen ist der Erkennung der Firmennamen recht ähnlich. Die lokale Grammatik wird durch die Analyse des internen Kontexts erstellt. Der externe Kontext dient dazu, die Erkennungsergebnisse zu kompensieren.

Interner Kontext

Normalerweise enthält eine Adresse zwei Charakteristika: eine Bezeichnung der Straßenart und eine dazugehörige Hausnummer.

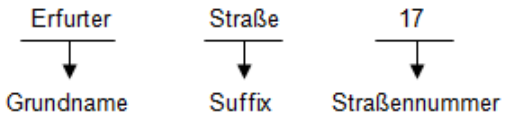


Abbildung 5.2: Struktur des Straßennamens

Ein üblicher Straßenname besteht aus einem Grundnamen, einem Suffix und einer Hausnummer. Der Grundname ist meistens ein Nomen oder eine Phrase, die durch mehrere Nomina, ggf. auch Sonderzeichen wie „ . “ oder „ - “ kombiniert wird. Das Suffix ist in den meisten Fällen das Wort „*Straße*“ mit aufzählbaren Variationsmöglichkeiten, wie *Str. Strasse, Str* etc. Oft wird das Suffix mit dem Grundnamen zusammengeschrieben. Des Weiteren kommen auch oft Wörter wie „*Allee, Bach, Brücke, Weg* etc.“ als Suffix vor. Die Straßennummer ist üblicherweise eine natürliche Nummer, die kürzer als 4-stellig ist.

Eine andere übliche Form der Straßennamen hat statt des Suffixes nach dem Grundnamen eine Präposition vor dem Grundnamen:

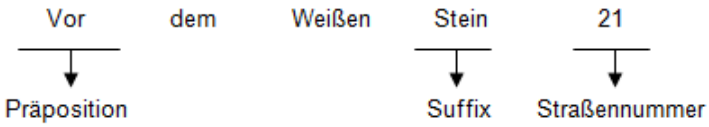


Abbildung 5.3: Andere Struktur des Straßennamens

In diesem Fall ist das Suffix meistens anders als „*Straße*“, bspw. „*Berg, Dorf, Ecke, Feld, Gebiet, Platz* etc.“

Da die meisten Straßennamen ähnliche Strukturen besitzen[80], können sie mit aufzählbaren endlichen Automaten beschrieben werden. Wie bei der Erkennung der Firmennamen wird jeder Straßenname als ein Pfad betrachtet. Durch die Kombination zwischen den Knoten können die möglichen Pfade abgedeckt werden.

Knoten	Anzahl der Varianten	Beispiele
Grundname	11	<PRE>, <PRE>-<PRE>
Suffix	22	Allee, Bach, Str etc.
Straßennummer	100	<nummer>, <nummer>a
Präposition	7	An, Am, Vor etc.

Tabelle 5.2: Variantenanzahl des Straßennamens

[80] Die Straßennamen, die keine Standart-Struktur besitzen (z.B. „in der Wann“), müssen als Sonderfälle betrachtet und analysiert werden.

Die Einträge in jedem Knoten werden durch reguläre Ausdrücke beschrieben, die wesentlich mehr Pfade als die angegebene Variantenanzahl durchsuchen können. (siehe Graphen der lokalen Grammatik in Anhang 7.3.3.2)

Externer Kontext

Die Straßennamen auf den Webseiten treten normalerweise mit den Kontakt- oder Impressumsinformationen auf. Es ist sehr schwierig, die Straßennamen allein durch den internen Kontext von den anderen Informationen zu trennen. Daher muss der externe Kontext zur Erkennung der Straßennamen beitragen.

Auf der linken Seite von den Straßennamen stehen oft die Firmennamen. Dahingegen folgt die PLZ generell hinter den Straßennamen. Um die überflüssigen Teile aus den gelernten Namen zu entfernen, wird eine Menge von Wörtern als Indikator gesammelt (siehe Anhang 7.4.2):

- Links von den Straßennamen:

  GmbH, AG, KG
  Büro, Fabrik
  Kontakt, Impressum, Home, Profile, Profil
  Technik, Konstruktion, Bau, Betrieb, Plattform, Maschine, Maschinen

- Rechts von den Straßennamen:

  Nummer mit mindestens vier Ziffern
  D, A, CH

Wie bei der Erkennung der Firmennamen zeigt der Indikator, welche Teile aus den gewonnenen Resultaten abgetrennt werden sollen, damit die richtigen Straßennamen extrahiert werden.

5.2.2 Grammatiken der Straßennamen

In dieser Untersuchung werden die Grammatikregeln nach den Strukturen erstellt, die die meisten Straßennamen erkennen können. Die Regeln werden aus einem manuell erstellten Korpus, das 1076 Straßennamen umfasst, induziert.

Straßennamen → GN SF SN
Straßennamen → GNSF SN
Straßennamen → PP AT GN SN
GN → <PRE> c (<E>|<PRE>)
GN → <PRE> c <PRE> c (<E>|<PRE>)
SF → Allee, Bach, Berg, Brücke, Ecke, Feld, Str. etc.
SN → <nummer>
SN → <nummer> <hnsuffix>
SN → <nummer> (&|,|+|-|/) <nummer>
SN → <nummer> Haus <hnsuffix> <nummer> (<E>|.0) G
<nummer> → eine Nummer 1 bis 999
<hnsuffix> → ein englischer Buchstabe a bis z
c → <E>, ., - etc. [81].

Weil die vorliegenden Grammatikregeln nur aus dem internen Kontext extrahiert werden, entscheidet der interne Kontext (insbesondere der Knoten: Straßennummer) über die Qualität

[81] Beschreibung der Regelelemente:

GN: Grundname;

SF: Suffix;

HN: Hausnummer;

GNSF: eine Zusammenfügung vom Grundnamen und Suffix

PP: Präposition

c: Kollektor zwischen den Knoten

<nummer>: eine natürliche Nummer

<hnsuffix>: ein Suffix von der Hausnummer

<PRE>: ein Wort mit einem großgeschriebenen Buchstaben am Anfang

<E>: Leerzeichen oder leerer Wert

der Namenserkennung. Die graphisch erstellte lokale Grammatik zur Erkennung der Straßennamen, die durch das Unitex-Tool implementiert wird, ist in Anhang 7.3.3.2 zu finden.

5.3 Erkennung der Firmenprofile

Die in einer Webseite aufgeteilten Informationen sind für Benutzer generell unterschiedlich relevant. In diesem Sinne kommen den Informationen entsprechend unterschiedliche Gewichte zu. Beispielsweise sind die Kontaktinformationen viel wichtiger und informativer als die Begrüßungsfloskeln. Daher ist es von großer Bedeutung, solche Informationen aus dem gesamten Text aufzufinden und so die Suchqualität zu erhöhen.

Außer den Kontaktinformationen sind die Firmenprofile (z.B. angebotene Produkte oder Geschäftsbereiche etc.) oft auch für die Websuchenden hochinteressant. Beispielsweise bringt die Beschreibung der Firmenprofile die Möglichkeit, eine Firma nach ihren Produkten, ihrem Geschäftsbereich, ihrer Größe, dem Standort etc. zu klassifizieren. Diese Maßnahme kann einerseits eine Verbesserung der Suchqualität versprechen und anderseits dem Benutzer mehr Suchmöglichkeiten bieten.

Die automatische Extraktion der Firmenprofile wird jedoch bis heute wenig erforscht, weil die Ausdrücke der Firmenprofile im Gegensatz zu Eigennamen ziemlich heterogen und komplex sind. Aufgrund dieser Heterogenität und Komplexität können die Firmenprofile sehr unterschiedlich ausdrückt werden. Entsprechend entstehen aus diesen Ausdrücken wesentlich mehr Strukturen. Das heißt, es ist unerlässlich, zahlreiche Grammatikregeln (wieder eine aufwendige manuelle Arbeit) zu erstellen, um alle (oder die meisten) möglichen Strukturen beschreiben zu können.

Um dieses Problem zu verringern, werden die in dieser Untersuchung verwendeten Korpora auf bestimmte Themen (Maschinenbau und ähnliche Branchen) beschränkt. In diesem Test wird versucht, die Beschreibungen der Produkte oder die Geschäftsbereiche mit Hilfe von lokalen Grammatiken gezielt zu extrahieren.

5.3.1 Klassifikation der Firmenprofile

Die Firmenprofile auf den Websites sind oft mit Hilfe gängiger Phrasen formuliert. Dies erlaubt eine automatische Extraktion durch die lokalen Grammatiken, weil die Ausdrücke ähnliche Strukturen enthalten. Im Vergleich zur Namenserkennung sind die Strukturen jedoch wesentlich komplexer. Die folgenden Sätze aus dem Trainingskorpus stellen einerseits die Homogenität der äquivalenten Ausdrücke und andererseits die vielseitigen Formulierungsmöglichkeiten dar.

157 *Auf modernsten Fertigungsmaschinen produzieren wir Schrittgetriebe in höchster Präzision.*

173 *Unsere Unternehmensgruppe ist in den Bereichen Maschinenbau, Automobilindustrie und Dienstleistung tätig.*

176 *Die HSM Technology GmbH ist seit Jahren sehr erfolgreich im Spezial- Maschinenbau tätig.*

Der Satz 157 enthält eine einfache Struktur „*produzieren*“ (Kategorie: Produzieren), die oft die Produkte dieser Firma präsentiert. Die Sätze 173 und 176 besitzen dieselbe Struktur „*ist ... tätig*“ (Kategorie: Tätig). Die in diesem Kontext vorkommenden Wörter „*in den Bereichen*“ und „*erfolgreich im*“ (Kategorie: Bereich) sind zwei typische Indikatoren für die Beschreibung des Geschäftsbereiches. Daher kann eine relativ komplexe Struktur durch die Kombination der Kategorien Tätig und Bereich beschrieben werden.

Der Satz 157 hat keine gültigen gemeinsamen Strukturen[82] mit den Sätzen 173 und 176. Dies stellt die Schwierigkeit dieser Untersuchung dar, dass zahlreiche Ausdrücke (auch Strukturen) zur Formulierung der Firmenprofile gesammelt werden müssen, um eine möglichst allgemeine Grammatik zu induzieren. Des Weiteren sind die Ausdrücke der Firmenprofile oft lange und komplexe Sätze. Durch die Abweichung der Wortstellung wird das Heterogenitätsproblem deutlich schärfer als die Namenserkennung.

[82] Aus einer gültigen gemeinsamen Struktur in der vorliegenden Untersuchung kann eine lokale Grammatik induziert werden, die wiederum verwendet werden kann, um die gewünschten Informationen zu extrahieren. Ansonsten ist eine gemeinsame Struktur nicht gültig.

Um die lokalen Grammatiken leichter zu induzieren, werden die Ausdrücke der Firmenprofile nach ihren Strukturen klassifiziert.

Einfache Strukturen sind elementare Strukturen und entsprechen meistens den Phasen, die normalerweise als Teilstrukturen betrachtet werden sollen.

- Kategorie **Produzieren** enthält die Verben, die stark mit „Produkten" verbunden sind. Z.B. *bauen, liefern, fertigen, erstellen, produzieren* etc.
- Kategorie **Erfahrung** enthält die Strukturen, die funktional ähnlich wie die Strukturen von „Erfahrung" sind. Z.B. *Lösung, Know-how* etc.
- Kategorie **Bereich** enthält die Strukturen, die die Geschäftsbereiche beschreiben können. Z.B. *im Bereich von, im Umfeld der* etc.
- Kategorie **Herstellung** enthält die Nominalphrasen von „Herstellung". Z.B. *Herstellung von, Fertigung zur Herstellung* etc.

Komplexe Strukturen bestehen aus mehreren einfachen Strukturen und sind meistens in der Lage, einen komplexen Satz zu beschreiben.

- Kategorie **Herstellen** besteht aus den Strukturen, die durch die Kombination von Verben wie „herstellen" und anderen Kategorien beschrieben werden können. Z.B.

  Wir stellen *Spezialmaschinen für Groß- und Industriebäckereien* her.

- Kategorie **Partner** enthält die Strukturen, die aus den Indikatoren wie „Partner" und anderen Kategorien bestehen können. Z.B.

  Weltweit sind wir Ihr Technologiepartner *für Dispergieren, Mischen, Trennen, Schleifen und Prüfen.*

- Kategorie **Kompetenz** enthält die Strukturen, die durch „Kompetenz" und ihre Synonyme repräsentiert werden kann. Z.B.

  Die Jan Korte GmbH ist ein mittelständisches Unternehmen mit Kernkompetenzen *im Tief-, Straßen-, Rohrleitungs- und Kanalbau.*

- Kategorie **Programm** umfasst die Strukturen, die das Produktprogramm beschreiben können. Z.B.

  PEDAX bietet ein Komplettprogramm *an Maschinen und Anlagen zur Bearbeitung von Betonstahl.*

- Kategorie **Abrunden** hat die Strukturen, die das Keywort „abrunden“ und seine Varianten enthalten. Z.B.

  Maschinen für Medizintechnik, Umwelttechnik und Sicherheitstechnik runden *unsere Produktpalette* ab.

- Kategorie **Spezialisieren** enthält die Strukturen, die die meisten Varianten von „spezialisieren“ abdecken können. Z.B.

  Wir sind spezialisiert auf *die Herstellung und Bearbeitung von Präzisions-Serienteilen für den Anlagen-, Maschinen- und Fahrzeugbau.*

Die obigen Kategorien stammen aus 608 Sätzen im Training-Korpus. Für jede Kategorie werden jeweils die entsprechenden Grammatikregeln getrennt entwickelt. Da die Regeln mit Hilfe des statistischen Ansatzes EDSI induziert werden, können sie nur die Häufigen Strukturen identifizieren. Die lokalen Grammatiken seltener Ausdrücke (Häufigkeit kleiner als 3) werden in der Untersuchung nicht berücksichtigt.

5.3.2 Grammatiken der Firmenprofile

Die lokalen Grammatiken jeder Kategorie werden als Subgraphen betrachtet, die anschließend einen gesamten Graphen bilden. Aufgrund der hohen Komplexität der Profilerkennung werden deutlich mehr Grammatikregeln erstellt, sodass ein möglichst hoher Recall erreicht wird. Die Regeln werden im Folgenden zunächst als kontextfreie Grammatiken präsentiert:

Firmenprofile → *<Herstellen>*
Firmenprofile → *<Partner>*
Firmenprofile → *<Kompetenz>*
Firmenprofile → *<Programm>*

Firmenprofile → <Abrunden>
Firmenprofile → <Spezialisieren>
Firmenprofile → <Produzieren>
Firmenprofile → <Erfahrung>
Firmenprofile → <Bereich>
<Herstellen> → bieten, bietet <E>, <SQM> Ihnen, Kunden
<Herstellen> → bieten, bietet <SQM> <Angebot> <SuffixH>
<Angebot> → Lösungen, Angebot, Programm, Portfolio, Palette etc.
<SuffixH> → an, für, von, <MOT><<er$>>, <Bereich>, etc.
< Bereich> → im Bereich, Umfeld <SuffixB>
< Bereich > → in, aus den, allen, <SuffixB>, etc.
< Bereich > → auf dem Gebiet, Sektor <SuffixB>
< SuffixB > → auf, in, im, von, für, zur, zum, <MOT><<er$>>, etc.[83]

Aufgrund der Vielzahl von Grammatiken können sie hier nur teilweise dargestellt werden. Die lokalen Grammatiken werden in der vorliegenden Untersuchung mit Unitex implementiert. Die genaue graphische Darstellung der Grammatikregeln ist im Anhang 7.3.3.3 zu finden.

Zur Erkennung der Firmenprofile werden zwei lokale Grammatiken entwickelt, die eine Kaskade-Verarbeitung[84] erlauben. In Bezug auf die im letzten Kapitel beschriebenen Kategorien unterscheiden sich die Grammatiken im Punkt der Komplexität. Die komplexen Grammatikregeln sind in der Lage, komplexe Strukturen durch mehrere Indikatoren zu identifizieren. Dahingegen dienen die einfachen Regeln dazu, einfache Strukturen oder kurze Teile der komplexen Strukturen zu erkennen. Dies führt ggf. zu einer übermäßigen Extraktion. Da die erstellten Grammatikregeln unterschiedliche Genauigkeiten zur Profilerkennung bieten, wer-

[83] Beschreibung der Knoten:

Das Komma trennt die äquivalenten Konstituenten;

< > markiert die Subgraphen;

<E> ist ein leerer Wert

[84] Siehe auch Kapitel 2.2.3.3

den sie entsprechend unterschiedlich gewichtet. Bei der Extraktion wird zunächst die komplexe lokale Grammatik eingesetzt. Falls keine Ausdrücke gefunden werden können, versucht die einfache Grammatik die Texte zu identifizieren. Dadurch wird gewährleistet, zwei extreme Fälle (übermäßige und mangelhafte Extraktion) zu vermeiden.

5.4 Evaluation

Für diesen Test werden insgesamt fünf Korpora bereitgestellt, die aus den Firmenseiten in der Branche Maschinenbau und anderen relevanten Bereichen stammen.

- MaInfo_name: 1076 Firmennamen;
- MaInfo_address: 1076 Straßennamen;
- MaInfo_context: Plaintexte aus 718 Firmensites;
- MaInfo_profile: 608 Ausdrücke der Firmenprofile;
- MaInfo_evaluation: Plaintexte aus 320 Firmensites.

Für die Firmen- und Straßennamen werden zwei Korpora, die jeweils 1076 Einträge umfassen, manuell erstellt. Diese Korpora enthalten keinen externen Kontext und dienen hauptsächlich dazu, die Grammatikregeln zu induzieren. Um den externen Kontext nicht außer Acht zu lassen, wird noch ein Trainingskorpus automatisch vorbereitet, das eine Sammlung der Webseiten von 718 Firmensites ist. Die Webseiten enthalten keine HTML-Tags und werden durch Unitex in Sätze zerlegt. Für die Firmenprofile wird ein Korpus mit 608 Ausdrücken manuell aufgebaut.

Des Weiteren steht ein Evaluationskorpus zur Verfügung, das die Plaintexte von 320 Firmensites aus dem Internet beinhaltet. Aus diesem Korpus werden die gewünschten Informationen manuell extrahiert und mit den automatisch gefundenen Daten verglichen. Das Evaluationskorpus enthält keine Firmen der vier Trainingskorpora. Damit wird die Störung der Trainingsdaten ausgeschlossen.

Aus den drei Trainingskorpora (MaInfo_name, MaInfo_address und MaInfo_profile) werden die lokalen Grammatiken mit Hilfe des EDSI-Ansatzes induziert, die im Unitex-Tool[85] graphisch erstellt und für die weitere Verarbeitung vorbereitet werden. Die externen Kontexte von Firmennamen und Straßennamen werden mit Perl-Skripten implementiert.

Zur Evaluation werden die erstellten lokalen Grammatiken auf dem Evaluationskorpus durchgeführt. Für die erzielten Resultate werden die drei üblichen Messungsmaße Präzision, Recall und F-Maß (siehe Kapitel 4.1.1) berechnet.

5.4.1 Evaluation der Eigennamen

Zur Erkennung der Eigennamen können regelbasierte Systeme ziemlich erfolgreich eingesetzt werden[86]. In der vorliegenden Untersuchung wird für die Firmen- und Straßennamen jeweils eine lokale Grammatik entwickelt, die sowohl den internen als auch den externen Kontext berücksichtigt.

Vor der Evaluation werden jedoch die Firmen- und Straßennamen aus dem Evaluationskorpus manuell gesammelt, die dann mit den automatisch extrahierten Daten verglichen werden.

Datentyp	Gesamte Einträge	Gelernte Einträge	Richtige Einträge	Recall (%)	Präzision (%)	F1 (%)
Firmennamen	320	316	264	82,5	83,5	83,0
Straßennamen	320	289	219	68,4	77,1	72,5

Tabelle 5.3: Evaluation der Eigennamenerkennung

Es ist zu beachten, dass die Schreibweise eines Firmennamens ggf. mehrere Varianten hat. Z.B.

{ Brinck & Co. GmbH
  Brinck und Co. GmbH

[85] Siehe Kapitel 2.3.4.2. Das Userhandbuch ist auf der Startseite vom Unitex frei verfügbar. Das Tool kann entweder über eine graphische Oberfläche oder per Kommandozeile durchgeführt werden. In der vorliegenden Arbeit werden die Befehle durch Perl-Skripte aufgerufen und durchgeführt.

[86] Siehe den SPARSER-Ansatz in Kapitel 2.2.3.1

{ BUT Blech und Tortechnik GmbH
  Firma BUT

Wie die Beispiele zeigen, werden alle Namen als korrekte Namen betrachtet. Aus den automatisch extrahierten Namen wird der häufigste Name als der beste Kandidat ausgewählt. Die anderen Namen werden aussortiert und können nicht weiter berücksichtigt werden. Falls der ausgewählte Name identisch mit einer der manuell gesammelten Namen ist, wird diese Erkennung als erfolgreich evaluiert. Diese Vorgehensweise gilt auch für die Erkennung der Straßennamen.

Da das Element „Rechtsform“ in der Grammatik eine zentrale Rolle spielt, ist die Erkennung davon abhängig. D.h. ein Firmenname ohne Rechtsform kann oft mit dieser Grammatik nicht erkannt werden. Ebenso wird die Straßennummer schwer zur Erkennung der Straßennamen gewichtet. Dies ist auch der Grund für die meisten falsch erkannten Straßennamen.

5.4.2 Evaluation der Firmenprofile

Da die automatische Extraktion der Firmenprofile bis heute noch nicht viel erforscht wurde und wird, für praktische Anwendungen aber hochinteressant sein kann, wird in diesem Test versucht, die Firmenprofile über Produkte oder Geschäftsbereiche mit den lokalen Grammatiken zu erkennen.

Datentyp	Gelernte Einträge	Richtige Einträge	Falsche Einträge	Recall (%)	Präzision (%)
Firmenprofile	592	484	108	-	81,8

Tabelle 5.4: Evaluation der Profilerkennung

Bei der Evaluation wird ein erkannter Ausdruck als korrekt bewertet, wenn er konkrete Produkte/Dienstleistungen oder Geschäftsbereiche angibt. Dabei treten einige Probleme auf:

- Eine Floskel enthält nur eine abstrakte Angabe, die nicht von der Extraktion erwartet wird.

  Nutzen Sie unsere Leistungsbereitschaft, sie bietet Ihnen den Anschluß an die Zukunft!

- Die Sätze werden manchmal vom Unitex falsch segmentiert, weil die Texte der Webseiten unstrukturiert sind. Z.B. besteht ein segmentierter Satz aus mehreren Sätzen. Da dieses Problem nicht von der Erkennung verursacht wird, wird der erkannte Ausdruck in diesem Fall nicht als fasch betrachtet.

- Andere Informationen verwenden ähnliche Strukturen wie die Firmenprofile, insbesondere die Seiten mit Stellenangeboten:

  Mit einer Ausbildungsquote von 10 Prozent stellen wir auch hinsichtich der Personalpolitik die Weichen für die Zukunft.

In Anbetracht der Tatsache, dass die entwickelten Grammatikregeln nicht alle Ausdrücke der Firmenprofile abdecken können, wird der Recall nicht berechnet. Da die meisten Firmen jedoch 1 bis 3 solcher Ausdrücke verwenden, ist dieser Wert abschätzbar. Im Folgenden werden die ersten sechs extrahierten Firmenprofile als Beispiele dargestellt:

- Das Leistungsspektrum umfasst die Berechnung, Auslegung und Konstruktion, sowie die 3D-Modellierung und Zeichnungserstellung mit Hilfe moderner CAD - Werkzeuge.

- 1999 von Herrn Dipl.-Ing. (FH) René Kiewitt in Berlin gegründet, beschäftigt sich das Unternehmen mit Projektaufgaben im allgemeinen Maschinen- und Anlagenbau.

- Vom individuellen Einzelstück bis zu Serienprodukten bieten wir unseren Kunden ein breites Spektrum an maßgeschneiderten und innovativen Lösungen.

- Ihr Ansprechpartner für industrielle Kühllösungen

- In mehr als drei Jahrzehnten haben wir uns durch konsequentes Innovationsmanagement, Ideen, Fleiß, Können und Flexibilität zu einem der führenden Spezialisten für die Konstruktion und Herstellung individueller, kundenspezifischer Systemlösungen im Bereich von Sondermaschinen und Automatisierungseinrichtungen entwickelt.

- Als erfahrener Partner bieten wir unseren Kunden aus der Automobil-, und Automobilzulieferindustrie, in der Werkzeugmaschinenautomation und im Bereich der Intralogistik intelligente Systemlösungen und setzen damit neue Maßstäbe für Wirtschaftlichkeit, Flexibilität und Qualität.

Die Extraktion erfolgt durch die lokalen Grammatiken und die Einschränkung der Themen, aber ohne die Unterstützung des externen Kontextes. Die extrahierten Firmenprofile bringen wiederum die Möglichkeit, dass die genauen Produkte oder Geschäftsbereiche aufgefunden und klassifiziert werden können.

5.5 Fazit

In dieser Untersuchung wurde zur Darstellung der Arbeitsweise und Beschreibungsfähigkeit dieser Grammatik-Theorie eine Beispielanwendung der lokalen Grammatiken realisiert. Die Entwicklung der Grammatikregeln wird mit Hilfe des EDSI-Ansatzes erleichtert.

Wie die Testergebnisse aufzeigen, sind lokale Grammatiken generell geeignet, spezielle Information zu erkennen und zu extrahieren. Durch eine Erweiterung der Grammatikregeln und der Indikatoren des externen Kontexts besteht die Möglichkeit, die Resultate zu verbessern.

Allerdings ist die manuelle Erstellung der lokalen Grammatiken trotz der Semi-Automatisierung der Grammatikinduktion sehr aufwendig. Diese Handarbeit besteht hauptsächlich aus zwei Gründen: Einerseits müssen die passenden Strukturen für die gewünschten Informationen ausgewählt werden, anderseits gehören die automatisch gruppierten Konstituenten nicht immer zur einer funktionalen Klasse. Ein anderer Nachteil ist, dass die für die bestimmten Themen erstellten lokalen Grammatiken nicht auf andere Themen zu übertragen sind.

6. Zusammenfassung und Ausblick

Die vorliegende Arbeit präsentiert den effizienten Ansatz EDSI zur automatischen Extraktion der syntaktischen Informationen, aus denen unterschiedliche Grammatikregeln automatisch oder semi-automatisch induziert werden können.

Automatische Induktion der Grammatiken ist ein etabliertes und fachübergreifendes Forschungsgebiet, dessen theoretische Hintergründe sich auf maschinelles Lernen (siehe Kapitel 2.1) und die Grammatik-Entwicklung (siehe Kapitel 2.2 und Kapitel 2.3) beziehen.

Aus technischer Sicht wurde der grundlegende Mechanismus des maschinellen Lernens hinsichtlich der linguistischen Anwendung beschrieben. Schrittweise wurden drei relevante Faktoren wie die Modellierung (entspricht distributionalen Informationen in Linguistik), die Wissensrepräsentation (entspricht Grammatikregeln in Linguistik) und die herkömmlichen Lernstrategien skizziert. Dies ist die technische Basis der vorliegenden Arbeit und erlaubt die Grammatik-Induktion durch die Unterstützung der Computertechnik zu automatisieren.

Auf der methodischen Ebene wurden die drei Faktoren des maschinellen Lernens im Bereich Grammatik-Induktion instanziiert. Zur Modellierung der Grammatiken wurden verschiedene Lernmodelle entwickelt. Nach den integrierten Strategien können die verschiedenen Ansätze zur automatischen Grammatik-Induktion generell in zwei Klassen aufgeteilt werden, die als regelbasierte oder lernbasierte Systeme bekannt sind. Die regelbasierten Systeme können vergleichsweise bessere Resultate liefern und sind mit der Anwendung lokaler Grammatiken stark verbunden. Dagegen erleichtern die lernbasierten Systeme die Arbeit zur Erstellung und zur Wartung der Grammatiken. Dabei wurden ein paar statistische und lernbasierte Ansätze präsentiert, die von der vorliegenden Arbeit referenziert werden. Die meisten lernbasierten Ansätze sind in der Lage, die Grammatiken (z.B. CFG) aus eingegebenen Korpora automatisch zu extrahieren.

Neben CFG können unterschiedliche Grammatiken induziert werden. Die extrahierten Grammatiken können in verschiedenen Formen präsentiert werden. Des Weiteren agierten lokale Grammatiken (siehe Kapitel 2.3) als der letzte Schwerpunkt der theoretischen Hinter-

gründe, um die induzierten Grammatiken darzustellen. Außer der formalen Beschreibung der lokalen Grammatiken wurde auch ein nützliches Tool Unitex, das Bestandteil dieser Grammatik-Entwicklung ist, grob vorgestellt.

Im Bereich der automatischen Grammatik-Induktion wird das Effizienzproblem bis heute wenig berücksichtigt, obwohl die klassischen Ansätze schon längst aufzeigen, dass sie eine ziemlich lange Laufzeit brauchen. Bei zunehmender Datenmenge stoßen die Ansätze schnell an ihre Grenzen. Der in dieser Arbeit entwickelte Ansatz EDSI ist ein statistischer und lernbasierter Ansatz, der syntaktische Informationen aus unstrukturierten Korpora automatisch extrahieren kann. EDSI ist ein neuer Ansatz, der sich auf die Erhöhung der Effizienz konzentriert und eine Lösung des Effizienz-Problems bietet.

Der EDSI-Ansatz unterscheidet sich von den herkömmlichen Ansätzen in dem Punkt, dass er das Induktionsproblem der Grammatikregeln aus einem neuen Blickwinkel betrachtet. Eine gängige Strategie der Auffindung der syntaktischen Strukturen beruht auf dem Vergleich zwischen unterschiedlichen Sätzen. Mit Hilfe des Levenshtein-Distanz-Algorithmus‘ kann auch die längste Struktur extrahiert werden (siehe Kapitel 2.2.4.1 und Kapitel 2.2.4.2). Dagegen indexiert EDSI die Sätze vor der Extraktion (siehe Kapitel 3.2.2.). Diese Vorverarbeitung, die die anderen Ansätze nicht haben, verursacht zwar einen zusätzlichen geringen Zeitaufwand, beschleunigt aber die nachkommende Extraktion. Hierdurch werden die gemeinsamen Strukturen ohne jeglichen Vergleich bereits gefunden und in dem Index gespeichert (siehe Kapitel 3.3.1.1). Bei der Extraktion werden alle möglichen Strukturen durch eine speziell entwickelte Methode (die Wortpositionen in Sätzen) gewonnen (siehe Kapitel 3.3.1.2). D.h. die meisten überflüssigen Sätze und Wörter können in der Untersuchung von Anfang an ausgeschlossen werden. Durch die Reduzierung der tatsächlich verarbeiteten Gegenstände wird der Effizienzwert entsprechend erhöht. Weil die Extraktion der syntaktischen Informationen beim EDSI als ein Suchproblem betrachtet wird, benötigt dieser Ansatz eine deutlich kürzere Verarbeitungszeit als die klassischen Ansätze, die auf den Aufzählungsmethoden basieren.

Zur Erkennung der besten Hypothesen im Überlappungsfall wurde im EDSI-Ansatz eine neue statistische Methode entwickelt (siehe Kapitel 3.4.1.1), die die Frequenz des letzten Wortes in der Hypothese ausnutzt. Diese neue Erkennungsmethode ermöglicht eine Erhöhung von Präzision und Recall der extrahierten Strukturen. Anschließend konvertierte der EDSI-Ansatz die

eingegebenen Korpora zu den Treebanks, die dazu dienen können, kontextfreie Grammatiken und weiterhin lokale Grammatiken zu induzieren.

Da es in der vorliegenden Arbeit um die Effizienzerhöhung geht, wird der EDSI-Ansatz auf einem englischen und einem deutschen Korpus quantitativ evaluiert und mit dem klassischen ABL-Ansatz in einer gleichen Umgebung verglichen. Der EDSI-Ansatz besteht wie der ABL-Ansatz aus zwei Lernphasen. Das Greedy-Lernen sammelt einfach alle möglichen Hypothesen, die durch das nachstehende Selektion-Lernen verfeinert werden. Die Ergebnisse (Zeitaufwand, Präzision und Recall etc.) aus den beiden Lernphasen wurden evaluiert und mit den entsprechenden Phasen aus dem ABL verglichen.

Wie die Evaluation aufzeigt, ist der EDSI-Ansatz im Vergleich zu den klassischen Ansätzen viel effizienter. Dabei kann der Zeitaufwand der Verarbeitung enorm reduziert werden (siehe Kapitel 4.3.1.1 und Kapitel 4.3.2.1). Nicht nur die hohe Effizienz sondern auch die Verbesserung der Resultate zeichnet EDSI aus (siehe Kapitel 4.3.1.2 und Kapitel 4.3.2.2).

Allerdings stehen beim EDSI-Ansatz noch ein paar Punkte aus, die künftig erforscht werden sollten:

- Zunächst sind Präzision und Recall von allen statistischen oder lernbasierten Systemen bis heute noch nicht reif genug für den praktischen Einsatz. Auch der EDSI-Ansatz macht in diesem Punkt keine Ausnahme. Da das Greedy-Lernen bereits alle möglichen Hypothesen findet, liegt es wahrscheinlich daran, dass das Selektion-Lernen immer noch viele falsche Hypothesen aufnimmt. Daher sollte in Zukunft eine bessere Methode des Selektion-Lernens entwickelt werden.

- Außerdem soll die Induktion der lokalen Grammatiken weiter automatisiert werden (z.B. kann die Verfeinerung der äquivalenten Hypothesen durch ein Lexikon erleichtet werden).

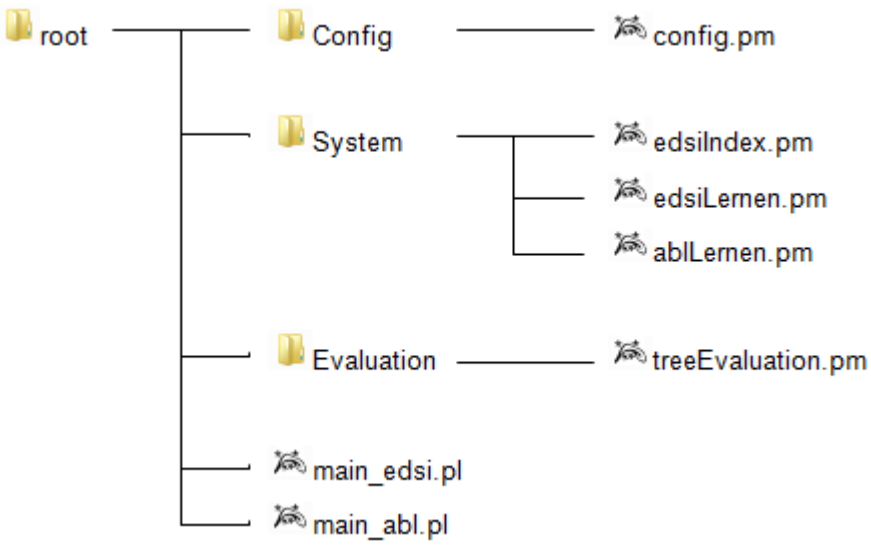
- Aus technischer Sicht verbraucht der EDSI-Ansatz mehr Speicherplatz (insbesondere bei der Extraktion der Patterns) als andere Ansätze, weil EDSI mehr zusätzliche Matrizes in der Verarbeitung benötigt. Obwohl dies heutzutage wenig problematisch ist, dürfte der Algorithmus in der weiteren Arbeit verbessert werden.

Da auch die Vorstellung der lokalen Grammatiken eine Zielsetzung der vorliegenden Arbeit ist, wurde anhand eines Beispiels die Vorgehensweise dieser Grammatikentwicklung dargestellt. Die lokalen Grammatiken wurden mit Unitex erstellt und präsentiert (siehe Kapitel 5.1.2, Kapitel 5.2.2 und Kapitel 5.3.2). Aus den gecrawlten Webseiten wurden die gewünschten Firmeninformationen extrahiert, die sehr hilfreich dabei sein können, die Suchqualität zu verbessern.

7. Anhang

7.1 Source-Code

7.1.1 Hierarchie des Code-Systems



Anmerkung: die Hierarchie des EDSI-Systems wird in der Konfigurationsdatei „config.pm“ definiert.

7.1.2 Darstellung der Codes

```
#################################################################
#
#    config.pm
#
#    - ConfigFile
#
#################################################################
package config;
use strict;
use warnings;
require Exporter;
my @ISA = qw( Exporter );
my @EXPORT = qw( getConfig );

#################################################################
#
#    Contructor initieraliseren
#
sub new {

    my $class    = $_[0];
    my $Modus    = $_[1] || 1;
    my $Strategy = $_[2] || 1;
    my $Systype  = $_[3] || 'selection';
    my $Diffleng = $_[4] || 5;
    my $Methode  = $_[5] || 'edsi';
    my $Source   = $_[6];
    my $Docnum   = $_[7];

    my $self = {};
    $self->{'Modus'}        = $Modus;
    $self->{'Strategy'}     = $Strategy;
    $self->{'Systype'}      = $Systype;
    $self->{'Diffleng'}     = $Diffleng;
    $self->{'Methode'}      = $Methode;
    $self->{'Source'}       = $Source;
    $self->{'Docnum'}       = $Docnum;
    $self->{'dirDataPool'}  = "../DataPool";
    $self->{'dirCorpora'}   = "$self->{'dirDataPool'}/Corpora";
    $self->{'dirResults'}   = "$self->{'dirDataPool'}/Results";

    bless( $self, $class );
}

#################################################################
#
#    Konfiguation zurueckliefern
#
sub getConfig {
    my $self = $_[0];

    return $self;
}
```

```
1;
```

```
#################################################################
#
#   edsiIndex.pm
#
#   - Index aufbauen
#
#################################################################
package edsiIndex;
use strict;
use warnings;
use Data::Dumper;
use Time::HiRes qw/gettimeofday/;
require Exporter;
my @ISA = qw( Exporter );
my @EXPORT = qw( getIndex );

#################################################################
#
#   Contructor initieraliseren
#
sub new {

    my $class  = $_[0];
    my $config = $_[1];

    my $self = {};
    $self->{'config'} = $config;

    bless( $self, $class );
}

#################################################################
#
#   getIndex: Alle Sätze indexieren
#
sub getIndex {

    my $self   = $_[0];
    my $corpus = $_[1];

    my $timeb = gettimeofday();

    my $anchorID = 0;
    my $counter  = 0;
    open( my $FHIN, '<:utf8', "$self->{'config'}-
>{'dirCorpora'}/$corpus" ) || die "$! $corpus";
    while( <$FHIN> ) {

        # Ein Satz parsern
        if( /^(\d+)\s+(.*?)\s*$/ ) {

            $counter++;
            last if( $counter > $self->{'config'}->{'Docnum'} );

            my $sentID = $1;
```

```
            my @segs    = split( /\s+/, $2 );
            pop( @segs ) if( $segs[-1] eq ',' || $segs[-1] eq '.' ||
$segs[-1] eq '?' );

            # Inhalt ins Hash speichern
            $self->{'sentences'}->{$sentID} = \@segs;
            $self->{'senlength'}->{$sentID} = scalar( @segs );

            my $onesentcount = 0;
            my %onesentnotiz = ();
            while( $onesentcount < $self->{'senlength'}->{$sentID} ) {

                my $onepos = $onesentcount + 1;
                if( exists $onesentnotiz{$segs[$onesentcount]} ) {
                    $onesentnotiz{$segs[$onesentcount]} .= '|' . $one-
pos;
                }
                else {
                    $onesentnotiz{$segs[$onesentcount]} = $onepos;
                }

                $onesentcount++;
            }

            while( my( $word, $poss ) = each %onesentnotiz ) {

                $self->{'indexfreq'}->{$word}++;
                if( ! exists $self->{'indexid'}->{$word} ) {
                    $anchorID++;
                    $self->{'indexid'}->{$word} = $anchorID;
                }

                $self->{'index'}->{$self->{'indexid'}->{$word}} .= ";"
if( exists $self->{'index'}->{$self->{'indexid'}->{$word}} );
                $self->{'index'}->{$self->{'indexid'}->{$word}} .=
"$sentID,$poss";
            }
        }
    }
    close( $FHIN );

    my $freqlist = '';
    if( $corpus =~m/^DE.*?/ ) {
        $freqlist = "DE_IDS_freq.txt";
    }
    else {
        $freqlist = "EN_OANC_journal_freq.txt";
    }
    open( my $FREQ, '<:utf8', "$self->{'config'}-
>{'dirCorpora'}/$freqlist" ) || die "$! $freqlist";
    while( <$FREQ> ) {
        chomp;
        if( /^(\S+)\s+(\d+)$/ ) {
            if( exists $self->{'indexfreq'}->{$1} ) {
                $self->{'indexfreq'}->{$1} += $2;
            }
        }
```

```
    }
    close( $FREQ );

    my $timee = gettimeofday();
    my $timetoindex = $timee - $timeb;

    return( $self, $timetoindex );
}

#############################################################
#
#   generatorID: ID generasieren
#
sub generatorID {

    my $self       = $_[0];
    my $paramCount = $_[1];

    # Füllt die leeren Stellen mit "0"
    my $prefix = '0' x ( 8 - length( $paramCount ) );
    my $id = $prefix . $paramCount;

    return $id;
}

1;
```

```
#################################################################
#
#   edsiLernen.pm
#
#   Greedy- und Selektion-Lernen
#
#################################################################
package edsiLernen;
use strict;
use warnings;
use Data::Dumper;
use Time::HiRes qw/gettimeofday/;
require Exporter;
my @ISA = qw( Exporter );
my @EXPORT = qw( processorSearch );

#############################################################
#
#   Contructor initieraliseren
#
sub new {

    my $class  = $_[0];
    my $data   = $_[1];

    my $self = {};
    $self = $data;
```

```
    bless( $self, $class );
}

############################################################
#
#   processorSearch: Learnen starten
#
sub processorSearch {

    my $self        = $_[0];
    my $timetoindex = $_[1];

    my $timeb = gettimeofday();

    # Falls die alten Datei existieren, entfernen
    my $treebankt = "$self->{'config'}->{'dirResults'}/$self-
>{'config'}->{'Source'}" . "_" . "$self->{'config'}->{'Docnum'}" .
"_" . "$self->{'config'}->{'Methode'}" . "_treebankT.txt";
    unlink $treebankt if( -e $treebankt );
    my $fuzzytreebankt = "$self->{'config'}->{'dirResults'}/$self-
>{'config'}->{'Source'}" . "_" . "$self->{'config'}->{'Docnum'}" .
"_" . "$self->{'config'}->{'Methode'}" . "_fuzzytreebankT.txt";
    unlink $fuzzytreebankt if( -e $fuzzytreebankt );
    my $treebanks = "$self->{'config'}->{'dirResults'}/$self-
>{'config'}->{'Source'}" . "_" . "$self->{'config'}->{'Docnum'}" .
"_" . "$self->{'config'}->{'Methode'}" . "_treebankS.txt";
    unlink $treebanks if( -e $treebanks && $self->{'config'}->{'Modus'}
== 2 );

    for my $targetsid( sort{$a <=> $b} keys %{$self->{'sentences'}} ) {
        my $targetsid = $sentids{$temid};

        # Greedy-Lernen
        my( $hypotheses, $hypoids, $sortweight, $sourcecons ) = $self-
>GreedyLernen( $targetsid );

        # Selektion-Lernen
        if( $self->{'config'}->{'Systype'} ne "noselection" ) {
            $self->SelektionLernen( $targetsid, $hypotheses, $hypoids,
$sortweight, $sourcecons );
        }
    }

    my $timee = gettimeofday();
    my $timetotest = $timee - $timeb + $timetoindex;
    open( my $LOG, '>>:utf8', "$self->{'config'}-
>{'dirResults'}/logs.txt" ) || die "$!";
    print $LOG "Evaluation: $self->{'config'}->{'Methode'}, $self-
>{'config'}->{'Source'}, $self->{'config'}->{'Docnum'}, $self-
>{'config'}->{'Systype'}, $self->{'config'}->{'Modus'}, $self-
>{'config'}->{'Strategy'}\n";
    print $LOG "Zeitaufwand zu indexieren: $timetoindex s\n";
    print $LOG "Zeitaufwand zu testen:     $timetotest s\n";
    close( $LOG );

    print "Evaluation: $self->{'config'}->{'Methode'}, $self-
>{'config'}->{'Source'}, $self->{'config'}->{'Docnum'}, $self-
```

```
>{'config'}->{'Systype'}, $self->{'config'}->{'Modus'}, $self-
>{'config'}->{'Strategy'}\n";
    print "Zeitaufwand zu indexieren: $timetoindex s\n";
    print "Zeitaufwand zu testen:     $timetotest s\n";

    return 1;
}

#############################################################
#
#   GreedyLernen: alle moeglichen Hypothesen extrahieren
#
sub GreedyLernen {

    my $self      = $_[0];
    my $targetsid = $_[1];

    my( $mapping, $tracing ) = $self->extractPatterns( $targetsid );

    # Die Hypothesen aus Target- und Source-Satz extrahieren
    my $hypotheses = {};
    my $hypoids    = {};
    my $sortweight = {};
    my $sourcecons = {};
    if( $self->{'config'}->{'Modus'} == 2 ) {
        if( $self->{'config'}->{'Strategy'} == 2 ) {
            ( $hypotheses, $hypoids, $sortweight, $sourcecons ) =
$self->extractHypothesesM2S2( $targetsid, $mapping );
        }
        else {
            ( $hypotheses, $hypoids, $sortweight, $sourcecons ) =
$self->extractHypothesesM2S1( $targetsid, $mapping );
        }
    }
    else {
        if( $self->{'config'}->{'Strategy'} == 2 ) {
            ( $hypotheses, $hypoids, $sortweight, $sourcecons ) =
$self->extractHypothesesM1S2( $targetsid, $mapping );
        }
        else {
            ( $hypotheses, $hypoids, $sortweight, $sourcecons ) =
$self->extractHypothesesM1S1( $targetsid, $mapping );
        }
    }

    #
    $self->constructFuzzyTreebank( $targetsid, $hypotheses, $hypoids );

    return( $hypotheses, $hypoids, $sortweight, $sourcecons );
}

#############################################################
#
#   extractPatterns: alle moeglichen Patterns extrahieren
#
sub extractPatterns {
```

```
    # Parameter
    my $self      = $_[0];
    my $targetsid = $_[1];

    # Von links nach rechts suchen
    my %tracing  = ();   # SourceID -> SouPoss => @SouPoss
    my %mapping  = ();   # SourceID -> SouPoss => @TarPoss
    my %minbegin = ();   # SourceID => MinPosBegin
    my $pointer  = 0;
    foreach my $oneseg( @{$self->{'sentences'}->{$targetsid}} ) {

        $pointer++;

        my $accindex = $self->{'index'}->{$self->{'indexid'}-
>{$oneseg}};
        my @sourceindex = split( /;/, $accindex );

        next unless( $sourceindex[1] );

        while( my $oneindex = shift @sourceindex ) {

            my( $sourcesid, $positions ) = ( $oneindex
=~/^(\d+),(.*?)$/ );

            next if( $sourcesid == $targetsid );

            $minbegin{$sourcesid} = '' if( !exists $minbe-
gin{$sourcesid} );
            my @positions = split( /\|/, $positions );
            while( my $onespos = shift @positions ) {
                if( $minbegin{$sourcesid} eq '' || $onespos < $minbe-
gin{$sourcesid} ) {
                    $minbegin{$sourcesid} = $onespos;
                    push( @{$tracing{$sourcesid}{$onespos}}, $ones-
pos );
                    push( @{$mapping{$sourcesid}{$onespos}}, $poin-
ter );
                }
                else {
                    foreach my $positions( keys %{$trac-
ing{$sourcesid}} ) {
                        my $targetendpos   = ${$map-
ping{$sourcesid}{$positions}}[-1];
                        my $sourceendpos   = ${$trac-
ing{$sourcesid}{$positions}}[-1];
                        my $sourcebeginpos = ${$trac-
ing{$sourcesid}{$positions}}[0];
                        next if( $pointer <= $targetendpos );
                        my @newtarpos = ();
                        my @newsoupos = ();
                        if( $onespos > $sourceendpos ) {
                            @newsoupos =
( @{$tracing{$sourcesid}{$positions}}, $onespos );
                            @newtarpos =
( @{$mapping{$sourcesid}{$positions}}, $pointer );
                            my $newpositions = join( ' ', @newsoupos );
                            $tracing{$sourcesid}{$newpositions} =
```

```
\@newsoupos;
                                $mapping{$sourcesid}{$newpositions} =
\@newtarpos;
                                delete( $tracing{$sourcesid}{$positions} );
                                delete( $mapping{$sourcesid}{$positions} );
                            }
                            elsif( $onespos < $sourceendpos && $onespos >
$sourcebeginpos ) {
                                my $curendpos = $sourceendpos;
                                my @cursoupos =
@{$tracing{$sourcesid}{$positions}};
                                my $tarendpos = '';
                                my @tarsoupos =
@{$mapping{$sourcesid}{$positions}};
                                while( $onespos <= $curendpos ) {
                                    $curendpos = pop @cursoupos;
                                    $tarendpos = pop @tarsoupos;
                                }
                                @newsoupos = ( @cursoupos, $curendpos,
$onespos );
                                @newtarpos = ( @tarsoupos, $tarendpos,
$pointer );
                                my $newpositions = join( ' ', @newsoupos );
                                $tracing{$sourcesid}{$newpositions} =
\@newsoupos;
                                $mapping{$sourcesid}{$newpositions} =
\@newtarpos;
                            }
                            else {
                                next;
                            }
                        }
                    }
                }
            }
        }

    return( \%mapping, \%tracing );
}

############################################################
#
#   extractHypothesesM1S1 (Modus 1, Strategie 1)
#
sub extractHypothesesM1S1 {

    my $self     = $_[0];
    my $targetid = $_[1];
    my $mapping  = $_[2];

    my @sententity = @{$self->{'sentences'}->{$targetid}};
    my $tarslength = $self->{'senlength'}->{$targetid};

    my %hypotheses = ();    # Slots
    my %hypoids    = ();
    my %sourcecons = ();
    my %sortweight = ();    # Slot mit jeweiligem Gewicht
```

```
    my %targetpars = ();
    my $counter    = 1;
    while( my( $sourceid, $oneblock ) = each( %$mapping ) ) {
        while( my( $positions, $onepattern ) = each( %$oneblock ) ) {
            my @parnodestar = @$onepattern;
            my $targetpart = join( ' ', @parnodestar );
            next if( exists $targetpars{$targetpart} );
            $targetpars{$targetpart} = 0;
            if( $parnodestar[0] > 1 ) {
                my $slote = $parnodestar[0] - 1;
                if( !exists $hypotheses{"1,$slote"} ) {
                    my $hypoid = "S" . $targetid . "C" . $counter++;
                    $hypoids{"1,$slote"} = $hypoid;
                    @{$hypotheses{"1,$slote"}} = (1 .. $slote);
                    $sortweight{"1,$slote"}    = $self->{'indexfreq'}-
>{$sententity[$slote-1]};
                }
            }
            if( $parnodestar[-1] < $tarslength ) {
                my $slotb = $parnodestar[-1] + 1;
                if( !exists $hypotheses{"$slotb,$tarslength"} ) {
                    my $hypoid = "S" . $targetid . "C" . $counter++;
                    $hypoids{"$slotb,$tarslength"} = $hypoid;
                    @{$hypotheses{"$slotb,$tarslength"}} = ($slotb ..
$tarslength);
                    $sortweight{"$slotb,$tarslength"}    = $self-
>{'indexfreq'}->{$sententity[$tarslength-1]};
                }
            }
            for( my $num=0; $num<$#parnodestar; $num++ ) {
                my $slotbegin = $parnodestar[$num] + 1;
                my $slotend   = $parnodestar[$num+1] - 1;
                if( $slotbegin <= $slotend && !exists $hypothes-
es{"$slotbegin,$slotend"} ) {
                    my $hypoid = "S" . $targetid . "C" . $counter++;
                    $hypoids{"$slotbegin,$slotend"}       = $hypoid;
                    @{$hypotheses{"$slotbegin,$slotend"}} = ($slotbe-
gin .. $slotend);
                    $sortweight{"$slotbegin,$slotend"}    = $self-
>{'indexfreq'}->{$sententity[$slotend-1]};
                }
            }
        }
    }

    return( \%hypotheses, \%hypoids, \%sortweight, \%sourcecons );
}

############################################################
#
#   extractHypothesesM1S2 (Modus 1, Strategie 2)
#
sub extractHypothesesM1S2 {

    my $self     = $_[0];
    my $targetid = $_[1];
    my $mapping  = $_[2];
```

```
    my @sententity = @{$self->{'sentences'}->{$targetid}};
    my $tarslength = $self->{'senlength'}->{$targetid};

    my %hypotheses = ();     # Slots
    my %hypoids    = ();
    my %sourcecons = ();
    my %sortweight = ();     # Slot mit jeweiligem Gewicht
    my $counter    = 1;
    while( my( $sourceid, $oneblock ) = each( %$mapping ) ) {
        while( my( $positions, $onepattern ) = each( %$oneblock ) ) {
            my @parnodestar = @$onepattern;
            if( $parnodestar[0] > 1 ) {
                my $slote = $parnodestar[0] - 1;
                if( !exists $hypotheses{"1,$slote"} ) {
                    my $hypoid = "S" . $targetid . "C" . $counter++;
                    $hypoids{"1,$slote"} = $hypoid;
                    @{$hypotheses{"1,$slote"}} = (1 .. $slote);
                    $sortweight{"1,$slote"}++;
                }
            }
            if( $parnodestar[-1] < $tarslength ) {
                my $slotb = $parnodestar[-1] + 1;
                if( !exists $hypotheses{"$slotb,$tarslength"} ) {
                    my $hypoid = "S" . $targetid . "C" . $counter++;
                    $hypoids{"$slotb,$tarslength"} = $hypoid;
                    @{$hypotheses{"$slotb,$tarslength"}} = ($slotb ..
$tarslength);
                    $sortweight{"$slotb,$tarslength"}++;
                }
            }
            for( my $num=0; $num<$#parnodestar; $num++ ) {
                my $slotbegin = $parnodestar[$num] + 1;
                my $slotend   = $parnodestar[$num+1] - 1;
                if( $slotbegin <= $slotend && !exists $hypothes-
es{"$slotbegin,$slotend"} ) {
                    my $hypoid = "S" . $targetid . "C" . $counter++;
                    $hypoids{"$slotbegin,$slotend"}       = $hypoid;
                    @{$hypotheses{"$slotbegin,$slotend"}} = ($slotbe-
gin .. $slotend);
                    $sortweight{"$slotbegin,$slotend"}++;
                }
            }
        }
    }

    return( \%hypotheses, \%hypoids, \%sortweight, \%sourcecons );
}

############################################################
#
#   extractHypothesesM2S1 (Modus 2, Strategie 1)
#
sub extractHypothesesM2S1 {

    my $self     = $_[0];
    my $targetid = $_[1];
```

```
    my $mapping  = $_[2];

    my @sententity = @{$self->{'sentences'}->{$targetid}};
    my $tarslength = $self->{'senlength'}->{$targetid};

    my %hypotheses = ();     # Slots
    my %hypoids    = ();
    my %sortweight = ();     # Slot mit jeweiligem Gewicht
    my %sourcecons = ();     # Konstituenten im Source
    my %pseudohyps = ();     #
    my $counter    = 1;
    while( my( $sourceid, $oneblock ) = each( %$mapping ) ) {
        my $souslength = $self->{'senlength'}->{$sourceid};
        while( my( $positions, $onepattern ) = each( %$oneblock ) ) {
            my @parnodestar = @$onepattern;
            my @parnodessou = split( / /, $positions );
            if( $parnodestar[0] > 1 ) {
                my $slote = $parnodestar[0] - 1;
                if( !exists $hypotheses{"1,$slote"} ) {
                    my $hypoid = "S" . $targetid . "C" . $counter++;
                    $hypoids{"1,$slote"} = $hypoid;
                    @{$hypotheses{"1,$slote"}} = (1 .. $slote);
                    $sortweight{"1,$slote"}    = $self->{'indexfreq'}-
>{$sententity[$slote-1]};
                }
                if( $parnodessou[0] > 1 ) {
                    @{$sourcecons{"1,$slote"}{$sourceid}} = (1 .. $par-
nodessou[0]-1);
                }
                else {
                    @{$sourcecons{"1,$slote"}{$sourceid}} = (0);
                }
            }
            if( $parnodestar[-1] < $tarslength ) {
                my $slotb = $parnodestar[-1] + 1;
                if( !exists $hypotheses{"$slotb,$tarslength"} ) {
                    my $hypoid = "S" . $targetid . "C" . $counter++;
                    $hypoids{"$slotb,$tarslength"} = $hypoid;
                    @{$hypotheses{"$slotb,$tarslength"}} = ($slotb ..
$tarslength);
                    $sortweight{"$slotb,$tarslength"}    = $self-
>{'indexfreq'}->{$sententity[$tarslength-1]};
                }
                if( $parnodessou[-1] < $souslength ) {
                    @{$sourcecons{"$slotb,$tarslength"}{$sourceid}} =
($parnodessou[-1]+1 .. $souslength);
                }
                else {
                    @{$sourcecons{"$slotb,$tarslength"}{$sourceid}} =
(0);
                }
            }
            for( my $num=0; $num<$#parnodestar; $num++ ) {
                my $slotbegin = $parnodestar[$num] + 1;
                my $slotend   = $parnodestar[$num+1] - 1;
                if( $slotbegin <= $slotend ) {
                    if( !exists $hypotheses{"$slotbegin,$slotend"} ) {
```

```
                        my $hypoid = "S" . $targetid . "C" . $coun-
ter++;
                        $hypoids{"$slotbegin,$slotend"}       = $hypo-
id;
                        @{$hypotheses{"$slotbegin,$slotend"}} = ($slot-
begin .. $slotend);
                        $sortweight{"$slotbegin,$slotend"}    = $self-
>{'indexfreq'}->{$sententity[$slotend-1]};
                    }
                    my $slotbsou = $parnodessou[$num] + 1;
                    my $slotesou = $parnodessou[$num+1] - 1;
                    if( $slotbsou <= $slotesou ) {

@{$sourcecons{"$slotbegin,$slotend"}{$sourceid}} = ($slotbsou .. $slo-
tesou);
                    }
                    else {
@{$sourcecons{"$slotbegin,$slotend"}{$sourceid}} = (0);
                    }
                }
            }
        }
    }

    return( \%hypotheses, \%hypoids, \%sortweight, \%sourcecons );
}

############################################################
#
#   extractHypothesesM2S2 (Modus 2, Strategie 2)
#
sub extractHypothesesM2S2 {

    my $self     = $_[0];
    my $targetid = $_[1];
    my $mapping  = $_[2];

    my @sententity = @{$self->{'sentences'}->{$targetid}};
    my $tarslength = $self->{'senlength'}->{$targetid};
    my %hypotheses = ();    # Slots
    my %hypoids    = ();
    my %sortweight = ();    # Slot mit jeweiligem Gewicht
    my %sourcecons = ();    # Konstituenten im Source
    my $counter    = 1;
    while( my( $sourceid, $oneblock ) = each( %$mapping ) ) {
        my $souslength = $self->{'senlength'}->{$sourceid};
        while( my( $positions, $onepattern ) = each( %$oneblock ) ) {
            my @parnodestar = @$onepattern;
            my @parnodessou = split( / /, $positions );
            if( $parnodestar[0] > 1 ) {
                my $slote = $parnodestar[0] - 1;
                if( !exists $hypotheses{"1,$slote"} ) {
                    my $hypoid = "S" . $targetid . "C" . $counter++;
                    $hypoids{"1,$slote"} = $hypoid;
                    @{$hypotheses{"1,$slote"}} = (1 .. $slote);
                    $sortweight{"1,$slote"}++;
                }
```

```
                if( $parnodessou[0] > 1 ) {
                    @{$sourcecons{"1,$slote"}{$sourceid}} = (1 .. $par-
nodessou[0]-1);
                }
                else {
                    @{$sourcecons{"1,$slote"}{$sourceid}} = (0);
                }
            }
            if( $parnodestar[-1] < $tarslength ) {
                my $slotb = $parnodestar[-1] + 1;
                if( !exists $hypotheses{"$slotb,$tarslength"} ) {
                    my $hypoid = "S" . $targetid . "C" . $counter++;
                    $hypoids{"$slotb,$tarslength"} = $hypoid;
                    @{$hypotheses{"$slotb,$tarslength"}} = ($slotb ..
$tarslength);
                    $sortweight{"$slotb,$tarslength"}++;
                }
                if( $parnodessou[-1] < $souslength ) {
                    @{$sourcecons{"$slotb,$tarslength"}{$sourceid}} =
($parnodessou[-1]+1 .. $souslength);
                }
                else {
                    @{$sourcecons{"$slotb,$tarslength"}{$sourceid}} =
(0);
                }
            }
            for( my $num=0; $num<$#parnodestar; $num++ ) {
                my $slotbegin = $parnodestar[$num] + 1;
                my $slotend   = $parnodestar[$num+1] - 1;
                if( $slotbegin <= $slotend ) {
                    if( !exists $hypotheses{"$slotbegin,$slotend"} ) {
                        my $hypoid = "S" . $targetid . "C" . $coun-
ter++;
                        $hypoids{"$slotbegin,$slotend"}       = $hypo-
id;
                        @{$hypotheses{"$slotbegin,$slotend"}} = ($slot-
begin .. $slotend);
                        $sortweight{"$slotbegin,$slotend"}++;
                    }
                    my $slotbsou = $parnodessou[$num] + 1;
                    my $slotesou = $parnodessou[$num+1] - 1;
                    if( $slotbsou <= $slotesou ) {

@{$sourcecons{"$slotbegin,$slotend"}{$sourceid}} = ($slotbsou .. $slo-
tesou);
                    }
                    else {
@{$sourcecons{"$slotbegin,$slotend"}{$sourceid}} = (0);
                    }
                }
            }
        }
    }

    return( \%hypotheses, \%hypoids, \%sortweight, \%sourcecons );
}
```

```
############################################################
#
#   constructFuzzyTreebank
#
sub constructFuzzyTreebank {

    my $self       = $_[0];
    my $targetid   = $_[1];
    my $hypotheses = $_[2];
    my $hypoids    = $_[3];

    my @sententity = @{$self->{'sentences'}->{$targetid}};
    my $tarslength = $self->{'senlength'}->{$targetid};
    my $filename = "$self->{'config'}->{'dirResults'}/$self-
>{'config'}->{'Source'}"
                 . "_" . "$self->{'config'}->{'Docnum'}"
                 . "_" . "$self->{'config'}->{'Methode'}"
                 . "_fuzzytreebankT.txt";
    open( my $RES, '>>:utf8', "$filename" ) || die "$!";
    print $RES "%% sent. no. $targetid\n";
    foreach my $onehypo( sort{$a cmp $b} keys %$hypotheses ) {
        my @onecons = @{$hypotheses->{$onehypo}};
        foreach my $num( 0 .. $#onecons ) {
            $onecons[$num] = $sententity[$onecons[$num]-1];
        }
        print $RES "$hypoids->{$onehypo}\t" . join( ' ', @onecons ) .
"\n";
    }
    print $RES "\n\n";
    close( $RES );

    return 1;
}

############################################################
#
#   SelektionLernen: Die Hypothesen verfeinern
#
sub SelektionLernen {

    my $self       = $_[0];
    my $targetid   = $_[1];
    my $hypotheses = $_[2];
    my $hypoids    = $_[3];
    my $sortweight = $_[4];
    my $sourcecons = $_[5];

    my $constituents = $self->selectConstituents( $hypotheses, $sort-
weight );

    # Baumstruktur T aufbauen
    my( $tree, $nodes, $hypos ) = $self->constructTreeT( $targetid,
$constituents, $hypoids );

    # Treebank aufbauen
    $self->constructTreebank( $targetid, $tree, $nodes, $hypos, $sour-
```

```
cecons );

    return 1;
}


############################################################
#
#   selectConstituents: Konstitutenten aus den Hypothesen selektieren
#
sub selectConstituents {

    my $self       = $_[0];
    my $hypotheses = $_[1];
    my $sortweight = $_[2];

    my %constituents = ();
    my $sumpat = 0;
    foreach my $oneslot( sort{$$sortweight{$a} <=> $$sortweight{$b}}
keys %$sortweight ) {
        my( $slotbegin, $slotend ) = ( $oneslot =~m/^(\d+),(\d+)$/ );
        if( $slotbegin == $slotend ) {
            $constituents{$oneslot} = 1;
            next;
        }
        if( !exists $constituents{$oneslot} ) {
            my $checker = 0;
            if( $sumpat == 0 ) {
                $checker = 1;
            }
            else {
                for my $oslot( keys %constituents ) {
                    my( $onepatbegin, $onepatend ) = ( $oslot
=~m/^(\d+),(\d+)$/ );
                    if( ($slotbegin <= $onepatbegin && $slotend >=
$onepatend) ||
                        ($slotbegin >= $onepatbegin && $slotend <=
$onepatend) ||
                        ($slotbegin > $onepatend) || ($slotend < $one-
patbegin) ) {
                        $checker = 1;
                    }
                    else {
                        $checker = 0;
                        last;
                    }
                }
            }
            if( $checker == 1 ) {
                $constituents{$oneslot} = $slotend - $slotbegin + 1;
                $sumpat = 1;
            }
        }
    }

    return( \%constituents );
}
```

```
#############################################################
#
#   constructTreeT: Baumstruktur aufbauen
#
sub constructTreeT {

    my $self     = $_[0];
    my $targetid = $_[1];
    my $constits = $_[2];
    my $hypoids  = $_[3];

    my @targetentity = @{$self->{'sentences'}->{$targetid}};
    my $targetlength = $self->{'senlength'}->{$targetid};
    my %nodes = ();
    my %hypos = ();
    my @targs = @targetentity;
    foreach my $onehypo( sort{$$constits{$a} <=> $$constits{$b}}
keys %$constits ) {
        my( $slotbegin, $slotend ) = ( $onehypo =~m/^(\d+),(\d+)$/ );
        my $hypoid = $hypoids->{$onehypo};
        my @poss = ($slotbegin .. $slotend);
        foreach my $pos( @poss ) {
            $nodes{$hypoid} .= "$targetentity[$pos-1] ";
        }
        $hypos{$hypoid} = $onehypo;
        $targs[$slotbegin-1] = "(" . $hypoid . " " . $targs[$slotbegin-
1];
        $targs[$slotend-1]   = $targs[$slotend-1] . ")";
    }

    my $tree = '';
    $tree = join( " ", @targs );
    my @tree = split( //, $tree );
    my $count = '';
    my $level = 0;
    foreach my $num( 0 .. $#tree ) {
        if( $tree[$num] eq '(' ) {
            $count = '(';
            $level++;
            $tree[$num] = "\n" . "\t" x $level . $tree[$num];
        }
        elsif( $tree[$num] eq ')' ) {
            if( $count eq ')' ) {
                $tree[$num] = "\n" . "\t" x $level . $tree[$num] .
"\n" . "\t" x $level;
            }
            else {
                $tree[$num] = $tree[$num] . "\n" . "\t" x $level;
            }
            $count = ')';
            $level--;
        }
    }
    $tree = join( '', @tree );
    $tree =~ s/\n+\s*\n+/\n/gs;
    $tree =~ s/\n+\s+$/\n/gs;
```

```
    return( $tree, \%nodes, \%hypos );
}

############################################################
#
#   constructTreebank
#
sub constructTreebank {

    my $self       = $_[0];
    my $targetid   = $_[1];
    my $tree       = $_[2];
    my $nodes      = $_[3];
    my $hypos      = $_[4];
    my $sourcecons = $_[5];

    $self->treebankT( $targetid, $tree, $nodes );
    $self->treebankS( $targetid, $tree, $nodes, $hypos, $sourcecons )
if( $self->{'config'}->{'Modus'} == 2 );

    return 1;
}

############################################################
#
#   treebankT
#
sub treebankT {

    my $self     = $_[0];
    my $targetid = $_[1];
    my $tree     = $_[2];
    my $nodes    = $_[3];
    my $filename = "$self->{'config'}->{'dirResults'}/$self-
>{'config'}->{'Source'}"
                 . "_" . "$self->{'config'}->{'Docnum'}"
                 . "_" . "$self->{'config'}->{'Methode'}"
                 . "_treebankT.txt";
    open( my $RES, '>>:utf8', "$filename" ) || die "$!";
    print $RES "%% sent. no. $targetid\n";
    print $RES "S($tree)\n\n";
    foreach my $conid( sort{$a cmp $b} keys %$nodes ) {
        print $RES "$conid\t$$nodes{$conid}\n";
    }
    print $RES "\n\n";
    close( $RES );

    return 1;
}

############################################################
#
#   treebankS
#
sub treebankS {
```

```
    my $self     = $_[0];
    my $targetid = $_[1];
    my $tree     = $_[2];
    my $nodes    = $_[3];
    my $pshypo   = $_[4];
    my $sourcons = $_[5];
    my $filename = "$self->{'config'}->{'dirResults'}/$self-
>{'config'}->{'Source'}"
                 . "_" . "$self->{'config'}->{'Docnum'}"
                 . "_" . "$self->{'config'}->{'Methode'}"
                 . "_treebankS.txt";
    open( my $RES, '>>:utf8', "$filename" ) || die "$!";
    print $RES "%% sent. no. $targetid\n";
    print $RES "S($tree)\n\n";
    foreach my $conid( sort{$a cmp $b} keys %$nodes ) {
        my $pscons = $$pshypo{$conid};
        my( $tarb, $tare ) = ( $pscons =~m/^(\d+),(\d+)$/ );
        my $tardis = $tare - $tarb;
        print $RES "$conid\t$$nodes{$conid}\n";

        my %sourold = ();
        my %sourdis = ();
        while( my( $sourceid, $poss ) = each %{$$sourcons{$pscons}} ) {
            my $sourcons = '';
            my $soudis   = 0;
            if( $$poss[0] == 0 ) {
                $sourcons = '<E>';
            }
            else {
                my @sententity = @{$self->{'sentences'}->{$sourceid}};
                my @scons = @$poss;

                foreach my $num( 0 .. $#scons ) {
                    $scons[$num] = $sententity[$scons[$num]-1];
                }
                $sourcons = join( ' ', @scons );
                $soudis   = $$poss[-1] - $$poss[0];
            }
            $sourdis{$sourcons} = abs( $tardis - $soudis );
            $sourold{$sourcons}++;
        }
        foreach my $onesour( sort{$sourold{$b} <=> $sourold{$a}}
keys %sourold ) {
            if( $sourold{$onesour} > 1 ) {
                print $RES "\t$onesour\t$sourold{$onesour}\n";
            }
            else {
                if( $sourdis{$onesour} < $self->{'config'}-
>{'Diffleng'} ) {
                    print $RES "\t$onesour\t$sourold{$onesour}\n";
                }
            }
        }
        print $RES "\n";
    }
    print $RES "\n\n";
    close( $RES );
```

```
    return 1;
}

1;
```

```
#############################################################
#
#   ablLernen.pm
#
#       Alignment- und Selektion-Lernen
#
#############################################################
package ablLernen;
use strict;
use warnings;
use Data::Dumper;
use Time::HiRes qw/gettimeofday/;
require Exporter;
my @ISA = qw( Exporter );
my @EXPORT = qw( processorSearch );


#############################################################
#
#   Contructor initieraliseren
#
sub new {

    my $class  = $_[0];
    my $config = $_[1];

    my $self = {};
    $self->{'config'} = $config;

    bless( $self, $class );
}

#############################################################
#
#   processorSearch: das Lernen starten
#
sub processorSearch {

    my $self   = $_[0];
    my $corpus = $_[1];
    my $timeb = gettimeofday();

    my $treebankt = "$self->{'config'}->{'dirResults'}/$self-
>{'config'}->{'Source'}" . "_" . "$self->{'config'}->{'Docnum'}" .
"_" . "$self->{'config'}->{'Methode'}" . "_treebankT.txt";
    unlink $treebankt if( -e $treebankt );
    my $fuzzytreebankt = "$self->{'config'}->{'dirResults'}/$self-
>{'config'}->{'Source'}" . "_" . "$self->{'config'}->{'Docnum'}" .
"_" . "$self->{'config'}->{'Methode'}" . "_fuzzytreebankT.txt";
```

```
    unlink $fuzzytreebankt if( -e $fuzzytreebankt );
    my $treebanks = "$self->{'config'}->{'dirResults'}/$self-
>{'config'}->{'Source'}" . "_" . "$self->{'config'}->{'Docnum'}" .
"_" . "$self->{'config'}->{'Methode'}" . "_treebankS.txt";
    unlink $treebanks if( -e $treebanks );

    my $counter   = 0;
    my %randomids = ();
    open( my $SENTENCE, '<:utf8', "$self->{'config'}-
>{'dirCorpora'}/$corpus" ) || die "$!";
    while( <$SENTENCE> ) {

        if( /^(\d+)\s+(.*?)\s*$/ ) {

            $counter++;
            last if( $counter > $self->{'config'}->{'Docnum'} );

            my $sentID = $1;
            my @segs   = split( / /, $2 );
            pop( @segs ) if( $segs[-1] eq ',' || $segs[-1] eq '.' ||
$segs[-1] eq '?' );
            $self->{'sentences'}->{$sentID} = \@segs;
            $self->{'senlength'}->{$sentID} = scalar( @segs );
            $randomids{$sentID} = rand( $self->{'config'}-
>{'Docnum'} );
        }
    }
    close( $SENTENCE );
    $counter = 0;
    foreach my $sentID( sort{$randomids{$a} <=> $randomids{$b}}
keys %randomids ) {
        $counter++;
        $self->{'randomids'}->{$sentID} = $counter;
    }

    foreach my $targetsid( sort{$a <=> $b} keys %{$self-
>{'sentences'}} ) {
        my $time1 = localtime();
        print "$time1\tZielsatz: $targetsid \n";

        # Alignment-Lernen
        my( $hypotheses, $hypoids, $sortweight, $sourcecons ) = $self-
>AlignmentLearning( $targetsid );

        # Selektion-Lernen
        if( $self->{'config'}->{'Systype'} ne "noselection" ) {
            $self->SelektionLernen( $targetsid, $hypotheses, $hypoids,
$sortweight, $sourcecons );
        }
    }

    my $timee = gettimeofday();
    my $timetotest = $timee - $timeb;
    open( my $LOG, '>>:utf8', "$self->{'config'}-
>{'dirResults'}/logs.txt" ) || die "$!";
    print $LOG "Evaluation: $self->{'config'}->{'Methode'}, $self-
>{'config'}->{'Source'}, $self->{'config'}->{'Docnum'}, $self-
```

```
>{'config'}->{'Systype'}, $self->{'config'}->{'Modus'}, $self-
>{'config'}->{'Strategy'}\n";
    print $LOG "Zeitaufwand zu testen:     $timetotest s\n";
    close( $LOG );

    print "Evaluation: $self->{'config'}->{'Methode'}, $self-
>{'config'}->{'Source'}, $self->{'config'}->{'Docnum'}, $self-
>{'config'}->{'Systype'}, $self->{'config'}->{'Modus'}, $self-
>{'config'}->{'Strategy'}\n";
    print "Zeitaufwand zu testen:     $timetotest s\n";

    return 1;
}


############################################################
#
#   AlignmentLearning
#
sub AlignmentLearning {

    my $self      = $_[0];
    my $targetsid = $_[1];

    # Patterns finden
    my( $mapping, $tracing ) = $self->extractPatterns( $targetsid );

    # Die Hypothesen aus Target- und Source-Satz extrahieren
    my( $hypotheses, $hypoids, $sortweight, $sourcecons ) = $self-
>extractHypothesesTarget( $targetsid, $mapping );

    #
    $self->constructFuzzyTreebank( $targetsid, $hypotheses, $hypoids );

    return( $hypotheses, $hypoids, $sortweight, $sourcecons );
}

############################################################
#
#   extractPatterns: alle moeglichen Patterns extrahieren
#
sub extractPatterns {

    my $self      = $_[0];
    my $targetsid = $_[1];

    my %tracing  = ();   # SourceID -> SouPoss => @SouPoss
    my %mapping  = ();   # SourceID -> SouPoss => @TarPoss
    foreach my $sourcesid( sort{$a <=> $b} keys %{$self-
>{'sentences'}} ) {
        next if( $sourcesid == $targetsid );

        my( $targetposs, $sourceposs ) = $self->EditDistance( $self-
>{'sentences'}->{$targetsid}, $self->{'sentences'}->{$sourcesid} );
        my $positions = join( ' ', @$sourceposs );
        if( $positions ) {
            $tracing{$sourcesid}{$positions} = $sourceposs;
```

```
            $mapping{$sourcesid}{$positions} = $targetposs;
        }
    }

    return( \%mapping, \%tracing );
}

#############################################################
#
#   EditDistance: Operationskosten berechnen
#
sub EditDistance {

    my $self    = $_[0];
    my $stringX = $_[1];
    my $stringY = $_[2];

    my @segmentsX = @$stringX;
    my @segmentsY = @$stringY;

    my $sumX = scalar( @segmentsX );
    my $sumY = scalar( @segmentsY );

    my @matrix = ();
    $matrix[0][0] = 0;
    foreach my $elemX( 1 .. $sumX ) {
        $matrix[$elemX][0] = $elemX * 1;
    }
    foreach my $elemY( 1 .. $sumY ) {
        $matrix[0][$elemY] = $elemY * 1;
    }

    foreach my $accX( 1 .. $sumX ) {
        foreach my $accY( 1 .. $sumY ) {
            my $costDEL = $matrix[$accX-1][$accY] + 1;
            my $costINS = $matrix[$accX][$accY-1] + 1;
            my $costSUB = 0;
            if( $segmentsX[$accX-1] eq $segmentsY[$accY-1] ) {
                $costSUB = $matrix[$accX-1][$accY-1] + 0;
            }
            else {
                $costSUB = $matrix[$accX-1][$accY-1] + 2;
            }
            my $min = 0;
            $min = ( $costDEL < $costINS) ? $costDEL : $costINS;
            $min = ( $min < $costSUB ) ? $min : $costSUB;
            $matrix[$accX][$accY] = $min;
        }
    }

    my @targetposs = ();
    my @sourceposs = ();
    while( $sumX > 0 && $sumY > 0 ) {
        if( $matrix[$sumX][$sumY] == $matrix[$sumX-1][$sumY] +1 ) {
            $sumX = $sumX - 1;
        }
        elsif( $matrix[$sumX][$sumY] == $matrix[$sumX][$sumY-1] +1 ) {
```

```
            $sumY = $sumY - 1;
        }
        else {
            if( $segmentsX[$sumX-1] eq $segmentsY[$sumY-1] ) {
                unshift( @targetposs, $sumX );
                unshift( @sourceposs, $sumY );
            }
            $sumX = $sumX - 1;
            $sumY = $sumY - 1;
        }
    }

    return( \@targetposs, \@sourceposs );
}

############################################################
#
#   extractHypothesesTarget: durch Patterns die Hypothesen extrahieren
#
sub extractHypothesesTarget {

    my $self     = $_[0];
    my $targetid = $_[1];
    my $mapping  = $_[2];

    my @sententity = @{$self->{'sentences'}->{$targetid}};
    my $tarslength = $self->{'senlength'}->{$targetid};

    my %hypotheses = ();     # Slots
    my %hypoids    = ();
    my %sortweight = ();     # Slot mit jeweiligem Gewicht
    my %sourcecons = ();     # Konstituenten im Source
    my $counter    = 1;
    while( my( $sourceid, $oneblock ) = each( %$mapping ) ) {
        my $souslength = $self->{'senlength'}->{$sourceid};
        while( my( $positions, $onepattern ) = each( %$oneblock ) ) {
            my @parnodestar = @$onepattern;
            if( $parnodestar[0] > 1 ) {
                my $slote = $parnodestar[0] - 1;
                if( !exists $hypotheses{"1,$slote"} ) {
                    my $hypoid = "S" . $targetid . "C" . $counter++;
                    $hypoids{"1,$slote"} = $hypoid;
                    @{$hypotheses{"1,$slote"}} = (1 .. $slote);
                    if( !exists $sortweight{"1,$slote"} || $sort-
weight{"1,$slote"} > $self->{'randomids'}->{$sourceid} ) {
                        $sortweight{"1,$slote"} = $self->{'randomids'}-
>{$sourceid};
                    }
                }
            }
            if( $parnodestar[-1] < $tarslength ) {
                my $slotb = $parnodestar[-1] + 1;
                if( !exists $hypotheses{"$slotb,$tarslength"} ) {
                    my $hypoid = "S" . $targetid . "C" . $counter++;
                    $hypoids{"$slotb,$tarslength"} = $hypoid;
                    @{$hypotheses{"$slotb,$tarslength"}} = ($slotb ..
$tarslength);
```

```
                    if( !exists $sortweight{"$slotb,$tarslength"} ||
$sortweight{"$slotb,$tarslength"} > $self->{'randomids'}->{$sourceid} )
{
                        $sortweight{"$slotb,$tarslength"} = $self-
>{'randomids'}->{$sourceid};
                    }
                }
            }
            for( my $num=0; $num<$#parnodestar; $num++ ) {
                my $slotbegin = $parnodestar[$num] + 1;
                my $slotend   = $parnodestar[$num+1] - 1;
                if( $slotbegin <= $slotend && !exists $hypothes-
es{"$slotbegin,$slotend"} ) {
                    my $hypoid = "S" . $targetid . "C" . $counter++;
                    $hypoids{"$slotbegin,$slotend"}       = $hypoid;
                    @{$hypotheses{"$slotbegin,$slotend"}} = ($slotbe-
gin .. $slotend);
                    if( !exists $sortweight{"$slotbegin,$slotend"} ||
$sortweight{"$slotbegin,$slotend"} > $self->{'randomids'}-
>{$sourceid} ) {
                        $sortweight{"$slotbegin,$slotend"} = $self-
>{'randomids'}->{$sourceid};
                    }
                }
            }
        }
    }

    return( \%hypotheses, \%hypoids, \%sortweight, \%sourcecons );
}

############################################################
#
#   constructFuzzyTreebank
#
sub constructFuzzyTreebank {

    my $self       = $_[0];
    my $targetid   = $_[1];
    my $hypotheses = $_[2];
    my $hypoids    = $_[3];

    my @sententity = @{$self->{'sentences'}->{$targetid}};
    my $tarslength = $self->{'senlength'}->{$targetid};
    my $filename = "$self->{'config'}->{'dirResults'}/$self-
>{'config'}->{'Source'}"
                 . "_" . "$self->{'config'}->{'Docnum'}"
                 . "_" . "$self->{'config'}->{'Methode'}"
                 . "_fuzzytreebankT.txt";
    open( my $RES, '>>:utf8', "$filename" ) || die "$!";
    print $RES "%% sent. no. $targetid\n";
    foreach my $onehypo( sort{$a cmp $b} keys %$hypotheses ) {
        my @onecons = @{$hypotheses->{$onehypo}};
        foreach my $num( 0 .. $#onecons ) {
            $onecons[$num] = $sententity[$onecons[$num]-1];
        }
        print $RES "$hypoids->{$onehypo}\t" . join( ' ', @onecons ) .
```

```
"\n";
    }
    print $RES "\n\n";
    close( $RES );

    return 1;
}

#############################################################
#
#   SelektionLernen: Die Hypothesen verfeinern
#
sub SelektionLernen {

    my $self       = $_[0];
    my $targetid   = $_[1];
    my $hypotheses = $_[2];
    my $hypoids    = $_[3];
    my $sortweight = $_[4];
    my $sourcecons = $_[5];

    # Die gefundenen Hypothesen im Target-Satz verfeinern
    my $constituents = $self->selectConstituents( $hypotheses, $sort-
weight );

    # Baumstruktur aufbauen
    my( $tree, $nodes, $hypos ) = $self->constructTreeT( $targetid,
$constituents, $hypoids );

    #
    $self->constructTreebank( $targetid, $tree, $nodes, $hypos, $sour-
cecons );

    return 1;
}

#############################################################
#
#   selectConstituents: Konstitutenten aus den Hypothesen selektieren
#
sub selectConstituents {

    my $self       = $_[0];
    my $hypotheses = $_[1];
    my $sortweight = $_[2];

    my %constituents = ();
    my $sumpat = 0;
    foreach my $oneslot( sort{$$sortweight{$a} <=> $$sortweight{$b}}
keys %$sortweight ) {
        my( $slotbegin, $slotend ) = ( $oneslot =~m/^(\d+),(\d+)$/ );
        if( $slotbegin == $slotend ) {
            $constituents{$oneslot} = 1;
            next;
        }
        if( !exists $constituents{$oneslot} ) {
            my $checker = 0;
```

```
            if( $sumpat == 0 ) {
                $checker = 1;
            }
            else {
                for my $oslot( keys %constituents ) {
                    my( $onepatbegin, $onepatend ) = ( $oslot
=~m/^(\d+),(\d+)$/ );
                    if( ($slotbegin <= $onepatbegin && $slotend >=
$onepatend) ||
                        ($slotbegin >= $onepatbegin && $slotend <=
$onepatend) ||
                        ($slotbegin > $onepatend) || ($slotend < $one-
patbegin) ) {
                        $checker = 1;
                    }
                    else {
                        $checker = 0;
                        last;
                    }
                }
            }
            if( $checker == 1 ) {
                $constituents{$oneslot} = $slotend - $slotbegin + 1;
                $sumpat = 1;
            }
        }
    }

    return( \%constituents );
}

############################################################
#
#   constructTreeT: Baumstruktur aufbauen
#
sub constructTreeT {

    my $self     = $_[0];
    my $targetid = $_[1];
    my $constits = $_[2];
    my $hypoids  = $_[3];

    my @targetentity = @{$self->{'sentences'}->{$targetid}};
    my $targetlength = $self->{'senlength'}->{$targetid};

    my %nodes = ();
    my %hypos = ();
    my @targs = @targetentity;
    foreach my $onehypo( sort{$$constits{$a} <=> $$constits{$b}}
keys %$constits ) {
        my( $slotbegin, $slotend ) = ( $onehypo =~m/^(\d+),(\d+)$/ );
        my $hypoid = $hypoids->{$onehypo};
        my @poss = ($slotbegin .. $slotend);
        foreach my $pos( @poss ) {
            $nodes{$hypoid} .= "$targetentity[$pos-1] ";
        }
        $hypos{$hypoid} = $onehypo;
```

```
        $targs[$slotbegin-1] = "(" . $hypoid . " " . $targs[$slotbegin-
1];
        $targs[$slotend-1]   = $targs[$slotend-1] . ")";
    }
    my $tree = '';
    $tree = join( " ", @targs );
    my @tree = split( //, $tree );
    my $count = '';
    my $level = 0;
    foreach my $num( 0 .. $#tree ) {
        if( $tree[$num] eq '(' ) {
            $count = '(';
            $level++;
            $tree[$num] = "\n" . "\t" x $level . $tree[$num];
        }
        elsif( $tree[$num] eq ')' ) {

            if( $count eq ')' ) {
                $tree[$num] = "\n" . "\t" x $level . $tree[$num] .
"\n" . "\t" x $level;
            }
            else {
                $tree[$num] = $tree[$num] . "\n" . "\t" x $level;
            }
            $count = ')';
            $level--;
        }
    }
    $tree = join( '', @tree );
    $tree =~ s/\n+\s*\n+/\n/gs;
    $tree =~ s/\n+\s+$/\n/gs;

    return( $tree, \%nodes, \%hypos );
}

#############################################################
#
#   constructTreebank
#
sub constructTreebank {
    my $self       = $_[0];
    my $targetid   = $_[1];
    my $tree       = $_[2];
    my $nodes      = $_[3];
    my $hypos      = $_[4];
    my $sourcecons = $_[5];

    $self->treebankT( $targetid, $tree, $nodes );

    return 1;
}

#############################################################
#
#   treebankT
#
```

```
sub treebankT {

    my $self     = $_[0];
    my $targetid = $_[1];
    my $tree     = $_[2];
    my $nodes    = $_[3];

    my $filename = "$self->{'config'}->{'dirResults'}/$self-
>{'config'}->{'Source'}"
                 . "_" . "$self->{'config'}->{'Docnum'}"
                 . "_" . "$self->{'config'}->{'Methode'}"
                 . "_treebankT.txt";
    open( my $RES, '>>:utf8', "$filename" ) || die "$!";
    print $RES "%% sent. no. $targetid\n";
    print $RES "S($tree)\n\n";
    foreach my $conid( sort{$a cmp $b} keys %$nodes ) {
        print $RES "$conid\t$$nodes{$conid}\n";
    }
    print $RES "\n\n";
    close( $RES );

    return 1;
}

1;
```

```
############################################################
#
#   treeEvaluation.pm
#
#   Resultate evaluieren
#
############################################################
package treeEvaluation;
use strict;
use warnings;
require Exporter;
my @ISA = qw( Exporter );
my @EXPORT = qw( evaluator );


############################################################
#
#   Contructor initieraliseren
#
sub new {

    my $class   = $_[0];
    my $config  = $_[1];

    my $self = {};
    $self->{'config'} = $config;
```

```
    bless( $self, $class );
}

############################################################
#
#   evaluator
#
sub evaluator {

    my $self = $_[0];

    my $goldcons = $self->parserTreebank();
    my( $sumtree, $newpats ) = $self->sumTreebank( $goldcons );
    $self->evaluateResults( $newpats, $sumtree );
}

############################################################
#
#   parserTreebank: Treebank einlesen
#
sub parserTreebank {

    my $self   = $_[0];
    my $docnum = $_[1];
    my $corpus = $_[2];

    my $file = "EN_tueba_Treebank.txt";
    $file    = "DE_tueba_Treebank.txt" if( $self->{'config'}-
>{'Source'} eq "DE_tueba" );
    my %goldcons  = ();
    my %nodevalue = ();
    my %nodetag   = ();
    my %relations = ();
    my %levelnode = ();
    my $counter   = 0;
    my $level     = 0;
    my $id        = '';
    my $doccount  = 0;
    open( my $TREE, '<:utf8', "$self->{'config'}-
>{'dirCorpora'}/$file" ) || die "$!";
    while( <$TREE> ) {
        if( m/\%\%\s+sent\.\s+no\.\s+(\d+)/ ) {
            $doccount++;
            last if( $doccount > $self->{'config'}->{'Docnum'} );

            $id = $1;
            %nodevalue = ();
            %nodetag   = ();
            %relations = ();
            %levelnode = ();
            $counter   = 0;
            $level     = 0;
            $relations{0}{1} = 0;
            $levelnode{0}    = 0;
        }
        elsif( m/.*?\S+.*?/ ) {
```

```
            if( m/^\s*\((\S+)\s+(\S+)\)\s*$/ ) {
                $counter++;
                $level++;
                $nodevalue{$counter} = lc( $2 );
                $nodetag{$counter}   = $1;
                $levelnode{$level}   = $counter;
                $relations{$levelnode{$level-1}}{$counter} = 0;
                $level--;
            }
            elsif( m/^\s*\((\S*)\s*$/ ) {
                $counter++;
                $level++;
                $nodevalue{$counter} = '';
                $nodetag{$counter}   = $1;
                $levelnode{$level}   = $counter;
                $relations{$levelnode{$level-1}}{$counter} = 0;
            }
            elsif( m/^\s*\S*\)\s*$/ ) {
                $level--;
            }
            else {
                print "ERROR: $_\n";
            }
        }
        else {
            while( $counter > 0 ) {
                $counter--;
                my $node = $counter + 1;
                next if( !exists $relations{$node} );
                for my $subnode( sort{$a <=> $b} keys %{$rela-
tions{$node}} ) {
                    $nodevalue{$node} .= ' ' if( $nodevalue{$node} ne
'' );
                    $nodevalue{$node} .= $nodevalue{$subnode};
                    $goldcons{$id}{$nodevalue{$subnode}}++;
                }
                $goldcons{$id}{$nodevalue{$node}}++;
            }
        }
    }
    close( $TREE );

    return( \%goldcons );
}

#############################################################
#
#   sumTreebank: Anzahl der Patterns in Treebank
#
sub sumTreebank {

    my $self     = $_[0];
    my $goldcons = $_[1];

    my $count = 0;
    my %newpatterns = ();
    while( my( $sid, $onetree ) = each %$goldcons ) {
```

```
        while( my( $goldcon, $freq ) = each %$onetree ) {
            my @newpat = split( / /, $goldcon );
            my $sumpat = scalar @newpat;
            $count++;
            $newpatterns{$sid}{$goldcon}++;;
        }
    }

    return( $count, \%newpatterns );
}

############################################################
#
#   evaluateResults: Anzahl der Patterns der Resultate
#
sub evaluateResults {

    my $self     = $_[0];
    my $treebank = $_[1];
    my $sumtree  = $_[2];

    my $countres = 0;
    my $counttru = 0;
    my $id   = '';

    my $filename = "_treebankT.txt";
    $filename = "_fuzzytreebankT.txt" if( $self->{'config'}-
>{'Systype'} eq 'noselection' );
    my $fileres = "$self->{'config'}->{'dirResults'}/$self->{'config'}-
>{'Source'}"
                  . "_" . "$self->{'config'}->{'Docnum'}"
                  . "_" . "$self->{'config'}->{'Methode'}"
                  . "$filename";
    open( my $RES, '<:utf8', "$fileres" ) || die "$!";
    while( <$RES> ) {
        chomp;
        my $cons = '';
        if( /^\%\% sent\. no\. (\d+)\s*$/ ) {
            $id = $1;
        }
        elsif( /^S\d+C\d+\s+(.*?)\s*$/ ) {
            $cons = $1;
            $countres++;
            if( exists $treebank->{$id}{$cons} ) {
                $counttru++;
            }
        }
    }
    close( $RES );

    my $countfal  = $countres - $counttru;
    my $recall    = $counttru / $sumtree;
    my $prezision = $counttru / $countres;
    open( my $LOG, '>>:utf8', "$self->{'config'}-
>{'dirResults'}/logs.txt" ) || die "$!";
    print $LOG "Standard Konstituenten:    $sumtree\n";
    print $LOG "Gelernte Hypothesen:       $countres\n";
```

```
    print $LOG "Rechtige Hypothesen:        $counttru\n";
    print $LOG "Falsche Hypothesen:         $countfal\n";
    print $LOG "Recall:                     $recall\n";
    print $LOG "Präzision:                  $prezision\n\n";
    close( $LOG );

    print "Standard Konstituenten:    $sumtree\n";
    print "Gelernte Hypothesen:       $countres\n";
    print "Rechtige Hypothesen:       $counttru\n";
    print "Falsche Hypothesen:        $countfal\n";
    print "Recall:                    $recall\n";
    print "Präzision:                 $prezision\n\n";

    return 1;
}

1;
```

```
#!/usr/bin/perl
#############################################################
#
#   main_edsi.pl
#   - EDSI-Ansatz mit Parameter aufrufen
#
#
#############################################################
use strict;
use warnings;
BEGIN {
        @INC = ( "Systems", "Evaluation", @INC );
}
use Config::config;
use Systems::edsiIndex;
use Systems::edsiLernen;
use Evaluation::treeEvaluation;

# Parameter
my $Modus      = "1";              # 1 oder 2
my $Strategy   = "1";              # 1 oder 2
my $Systype    = "selection";      # selection oder noselection (mit
oder ohne Selektion-Lernen
my $Diffleng   = "3";              #

my $methode    = "edsi";           # edsi oder abl
my $source     = "EN_tueba";       # tuebaes oder tuebads
my $docnumber  = 2000;             # 1 bis 2200 im Englischen oder 1
bis 5500 im Deutschen
my $corpusname = $source . '.txt';

# config
my $config = new config( $Modus, $Strategy, $Systype, $Diffleng, $me-
thode, $source, $docnumber );
my $configrf = $config->getConfig();

# den Ansatz starten
```

```
# InvertedFiles aufbauen
my $invertedfile = new edsiIndex( $configrf );
my( $data, $timetoindex ) = $invertedfile->getIndex( $corpusname );

# Treebank aufbauen
my $pattern = new edsiLernen( $data );
$pattern->processorSearch( $timetoindex );

# Evaluierung
my $treebank = new treeEvaluation( $configrf );
$treebank->evaluator();
```

```
#!/usr/bin/perl
############################################################
#
#   main_abl.pl
#   - ABL-Ansatz aufrufen
#
#
############################################################
use strict;
use warnings;
BEGIN {
        @INC = ( "Systems", "Evaluation", @INC );
}
use Config::config;
use Systems::ablLernen;
use Evaluation::treeEvaluation;

# Parameter
my $Modus      = "1";              # 1 oder 2
my $Strategy   = "1";              # 1 oder 2
my $Systype    = "selection";      # selection oder noselection (mit
oder ohne Selektion-Lernen
my $Diffleng   = "3";              #

my $source     = "EN_tueba";       # tuebaes oder tuebads
my $docnumber  = 2000;               # 1 bis 2200 im Englischen oder 1
bis 5500 im Deutschen
my $corpusname = $source . '.txt';

# config
my $config = new config( $Modus, $Strategy, $Systype, $Diffleng, 'abl',
$source, $docnumber );
my $configrf = $config->getConfig();

# den Ansatz starten
# Treebank aufbauen
my $pattern = new ablSearchPatterns( $configrf );
$pattern->processorSearch( $corpusname );

# Evaluation
my $treebank = new treeEvaluation( $configrf );
$treebank->evaluator();
```

7.2 Baumstruktur des Satzes 490 im englischen Korpus

7.2.1 Baumstruktur aus Treebank T

```
%% sent. no. 490
S(
        (S490C14 we)
        (S490C12
                (S490C23 are)
                (S490C15
                        (S490C9 going)
                        (S490C22
                                (S490C28 to)
                                (S490C4
                                        (S490C20 get)
                                        (S490C7
                                                (S490C26 there)
                                                (S490C5 on
                                                        (S490C2 tuesday)
                                                )
                                        )
                                )
                        )
                )
        )
)

S490C2      tuesday
S490C4      get there on tuesday
S490C5      on tuesday
S490C7      there on tuesday
S490C9      going
S490C12     are going to get there on tuesday
S490C14     we
S490C15     going to get there on tuesday
S490C20     get
S490C22     to get there on tuesday
S490C23     are
S490C26     there
S490C28     to
```

7.2.2 Baumstruktur aus Treebank S

490: we are going to get there on tuesday

on tuesday

at ten thirty six
very late at night
<E>
, coming up very soon
late
this time
at one o'clock
and see
at nine o'clock
from like nine to twelve
one at twenty thirty
was a six o'clock express
in hannover
to the office then
too
monday morning
at midnight
is any changes or anything
is some play
for that
for four days
again for
flights
three days
the next morning as well
ten at night sunday
to change
then
at the station at dammtor
by midday
is one at eight o'clock
right in the morning
or
at ten o'clock their time
i thought
two days
at three in the afternoon
is any musical playing locally
was an eight o'clock flight
to go over things

	at noon at four the next morning three days so pretty early for a couple days if we wanted to
going	lying supposed close free getting paid trying running when we get willing starting wanting you ready ready
we	<E> so you so when you think you all three of them you said you is that what you my in-laws that means if you at least they how is no those days you and i i my parents so the is on what day when it seems like afternoons what time so here so what

any times that
how
i think maybe you
what afternoons
they
what about
there
maybe the
looks like they

tuesday

my schedule
<E>
the nineteenth
the fifteenth
your calendar
our time
vacation then
what time i go
leaving hanover around four
the fourth
the sunday before
getting back
the twenty ninth
the seventeenth
thursday night the fourth
a four day weekend
those dates
the seventh
the nineteenth actually
monday morning
the express flight
the twenty first
vacation that week
the third , right
the eleventh
time
our way
monday night again
an international flight
our trip
the twenty fourth
on thursday
thursday the second
it
vacation this week
saturday early morning

november fourth
the second of february
vacation
saturday
the twenty eighth
those days
the expense account
monday then
what day
the eighteenth
thursday anyway
friday
me
that day
thursday
sunday then
the plane
the same day
monday the fifteenth
the twenty second
sunday evening
february the third
the wrong week
the plane all morning
the first try
thursday june third
the tenth then
my other calendar
the eighth
the sixteenth
friday with british-airways
hotels in hanover
that wednesday
one of these days
saturdays
the sixth
your expense account
that
monday
the fifteenth of october
a monday
the twenty seventh
both monday and friday
wednesday we will leave
the twenty fifth
the safe side
wednesday
this end

	the tenth
	the twenty sixth
	the sixteenth of november
	the thirty first
	thursday the first
	the weekend
	the day
	friday the twenty first
	thursday at eleven
	the new lab
	the ninth
	then
	a sunday
	thursday , your office
	the ninth then
	saturday march fifth
	friday the ninth
	the ninth of march
	this
	the thirteenth
	the twenty seven
	this business
	monday afternoon
	monday the eighth
	friday the twelfth
	a saturday maybe
	sunday
	those two days
	that thursday and friday
	friday the eleventh
	the twentieth
	the twelfth
	the train
	the first
	that afternoon
	fridays
	the third
	to berlin on wednesday
	that friday
	september fifteenth
we are	
	that is
	i think i am
	thursday i am
	actually i am
	plus it is

so i am i know i am because i am how long is this not worry about arnie is probably that is probably it is i am is everything man it is looks like i am not no it is only what time you do you have a because i am not that is not i was she is not of may , i am do you think that is yeah i am do you think it is
there back here an early start half a day back to the meeting
to
get there on tuesday make an appointment for dewitt latimer the opera with jay midnight our time you miss that , huh six via dortmund to hanover nine a possibility four , so what about friday leave that early call and make germany is coming up

leave in the evening
check that
come back as early as possible
eleven , how about that
do it
a play one of them
go with the cheapest one
take charge of the hotel
the fifth
go for that
spend in hanover
make in a few months
cancel my dentist appointment now
see the grand canyon
be one thirty though
leave in the morning again
meet here at the office then
plan another business trip
stay at that one
visit some of the bars
take us
be back here by the twenty eighth
a restaurant or bar or something
hannover
need a car
meet around one or so
sleep it off
relax
be interested in doing something like that
be a good time
go to the weekend again
four , does that sound good
go with that again then
talk about a trip , right
setup a meeting
talk about going back
make an appointment with you
take the hotel luisenhof
plan the trip to hanover
do that in the next three months
do in the evenings
fit three days
five o'clock
make an appointment for
pittsburgh
go the fastest way possible
the hotel prinzenhof
see you again

schedule another meeting
new year 's
it as well
bail me out of anything
talk about this business trip to hanover
reschedule my work meeting
leave at four o'clock in the morning
need to cancel
need a lot of time to sleep
need about two hours
a restaurant
sorry
have a meeting this monday
see what time you could meet
me
a restaurant if you would like
the center , fifteen minutes
the train station
stay at hotel luisenhof
the hotel
be tied up with that
midnight
be in the morning
chill out before the morning
the twenty second
meet for at least two hours
make arrangements for it now
go with that
go some time really soon
be back monday
four thirty , so that is bad
schedule a two hour appointment with you
five
make sure
do this
pick me up
go away again
go
be in hanover around noon
one if you want
book a plane ticket to hanover
the end of the month
four thirty at my office
ten then
do much sightseeing
schedule a trip to hanover
want to schedule something
have time to go swimming

an opera thursday
keep the eleventh free
go with the six o'clock one
me , see you then
change
three in your office is fine
the twenty fifth
go into
germany
catch back up here
plan our third trip this year
talk about the return leg
meet in the same place
hanover
leave
prepare a talk from one to five
start at six in the morning tomorrow
the tenth
look at the next week after that
recover
meet
, little bit
eleven
plan a business trip to hanover
be bigger than mine
take care of
do too
schedule our next business trip , ha
set up another meeting
go see plays
schedule our trip here
give you the address
work
make an appointment
want to visit our affiliates in hanover
continue this
make a meeting this afternoon
be out of town at a conference
go shopping with you
be back from your vacation by then
fit you in at nine o'clock
your teeth you might be sore afterwards
five that day
have a vehicle
see you too miss cramer
go during the week
look at the hours monday
consider the hotel cristal hannover

be weekdays or weekends or
take a ten o'clock flight
schedule another meeting in april again
september seventh
do , but
the office then
take this month
three
do today
meet with you again
sleep for the meeting then
a bar
do during that time
go with a double
leave right after the meeting
twelve thirty five
go to the pub and stuff
look at , early morning
leave at six in the morning
a bar or something
be enough time
go those days
be pretty busy then
be back by sunday
have to go to december
the bar or something in the evening
go to
the fifteenth
be another day and a half
play in germany or just do business
make this thing as early as possible
fly
meet you at one thirty
pay cash or with card
two
stay away from december
go to the weekend
have a list of hotels here
central station
do the second week of june
do before
know when you are available this time
go to hanover to visit a branch
the seventeenth
twenty fourth
be two in the morning
noon
plan it as we want

the eleventh
pay them by cash
be a little tricky now
stay at
plan our next business trip to hanover
be able to do that
take a business trip to hanover
the tenth you are free
the thirtieth
ten o'clock
do it within the next three months
hanover at nine
be in the city at a seminar
seeing you then also
meet again
plan the trip home
schedule a meeting with you sometime
hanover , ha
two in the afternoon
do this as soon as possible
twelve is what it will be
be a little tight
come back early or late
twelfth i am free
be during the day
be short trip
leave wednesday or thursday
avoid going to work
hawaii this weekend , huh
a bar afterwards
be about a hundred sixty six euros
make that an overnight flight
schedule the next meeting
cover everything we needed to today
have lunch after the seminar
meet in my office
eleven meeting
you too , bye bye
meet in the next two weeks
leave at ten thirty in the morning
be a day and a half
work instead of
worry about entertainment at all
the eighteenth
do it next week if possible
leave in the morning
be back until friday
be booked all day

drop me off
check out wednesday morning from the hotel
look at
make arrangements for that too
discuss our april meeting
work with
knock out this last trip
twelve
seeing you then
catch the flight back
four
lunch at highlander my treat
be running around all the time
be a good time then
meet at your office then
have time to shop
plan this trip to hanover
be within the next two weeks
put in half a day
a movie or see a play
use the swimming pool
discuss our business meeting to hanover
the more economical hotel luisenhof
talk further about this
me , i will see you then
have much time for recreational things
the central station
plan a trip to germany
eleven in the morning
please
take care of our business meeting
five in the evening
eight
miss that
pick up my filofax
have to meet again
donate blood at three o'clock
the sixth
have to meet in december
do after the meeting
look at september
take
six thirty will be fine
meeting with you
the cafeteria
travel
plan our trip to hanover
the twenty seventh

do again this time
make another trip to hanover
set up our trip to hanover
leave as early as possible
germany together
one
be back before then
say eleven to be sure
need to leave early
relax at the hotel
finish the whole thing up by wednesday
receiving the tickets in the mail
the thirty first
do
find
be probably very early in the morning
ten
the airport
be in dresden anyways
have this important of a meeting then
meet in your office
you later tonight then
come back
figure out this trip to hanover
do that
take that train
see a movie
fly as well
the time
nine o'clock in the morning
a play or maybe do some sightseeing
arrange a meeting here
leave at four or
stay
schedule these things
do here
talk to you again
eight in the evening
plan another trip to hanover
do that evening
meet here around eight
leave for a couple days
four thirty
seven thirty
be in charge for that
the twenty second free
go to that
relax after that business meeting

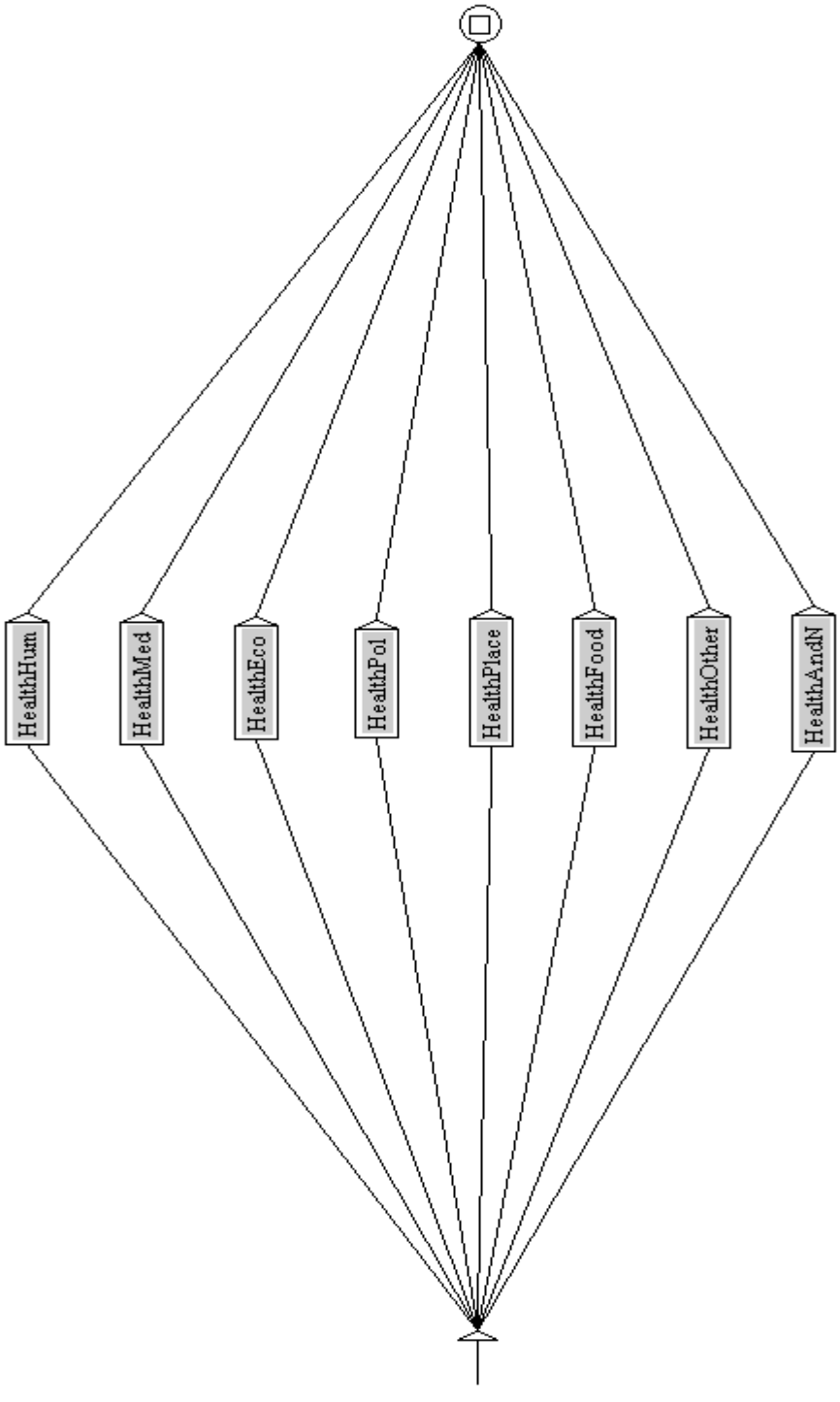
	the theater
	be fine for you
	plan our next trip to hanover
	talk about this a little bit more
	do this too many more times
	the station
	meet then
	handle that
	talk about
	come back thursday evening then
	me like we are looking in august
	your people
	plan a trip to hanover
	be back until february the second
	tell me
	move to the weekend
	me to stay at the maritim hotel
	be
	do it sometime after then
	the center
	be back before the eleventh
	go over things
	buy all new clothes
	schedule a trip to hannover here
	come back wednesday night
	be raunchy
	have to go to the weekend
	be in hanover
	hanover now
	five o'clock is good for me
	do anything else after that
	meet for
	one of those beer halls
	schedule another meeting with you
	set up another appointment
	have to change that
	the second
	have much time just to relax
	leave at nine
	make a meeting
	sleep
get	
	spend longer out
	meet
	spend some extra time
	spend

do while we are spend that extra time meet with our branch spend some time put us in see what is send us take a quick trip bring your family know what is stay arrive be
there on tuesday <E> around that done early it over with or whatever is closest to the center a few beers together for a couple of hours back from lunch in much later than that okay a nine o'clock flight us home in time when we arrive in hanover together again in a couple weeks together to talk further about this to the pittsburgh airport it done with that evening anymore this dealt with the better a double something close to the train station whatever is closest something a little bigger here at an earthly hour back then , monday evening to hanover this trip going to that hotel a single or a double a little tiresome year after year together for a meeting in around twelve thirty at night a little bit busier that week

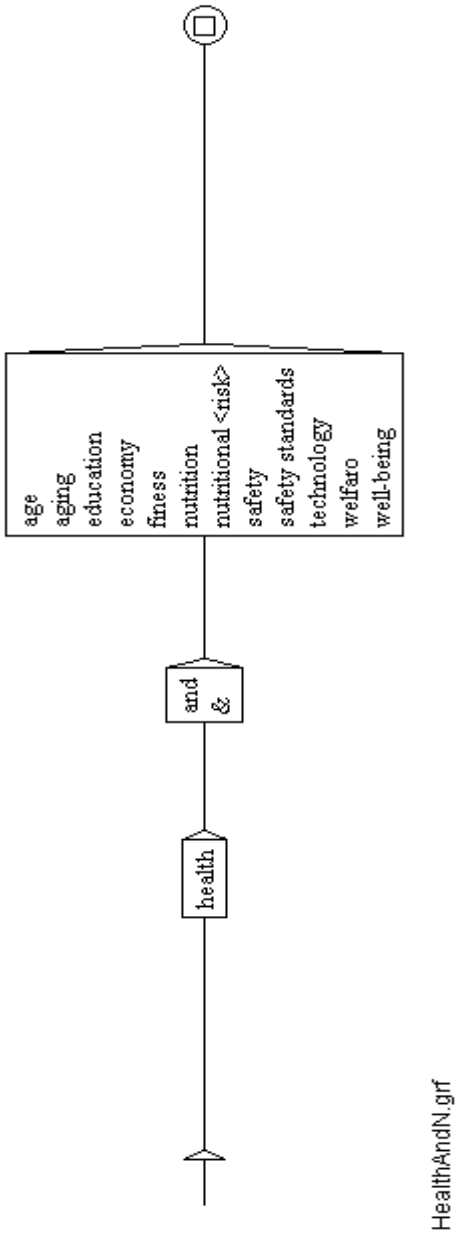
here early evening
two singles
up that early
that flight if possible
a single room
together again
to the airport
together
this trip organized then
back from the branch
about three days together to go
together for that meeting
to the airport and that
to the hotel
back
this done as soon as possible
home and
to the hotel from the airport
some doughnuts or something
out of here
back at midnight
together for another two hour meeting
an early start at nine
some german beer
up in the morning
doubles or singles

7.3 Lokale Grammatiken

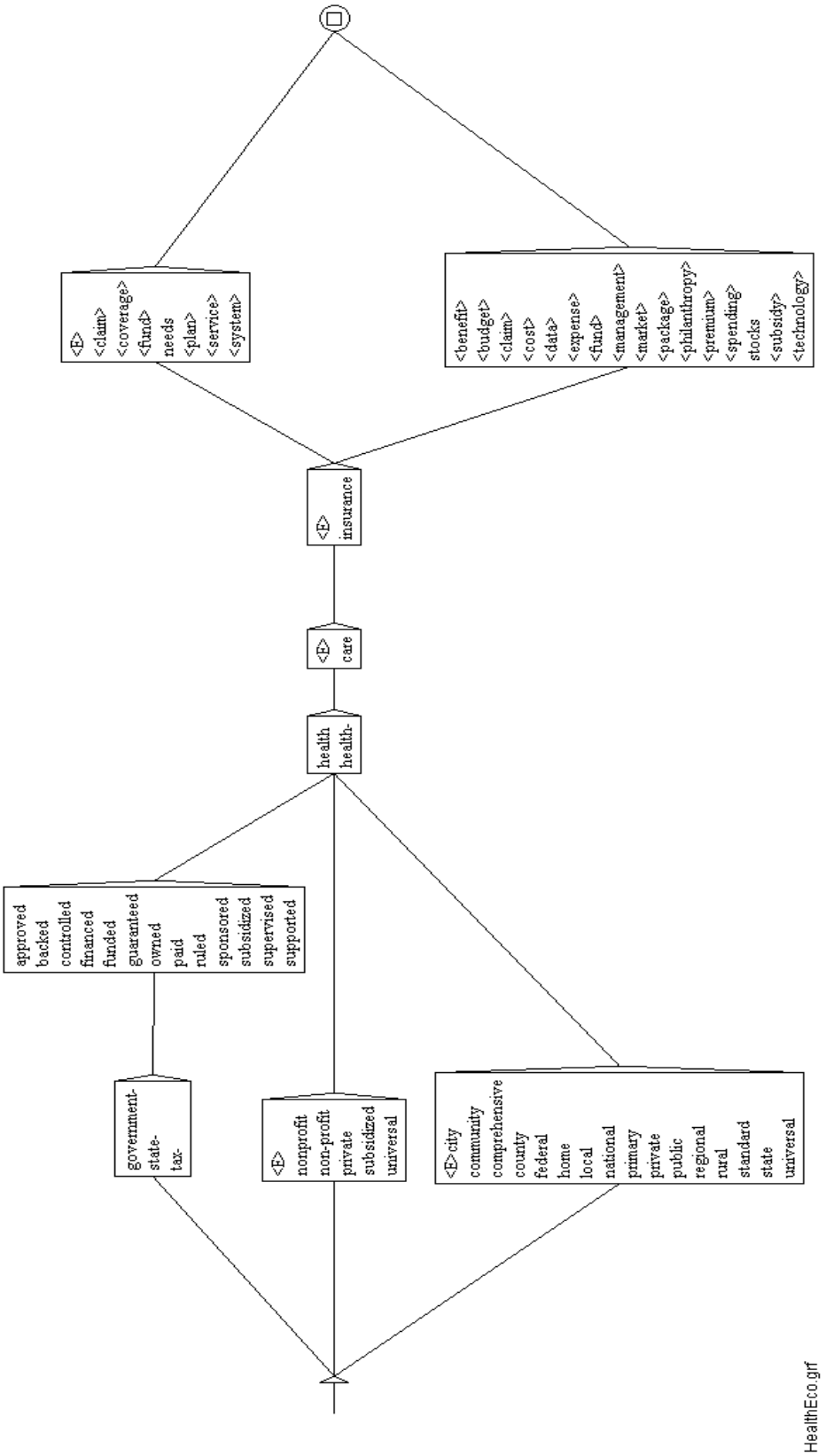
7.3.1 Beispiel „health“



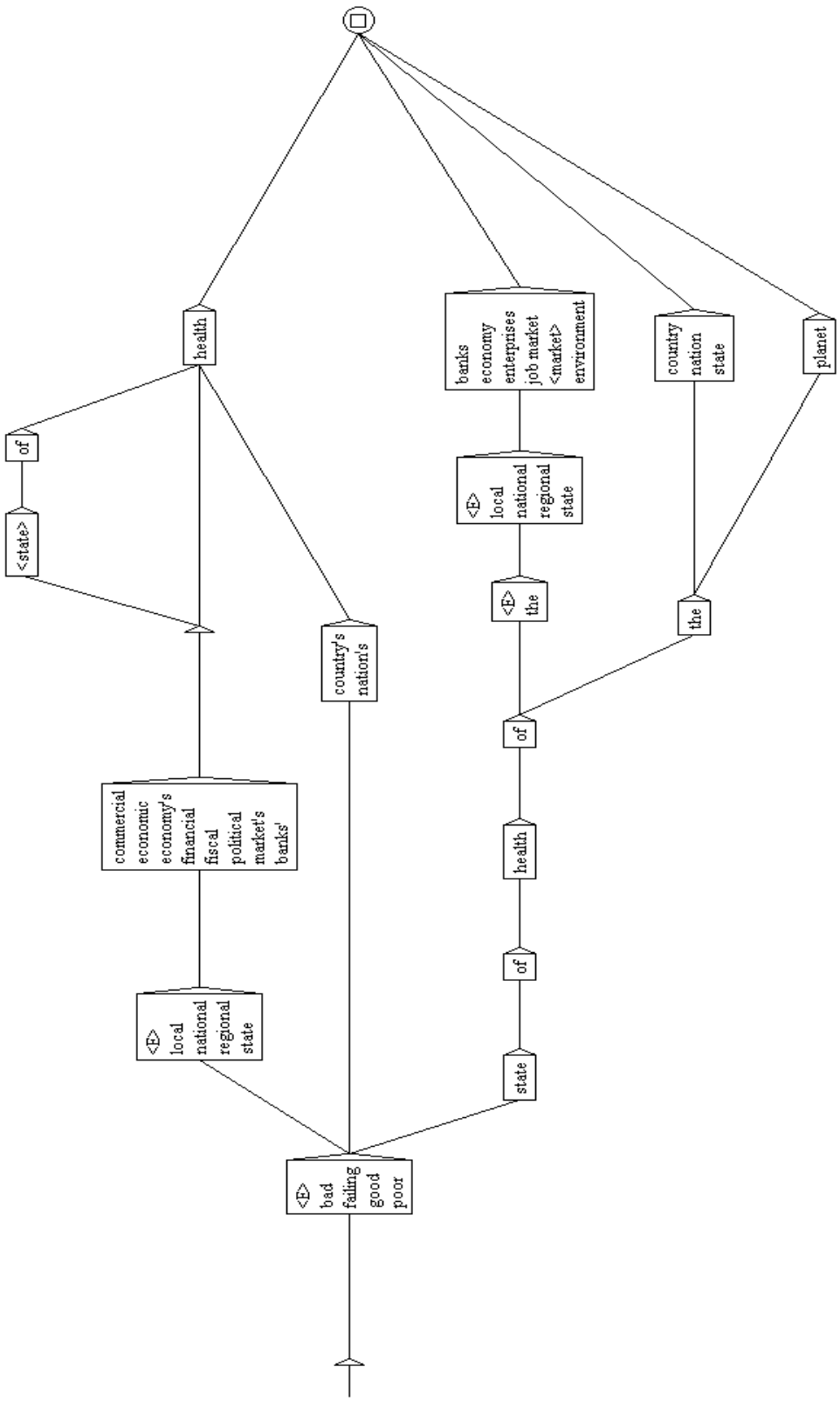
Subgragh: HealthAndN



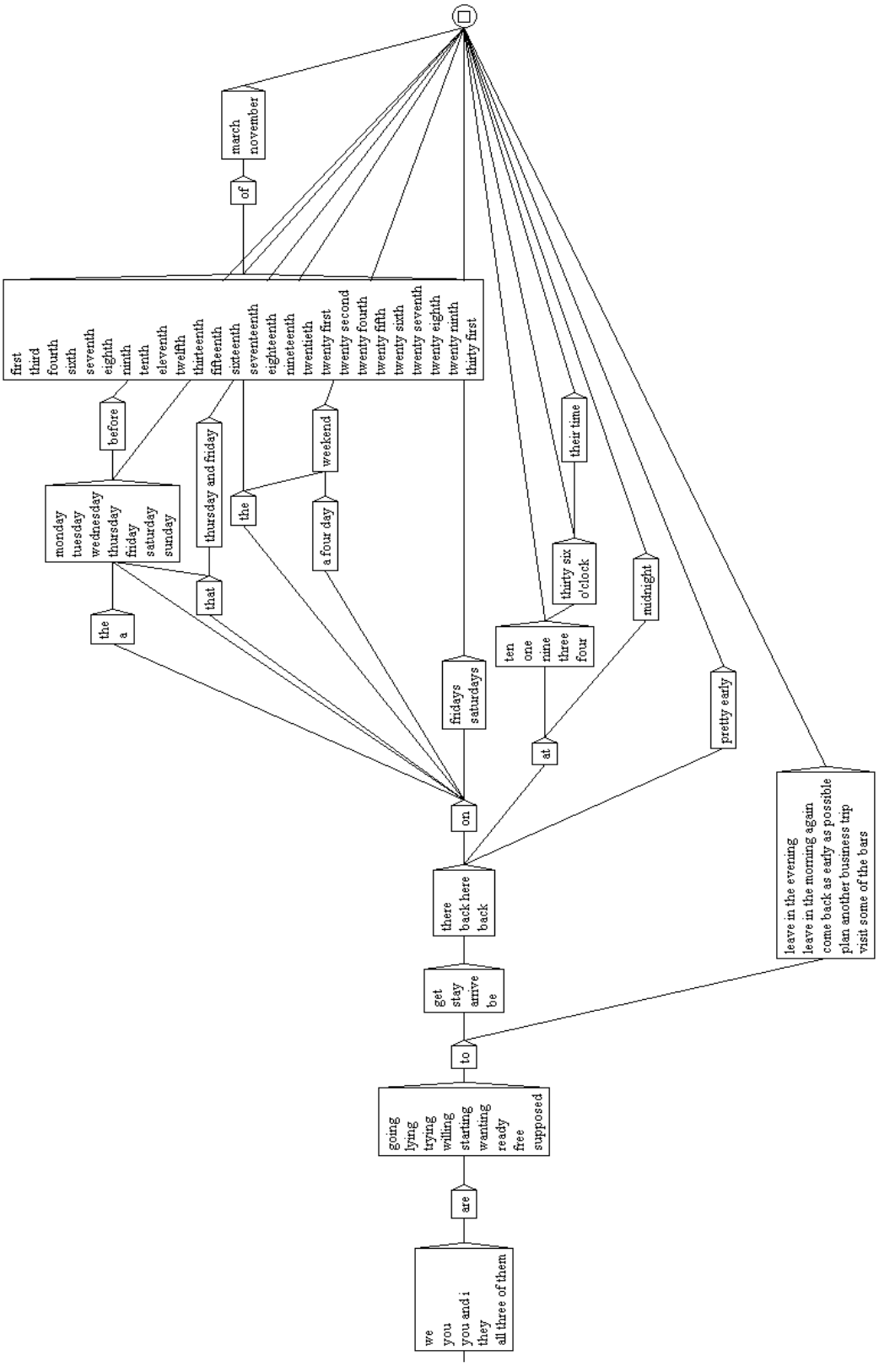
Subgragh: HealthEco



Subgragh: HealthOther

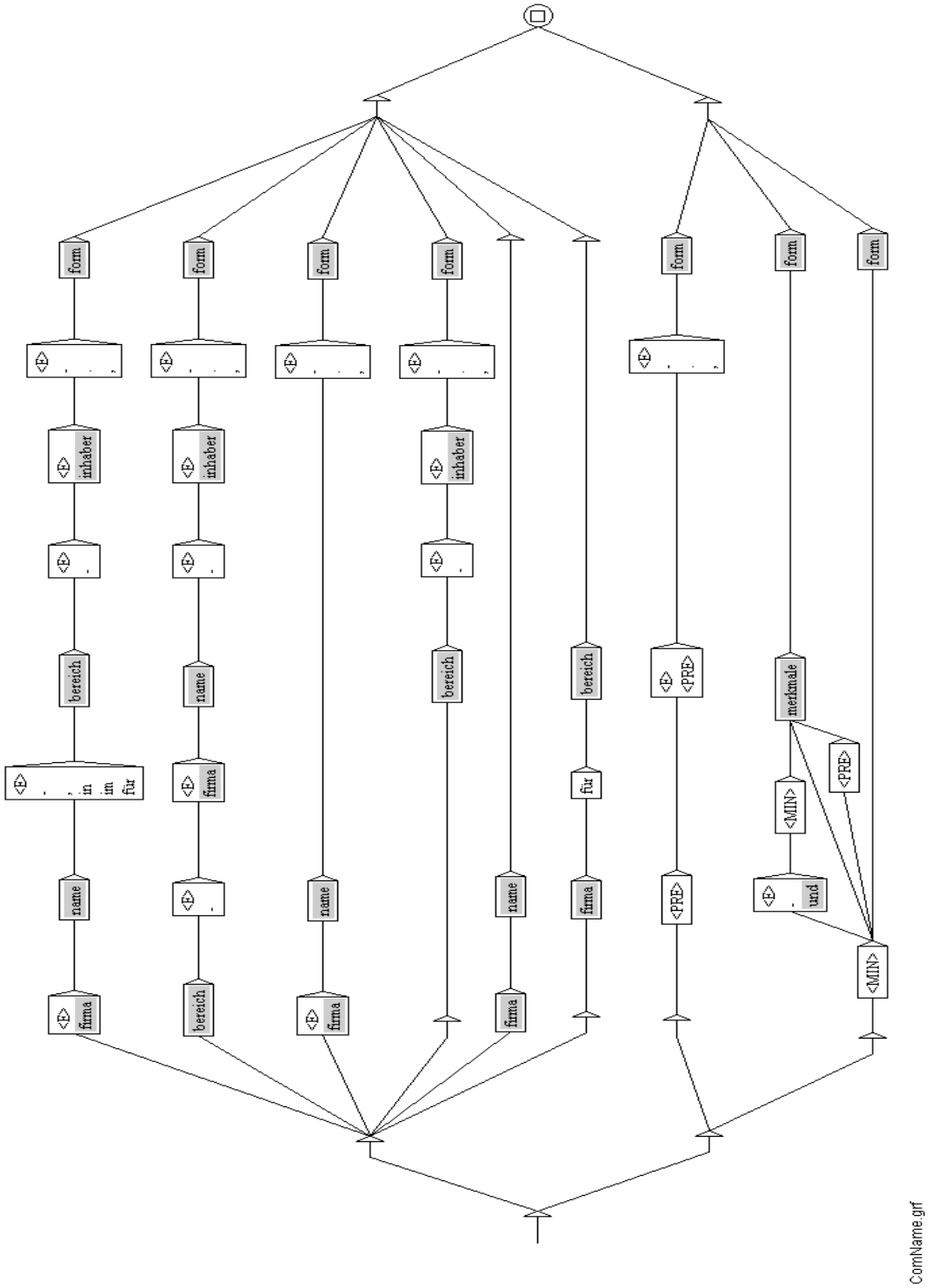


7.3.2 Lokale Grammatik des Satzes 490 im englischen Korpus

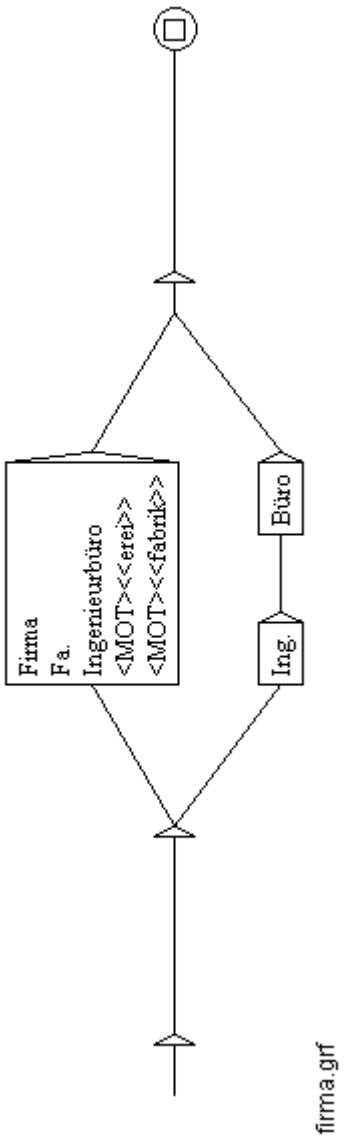


7.3.3 Extraktion der Firmeninformationen

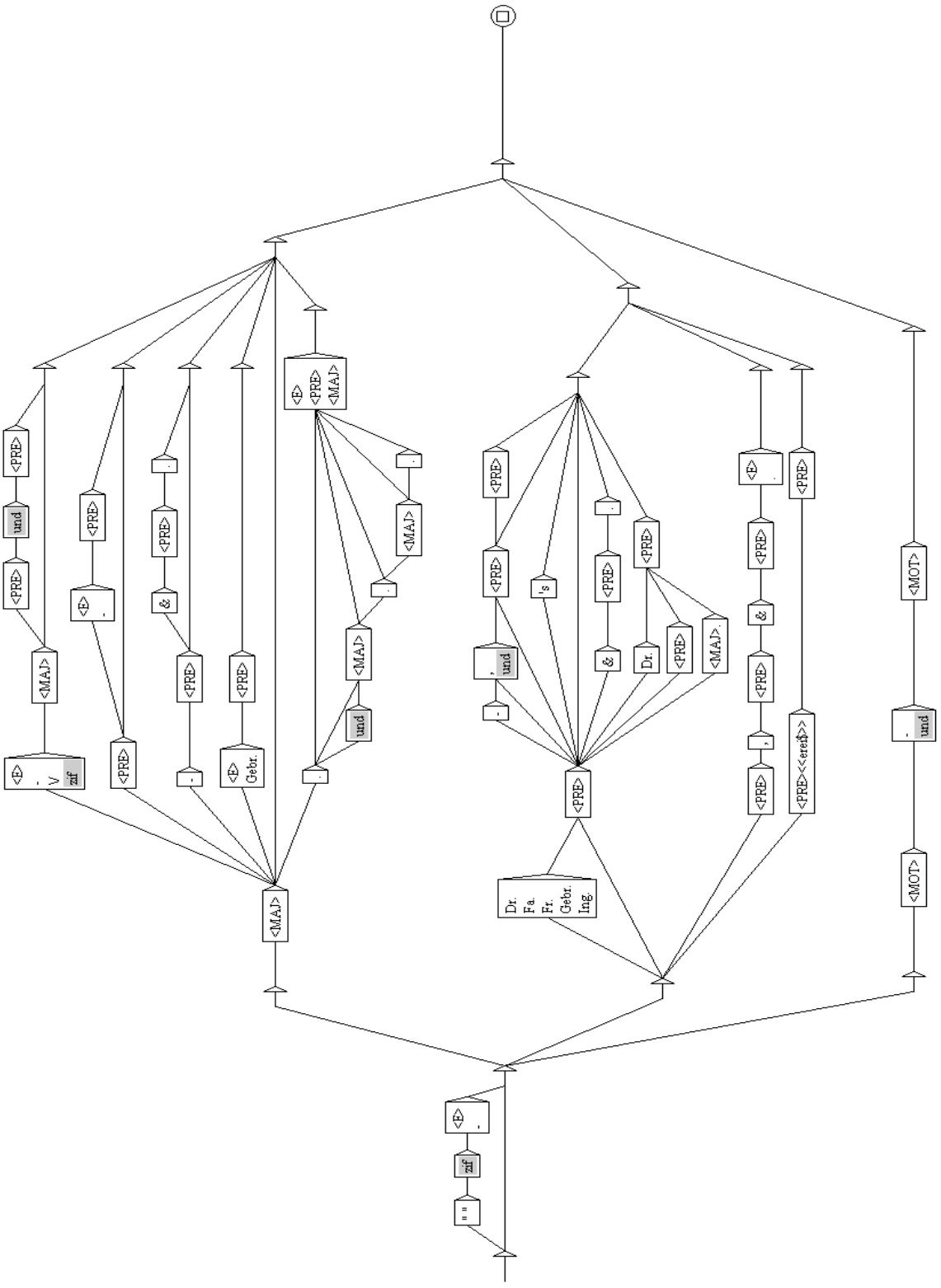
7.3.3.1 Lokale Grammatik der Firmennamen



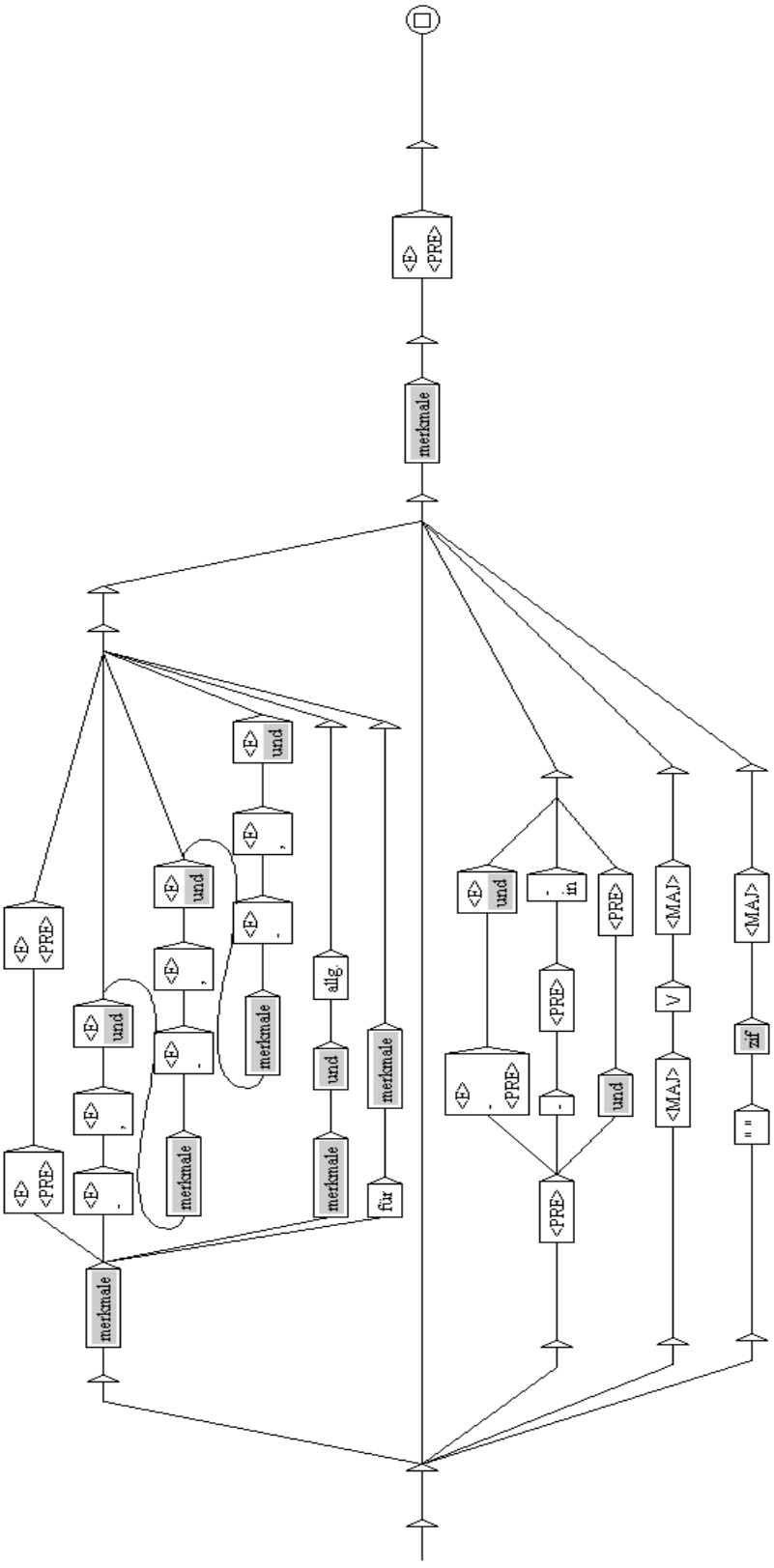
Subgragh: firma



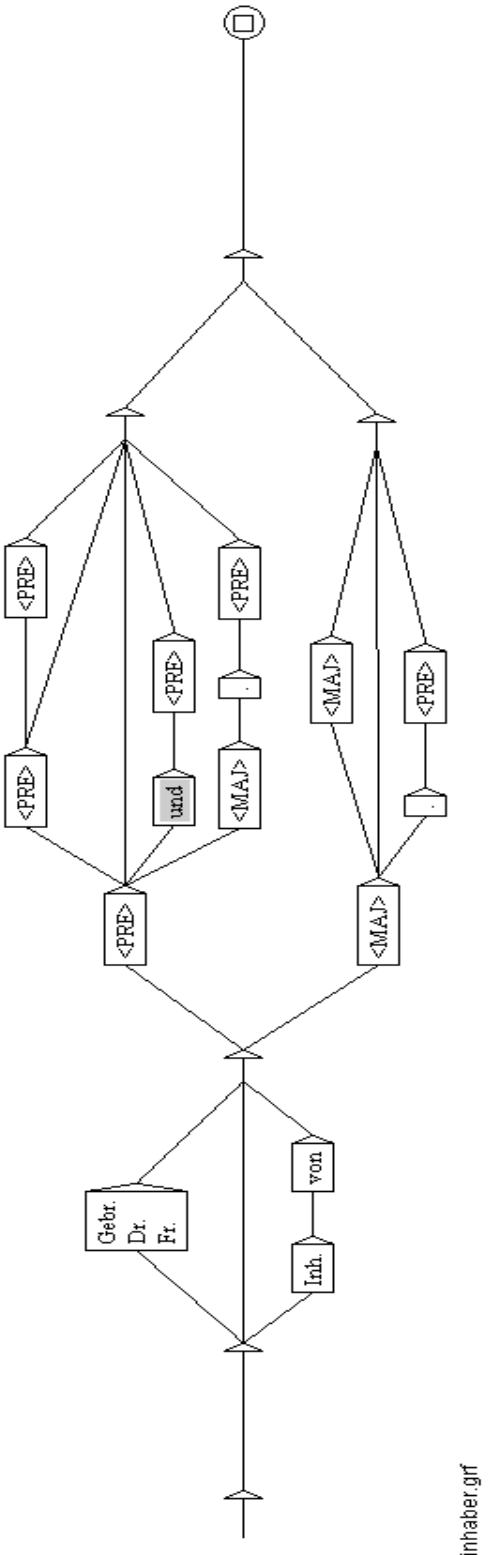
Subgragh: name



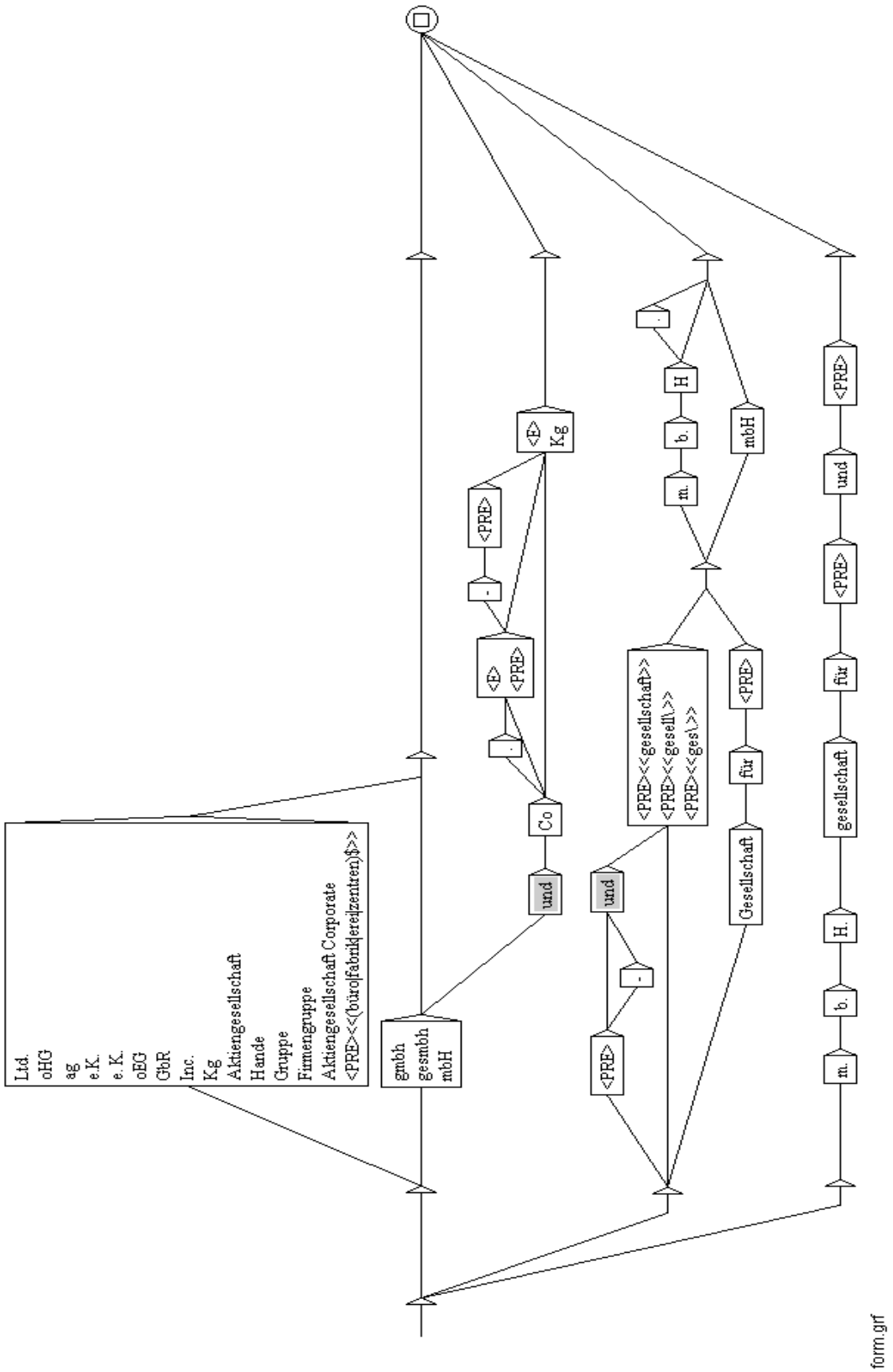
Subgragh: bereich



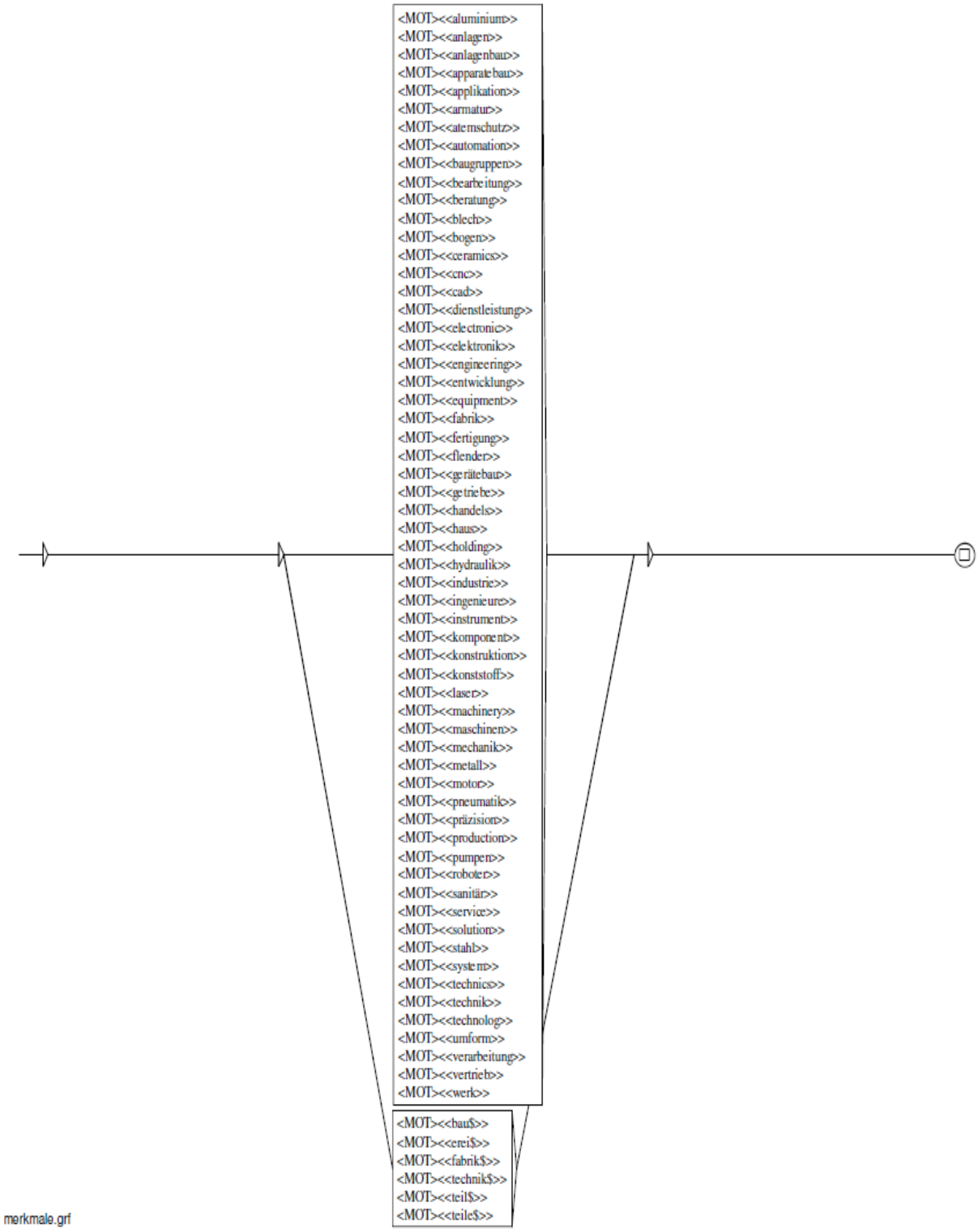
Subgragh: inhaber



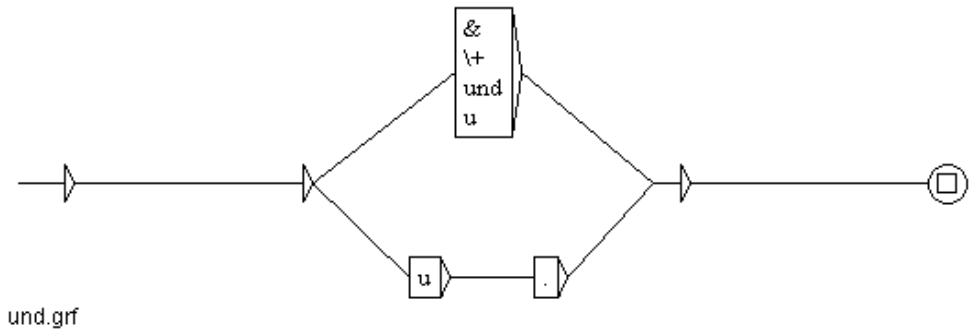
Subgragh: form



Subgragh: merkmale

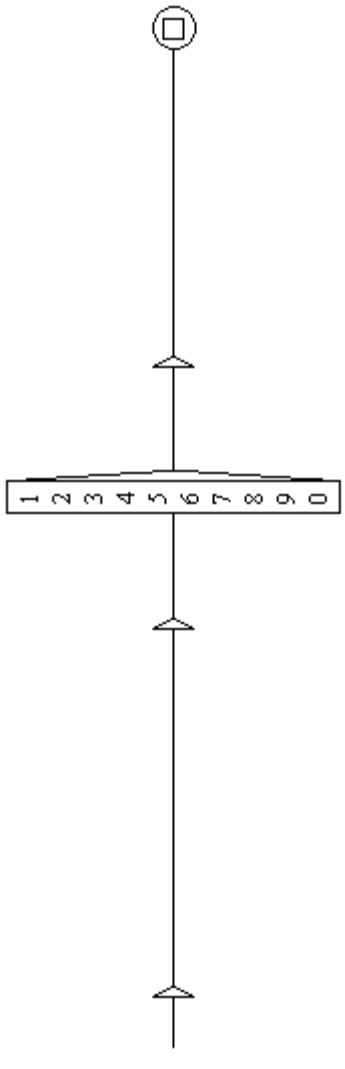


Subgragh: und



und.grf

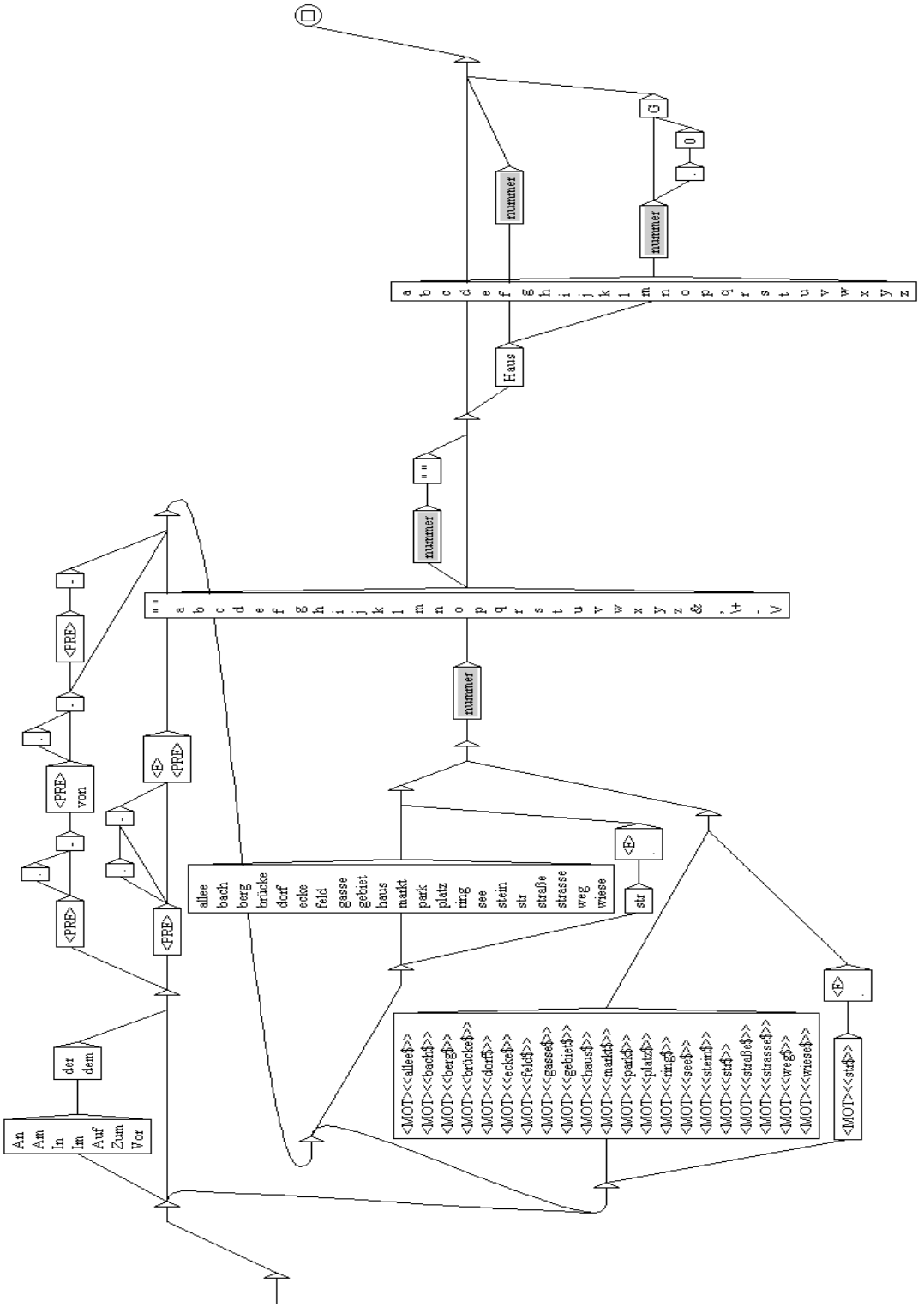
Subgragh: zif



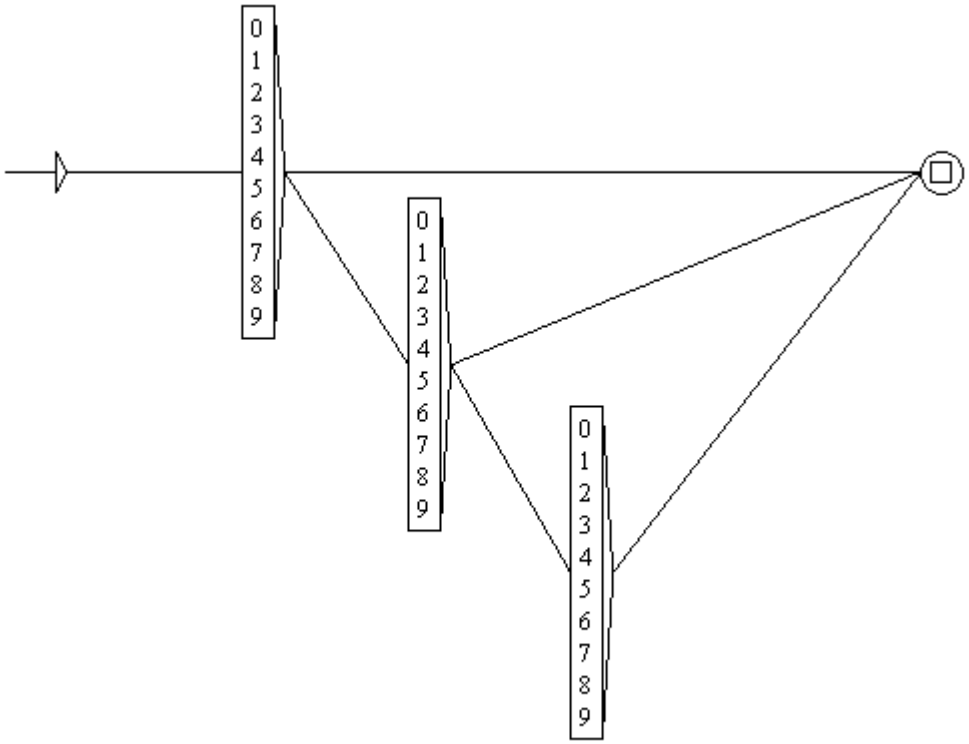
zif.grf

7.3.3.2 Lokale Grammatik der Straßennamen

ComAddress



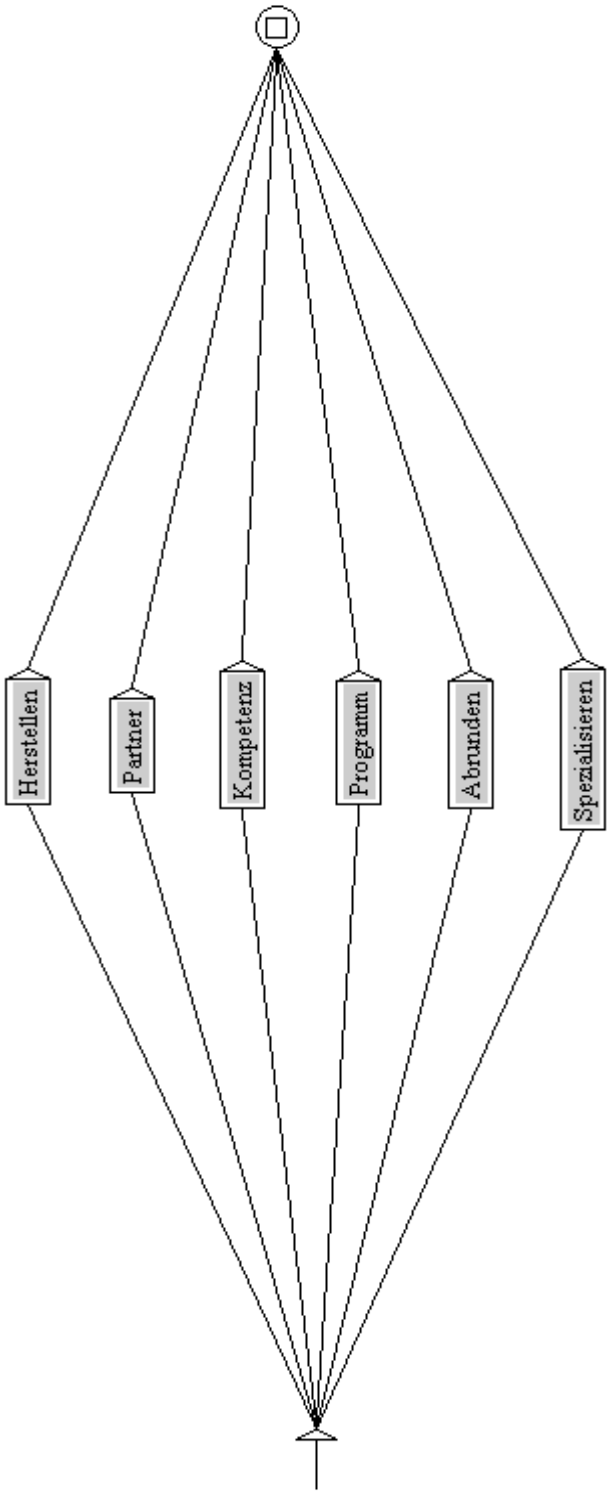
Subgragh: nummer



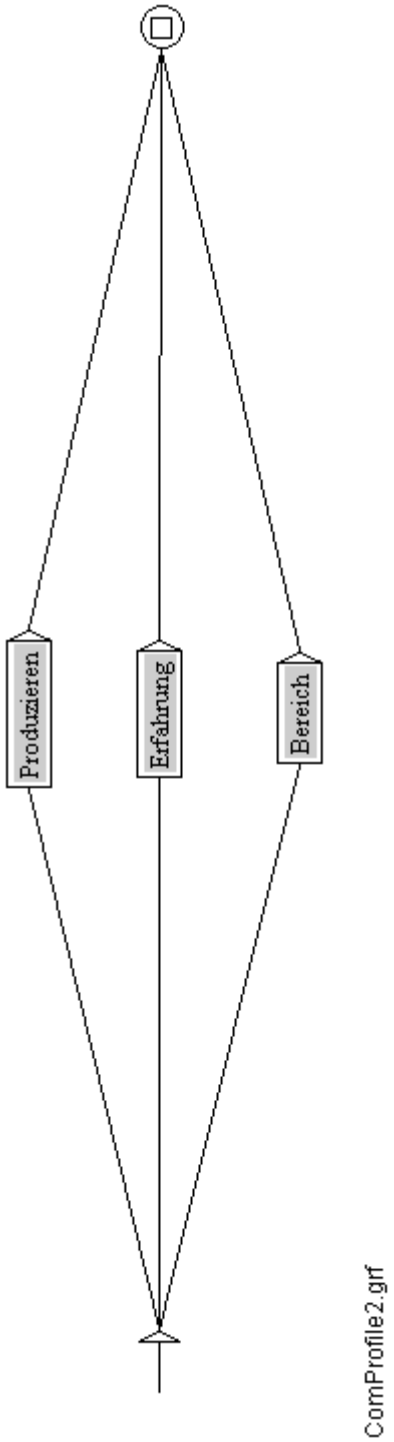
nummer.grf

7.3.3.3 Lokale Grammatiken der Firmenprofile

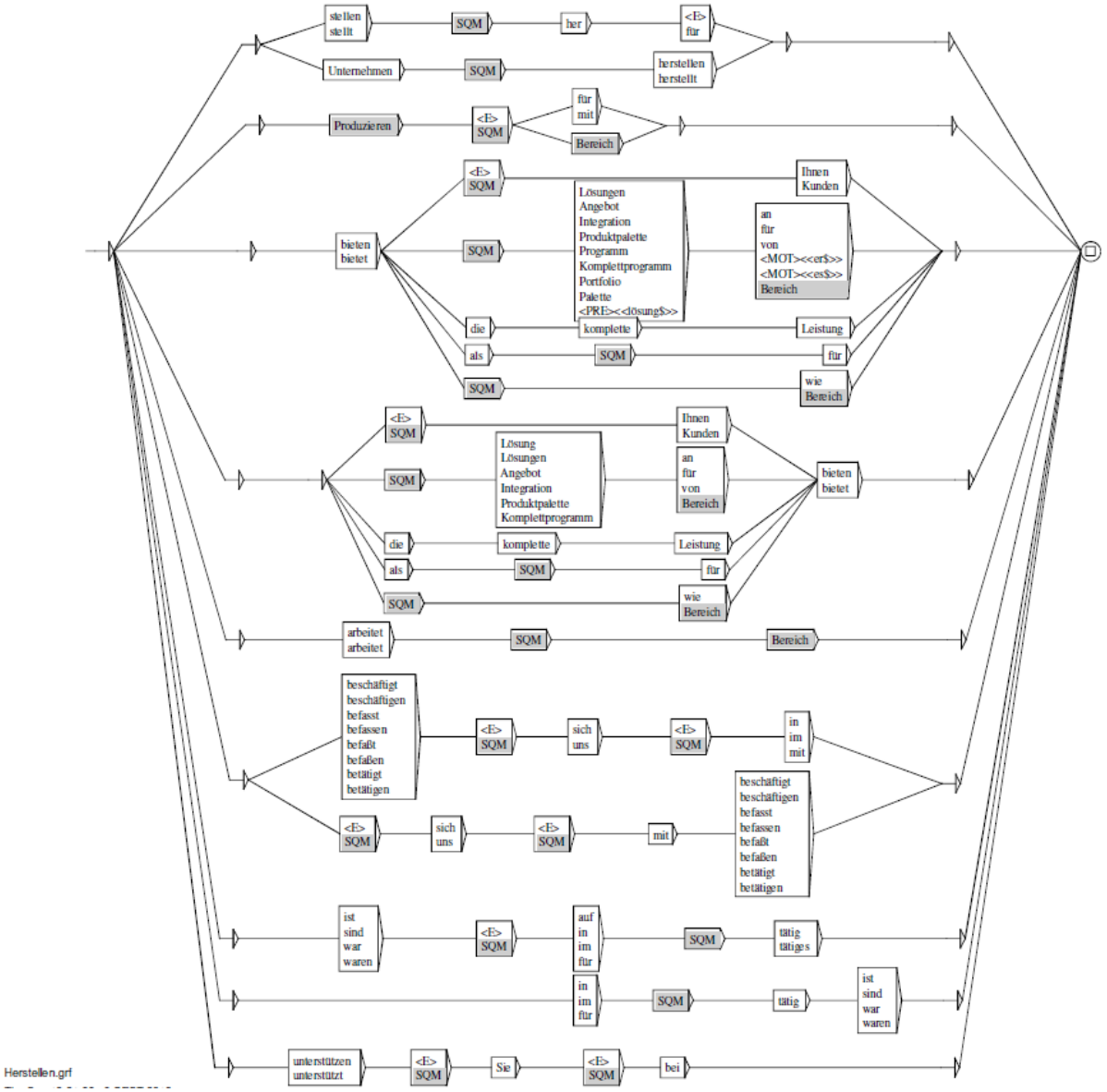
ComProfile1



ComProfile2

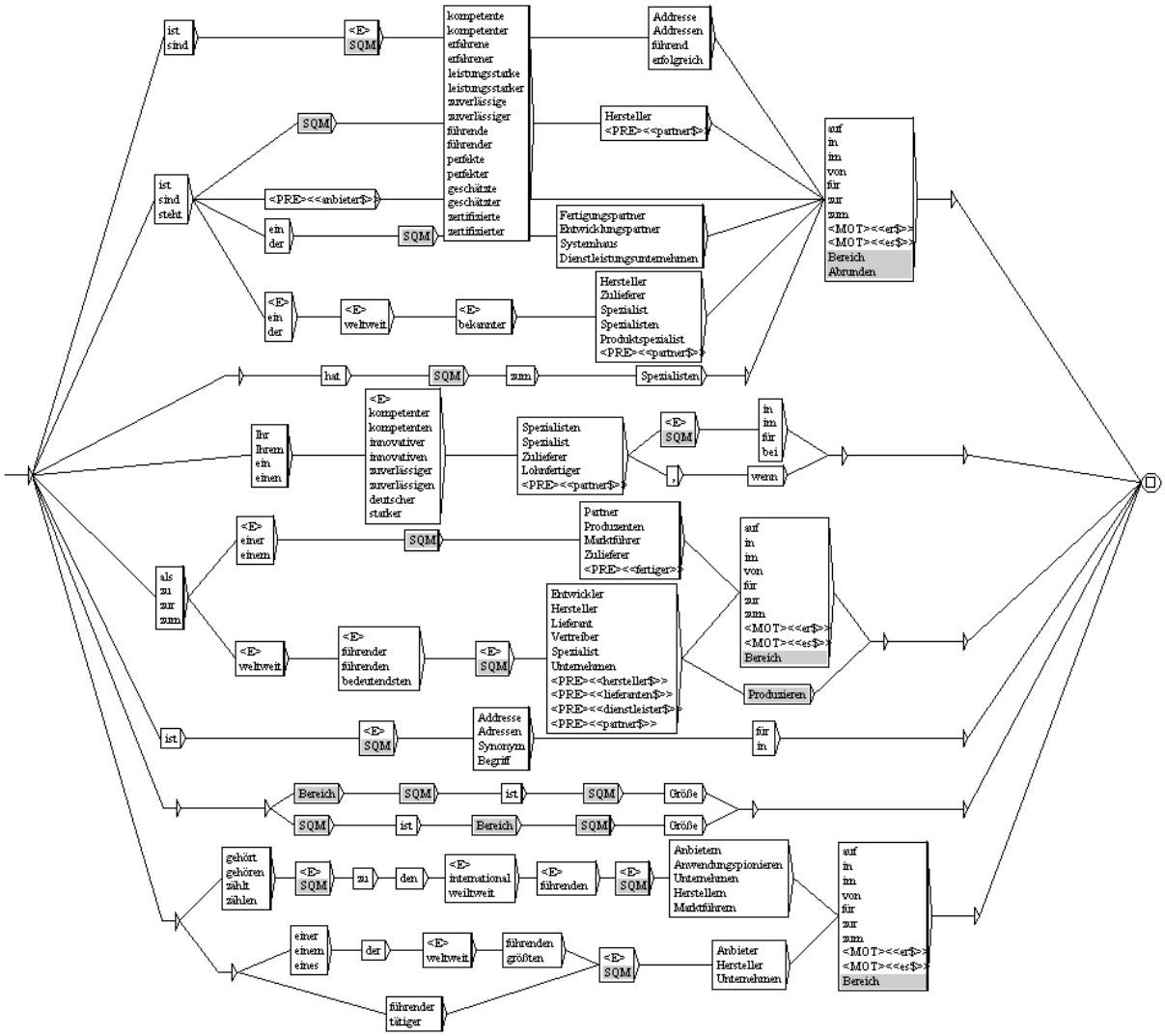


Subgragh: Herstellen



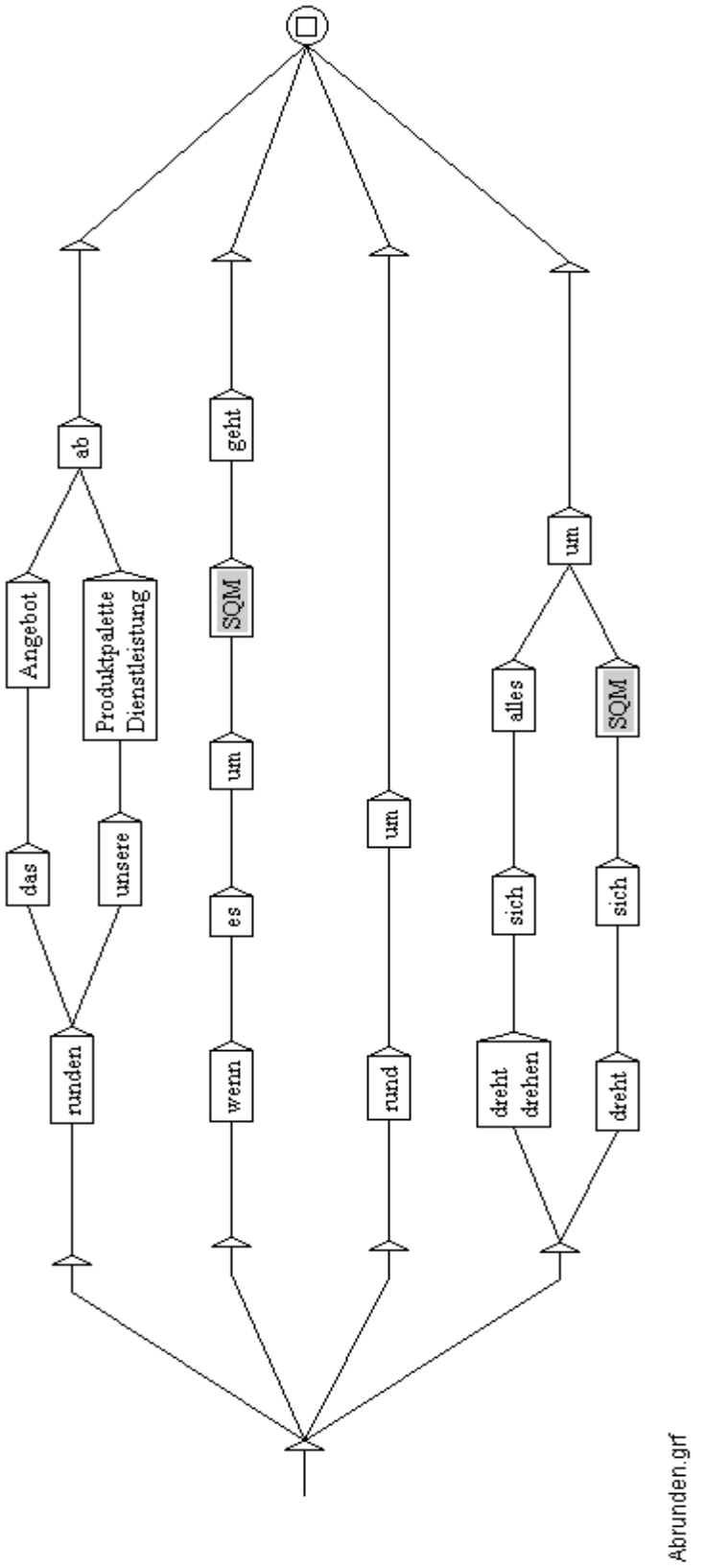
Herstellen.grf

Subgragh: Partner

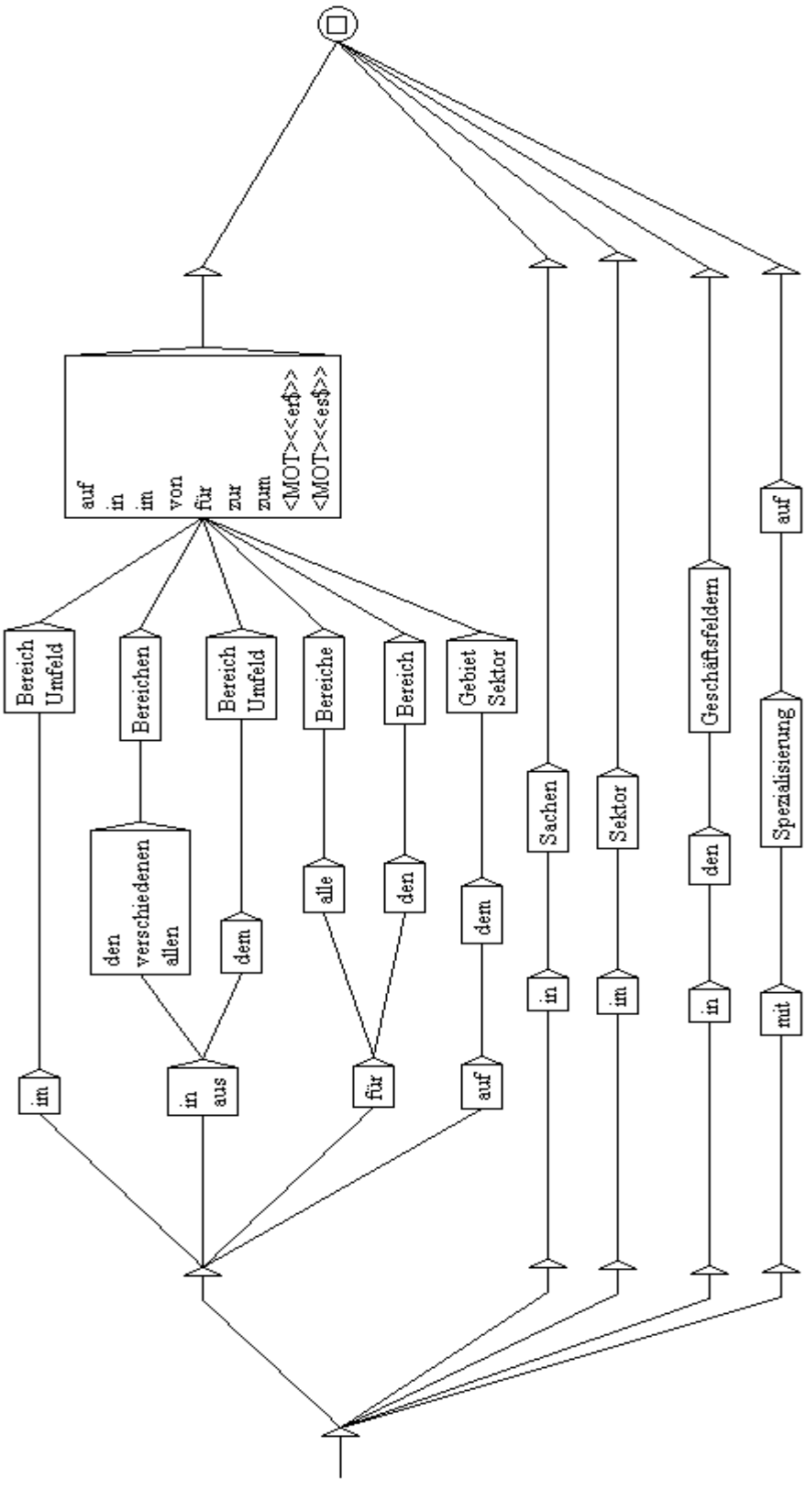


Partner.grf

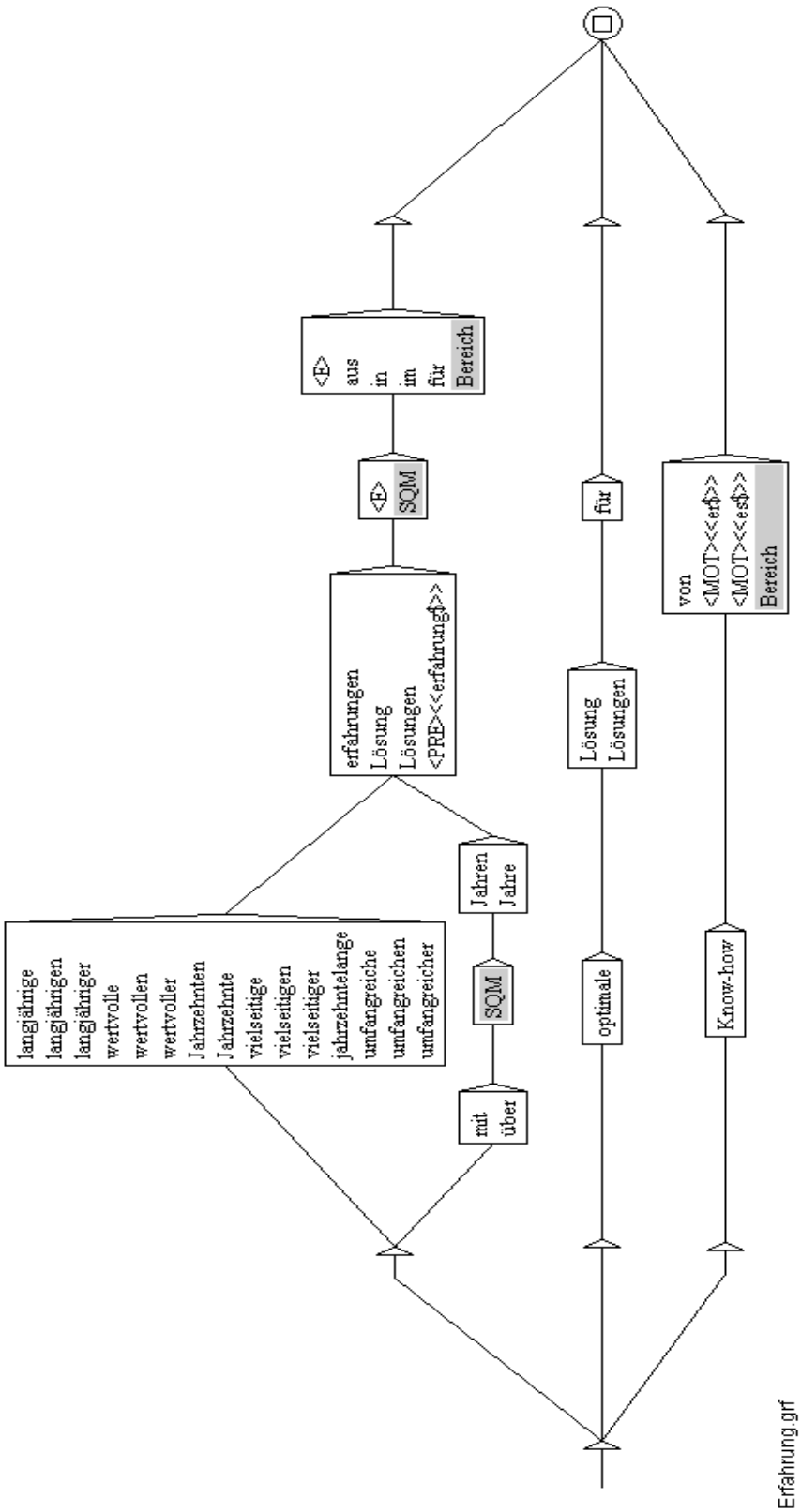
Subgragh: Abrunden



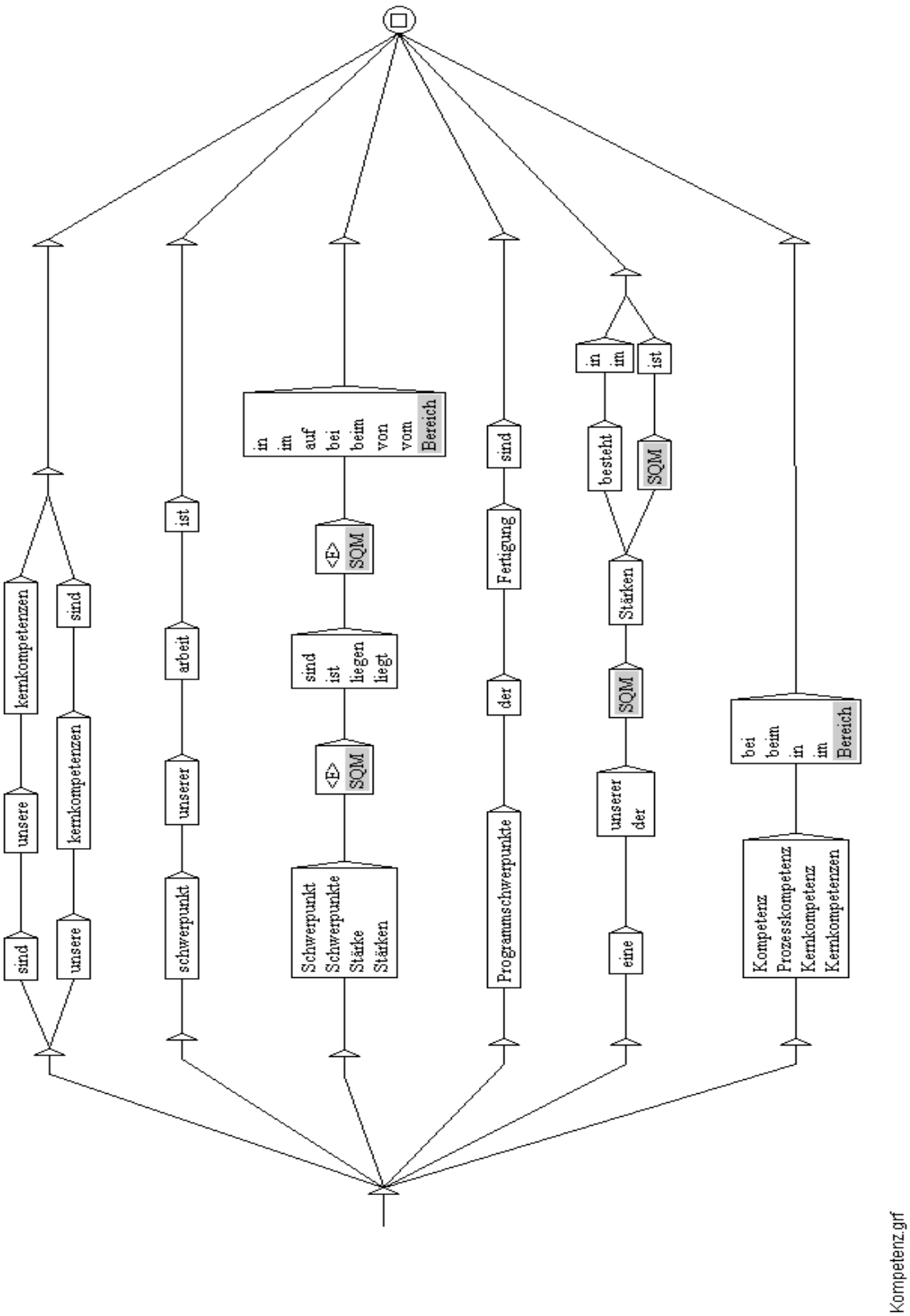
Subgragh: Bereich



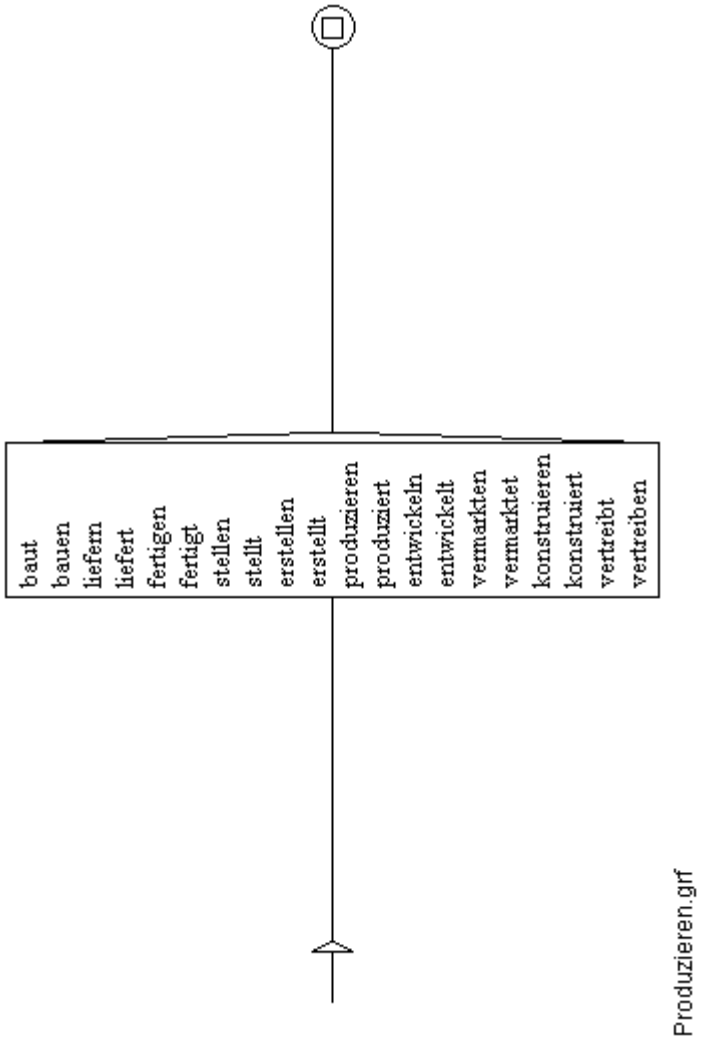
Subgragh: Erfahrung



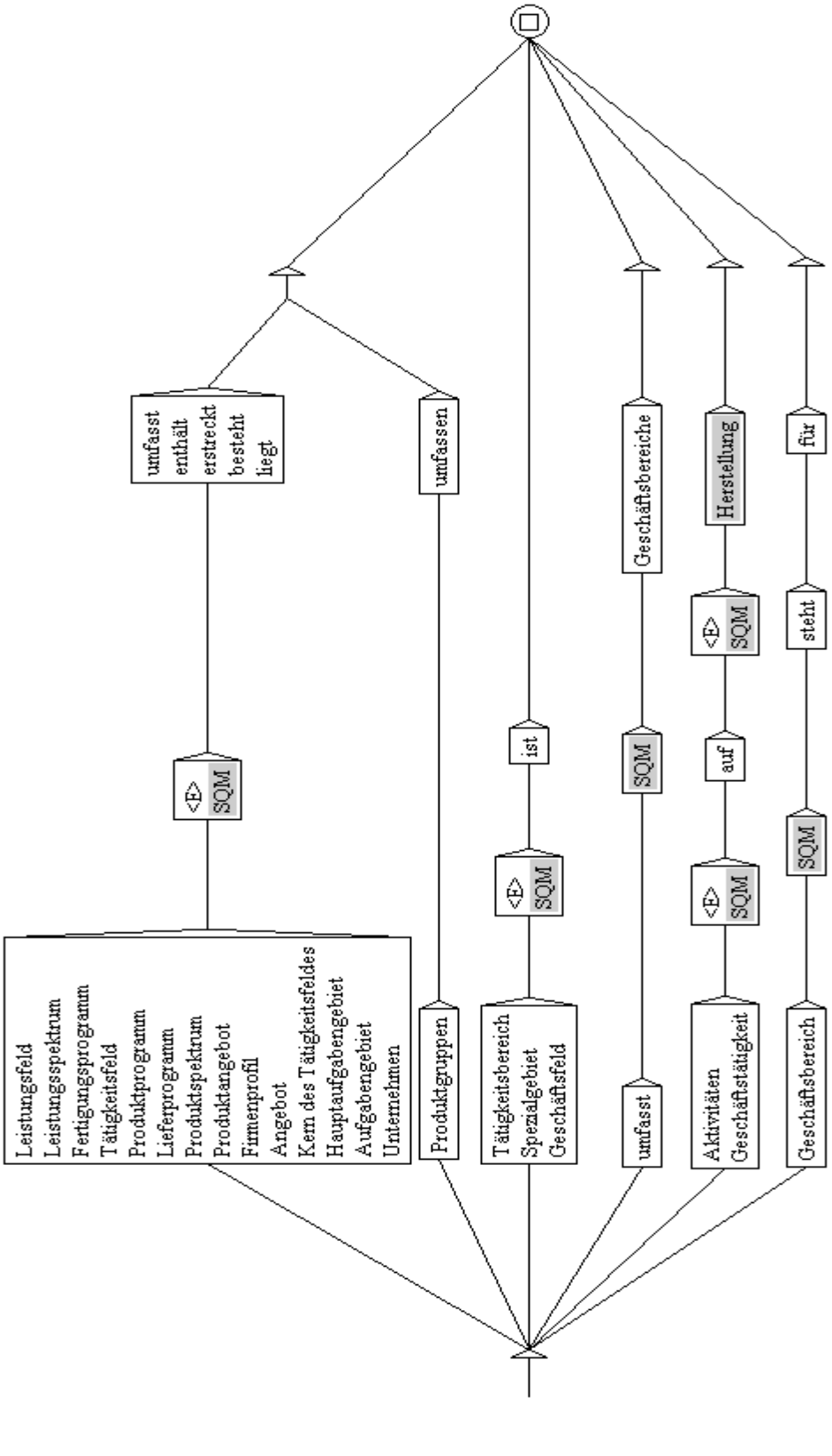
Subgragh: Kompetenz



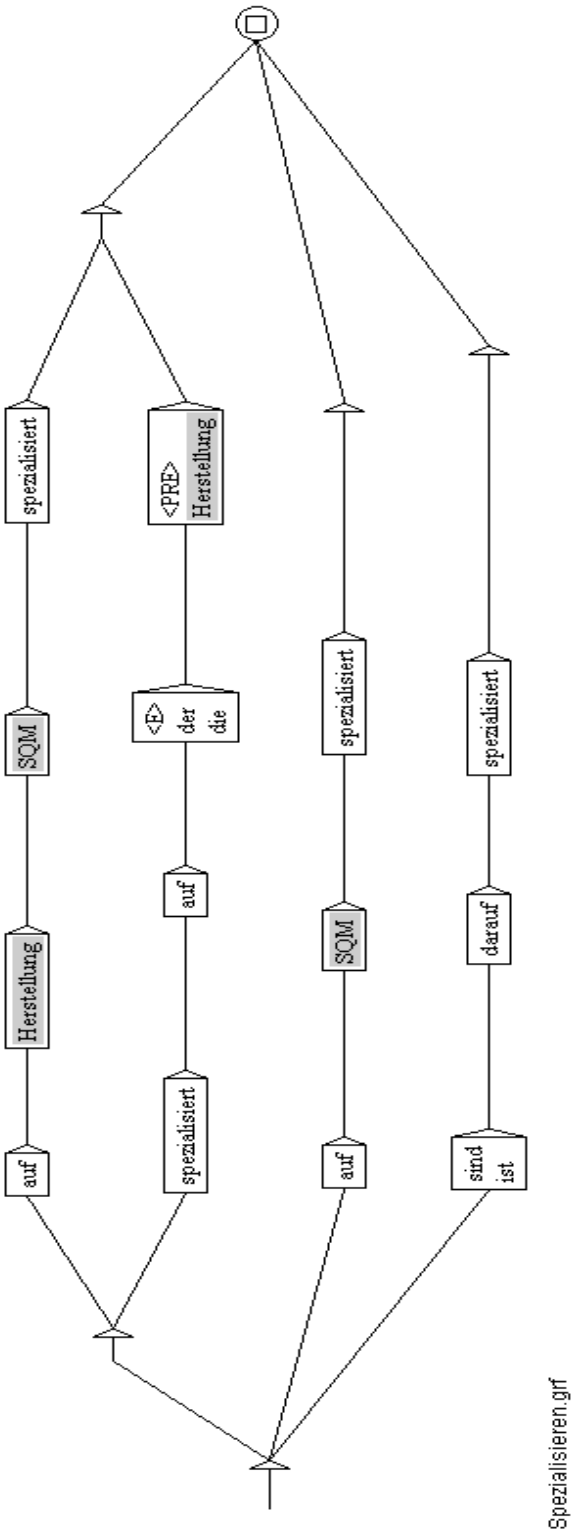
Subgragh: Produzieren



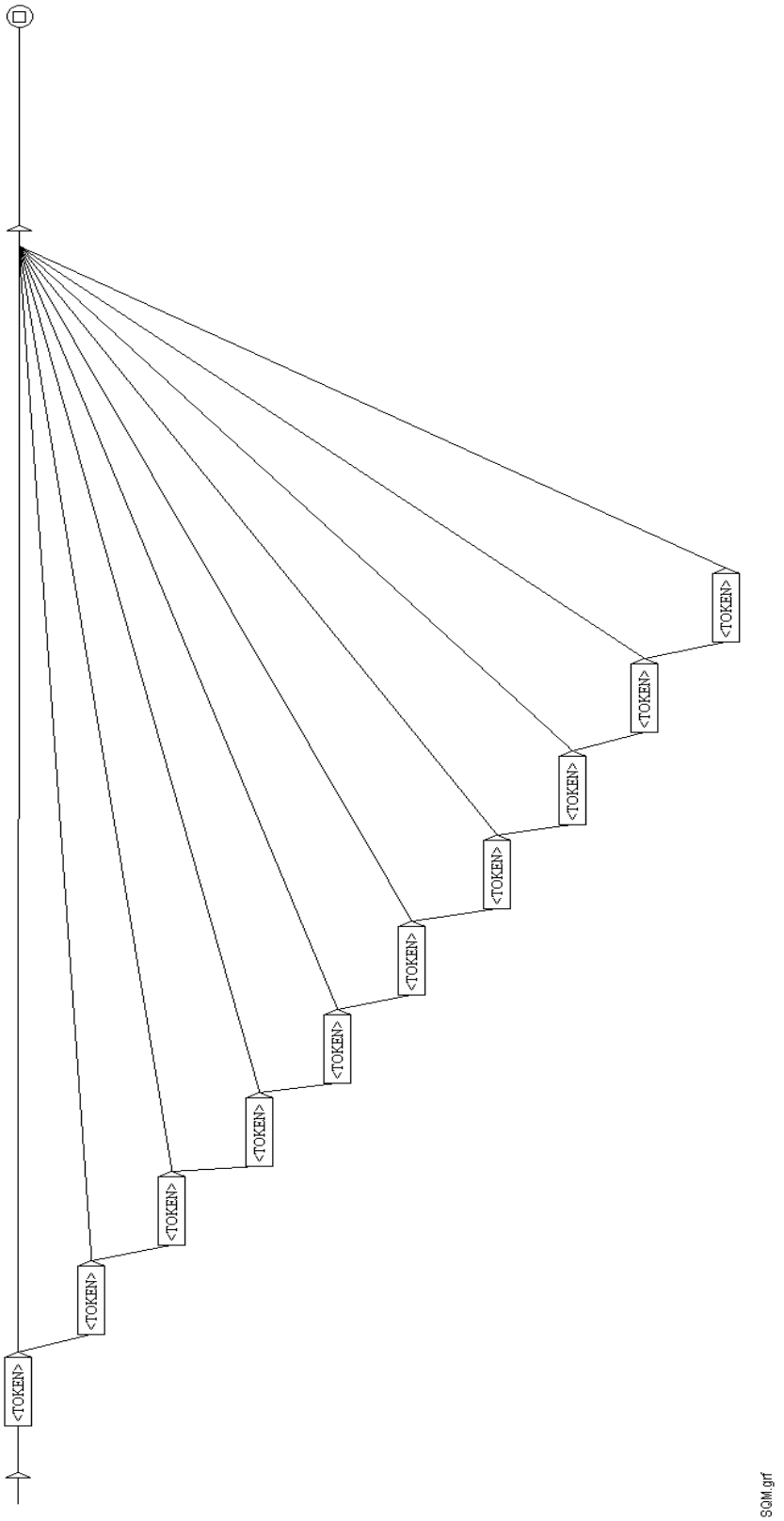
Subgragh: Programm



Subgragh: Spezialisieren



Subgragh: SQM



7.4 Indikatoren externer Kontexte

7.4.1 Firmennamen

Linke Seite
das
der
die
des
home
ihr
ihre
kontakt
impressum
profil
email
Domain-Name

7.4.2 Straßennamen

Knoten	Anzahl der Varianten
GmbH	D
mbH	A
AG	CH
Aktiengesellschaft	\d{5,}
e.K	.
eK	-
Kg	,
GbR	
oHG	
Ltd	
Inc	
Büro	
Fabrik	
Anlagen	
bau	
erei	
nik	
technology	
Konstruktion	
Betrieb	
Plattform	

Maschine Maschinen Home Kontakt Impressum Profile Profil Email . - , D A CH \d{5,} . - ,	

8. Literaturverzeichnis

Abney, S. (1996): Partial Parsing via Finite-State Cascade. In Proceedings, Workshop on Robust Parsing, 8th ESSLLI. Prague, 8-15

Adriaans, P. (2001): Learning shallow context-free languages under simple distributions. In: ILLC. (Online: http://dare.uva.nl/document/1211; Stand: 07.12.2012)

Adriaans, P. & van Zaanen, M. (2004): Computational Grammar Induction for Linguists, Grammars, Vol. 7. Tarragona, Spain, 57-68

Alpaydin, E. (2008): Maschinelles Lernen, Oldenbourg, München

Angluin, D. (1980): Inductive inference of formal languages from positive data. In: Information and Control, Vol. 45, 117-135

Angluin, D. (1987): Learning Regular Sets from Queries and Counterexamples. In: Information and Computation, Vol. 75, 87-106

Angluin, D. (1988): Negative results for equivalence queries, Machine Learning, 5. Boston: Kluwer Academic Publishers, 121-150

Atwell, E.; Demetriou, G. & Hughes, J. (2000): A comparative evaluation of modern English corpus grammatical schemes. In ICAME Journal, Vol. 24, 7-23.

Bahl, L.; Brown, P.; de Souza, P. & Mercer, R. (1986): Maximum mutual information estimation of hidden Markov model parameters for speech recognition. In: Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, 49-52

Baeza-Yates, R. & Ribeiro-Neto, B. (1999): Modern Information Retrieval. New York: ACM Press

Berko, J. (1958): The child's learning of English morphology. In: Word, Vol 14, 150-177

Bonnema, R.; Bod, R. & Scha, R. (1997): A DOP model for semantic interpretation. In Proceedings of the Association for Computational Linguistics/European Chapter of the Association for Computational Linguistics, Madrid, 159-167

Brill, E. (1993): Automatic grammar induction and parsing free text: a transformation-based approach. In: Proceedings of the workshop on Human Language Technology, 259 – 265

Bsiri, S.; Geierhos, M. & Ringlstetter, C. (2008): Structuring Job Search via Local Grammars. In: Advances in Natural Language Processing and Applications, Vol. 33, 201-212

Bunke, H. & Sanfeliu, A. (1990): Syntactic and structural pattern recognition theory and applications. Series in Computer Science, Vol. 7. Singapore: World Scientific.

Butt, M. & King, T. H. (2002): Urdu and the Parallel Grammar Project. In: Proceedings of COLING-2002 workshop on Asian Language Resources and International Standardization, Vol. 12, 1-3

Bünting, K. D. (1996): Einführung in die Linguistik. Weinheim: Athenäum

Charniak, E. (1993): Statistical Language Learning. Cambridge, MA: The MIT Press

Charniak, E. (2002): A maximum entropy-inspired parser. In: Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference, Washington, 132-139

Chen, S. F. (1995): Bayesian grammar induction for language modeling. In: Proceedings of the 33rd annual meeting on Association for Computational Linguistics, 228 – 235

Chidlovskii, B. Ragetli, J. & Rijke, M. (2000): Wrapper Generation via Grammar Induction. In: Machine Learning, ECML 2000, Vol. 1810, 96-108

Chomsky, N. (1956): Three models for the description of language. In: PGIT, 2(3), 113-124

Chomsky, N. (1965): Aspects of the Theory of Syntax. Cambridge, Massachusetts: The MIT Press

Chomsky, N. (1971): Cartesianische Linguistik: Ein Kapitel in der Geschichte des Rationalismus. Tübingen: Niemeyer

Christodoulopoulos, C.; Goldwater, S. & Steedman, M. (2010): Two Decades of Unsupervied POS induction: How far have we come? In: Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, 575-584

Chung, E. & Kuwahara, M. (1998): Comparative study of freeway incident detection algorithms. (Online: http://www.transport.iis.u-tokyo.ac.jp/publications/1998-021.pdf; Stand: 01.12.2012)

Collins, M. (1997): Three generative, lexicalised models for statistical parsing. In: Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics. Madrid, Spain, 16-23

Conklin, D. (2003): Music Generation from Statistical Models. In: Proceedings of the AISB 2003 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences, 30-35

Cruz-Alcazar, P. & Vidal-Ruiz, E. (1997): A study of Grammatical Inference Algorithms in Automatic Music Composition and Musical Style Recognition. In: Proceedings from the

"Workshop on Automata Induction, Grammatical Inference, and Language Acquisition". ICML97. (Online: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.48.2069&rep=rep1&type=pdf; Stand: 01.12.2012)

Cruz-Alcazar, P. & Vidal-Ruiz, E. (1998): Learning regular grammars to model musical style: Comparing different coding schemes. In: LNCS, Vol. 1433/1998, 211-222

Dempster, A. (1998): Logicist Statistics I. Models and Modeling. In: Statistical Science, Vol. 13(3), 248-276

Dennis, S. & Harrington, M. (2001): The Syntagmatic Paradigmatic Model: An distributed instance-based model of sentence processing. (Online: http://www.psychology.adelaide.edu.au/personalpages/staff/simondennis/NLPNN2001.pdf; Stand: 01.12.2012)

Dennis, S. (2004): An unsupervised method for the extraction of propositional information from text. In: Proceedings of the National Academy of Sciences, Vol. 101, 5206-5213.

Dennis, S. (2005): A Memory-based Theory of Verbal Cognition. In: Cognitive Science, Vol. 29, 145-193

Dorn, J. & Gottlob, G. (1997): Künstliche Intelligenz. In: P. Rechenberg & G. Pomberger (Hrsg.): Informatik-Handbuch. München, 975-998

Erdmenger, M. (1995): Computer im Fremdsprachenunterricht der Erwachsenenbildung. In: G. Burger (Hrsg.): Fremdsprachenunterricht in der Erwachsenenbildung. München: Hueber, 145-158

Erman, B. & Warren, B. (2000): The idiom principle and the open choice principle. In: S. Sarangi & D. Graddol (Hrsg.): Text, Vol. 20. Berlin: Mouton de Gruyter, 29-62

Fodor, J. A. & Pylyshyn, Z. W. (1988): Connectionism and cognitive architecture: A critical analysis. In: Cognition, Vol. 28, 3-71

Frawley, W.; Piatetsky-Shapiro, G. & Matheus, C. (1992): Knowledge Discovery in Databases: An Overview. AI Magazine, Vol. 13, 57-70

Friburger, N. & Maurel, D. (2002a): Finite-State Transducer Cascade to Extract Proper Names in Texts. In: LNCS (2494). Heidelberg: Springer, 115-124

Friburger, N. & Maurel, D. (2002b): Textual similarity based on proper names. In: MFIR 2002, at the 25th ACMSIGIR Conference. New York: ACM, 155-167

Fu, K. S. (1974): Syntactic methods in pattern recognition. In: R. Bellman (Hrsg.): Mathematics in Science and Engineering, Vol. 112. New York and London: Academic Press.

Fu, K. S. (1982): Syntactic Pattern Recognition and Applications. Englewood Cliffs, N.J.: Prentice-Hall,.

Fu, K. S. & Booth, T. (1986): Grammatical inference: Introduction and survey - Part I. In: IEEE Transactions on Pattern Analysis and Machine Intelligence, 95-111

Finch, S. P. & Cbater, N. (1992). Bootstrapping syntactic categories. In: Proceedings of the 14th Annual Conferenceof the Cognitive Science Society of America, 820-825

Garigliano, R.; Urbanowicz, A. & Nettleton, D. (1998): University of Durham: Description of the LOLITA System as used in MUC-7. (Online: http://acl.ldc.upenn.edu/M/M95/M95-1007.pdf; Stand: 07.12.2012)

Garrigues, M. (1995): Prepositions and the names of countries and islands: a local grammar for the automatic analysis of texts. In: Language Research, Vol. 31. Seoul: Seoul National University, 309–334

Gavrilis, D.; Tsoulos, I. & Dermatas, E. (2006): Evolutionary Grammar Induction for Protein Relation Extraction. In: 11-th International Conference Speech and Computer. (Online: http://users.cs.uoi.gr/~itsoulos/publications/protein.pdf; Stand: 02.12.2012)

Gold, E. M. (1967): Language Identification in the Limit. In: Information and Control, Vol. 10, 447-474

Goldin-Meadow, S.; Seligman, M. & Gelman, R. (1976): Language in the two-year old. In: Cognition, Vol. 4, 189-202

Gottlob, G.; Nejdl, W. (1990): Expert Systems in Engineering: Principles and Applications. In: International Workshop Vienna, Austria: Springer Verlag

Goutte, C.; Gaussier, E. (2005): A Probabilistic Interpretation of Precision, Recall and F-score, with Implication for Evaluation. In: Proceedings of 27th European conference on IR research (ECIR 2005), 345-359.

Grishman, R. (1995): The NYU System for MUC-6 or Where's the Syntax? In: Proceedings of the Message Understanding Conference (MUC-6). (Online: http://www.dtic.mil/dtic/tr/fulltext/u2/a460232.pdf; Stand: 02.12.2012)

Grishman, R. & Sundheim, B. (1995): Message Understanding Conference – 6: A Brief History. In: Proceedings of COLING 96. Copenhagen, 466-471

Gross, M. (1993): Local grammars and their representation by finite automata. In: M. Hoey (Hrsg.): Data, description, discourse. London: Harper Collins, 26–38

Gross, M. (1997): The Construction of Local Grammars. In: E. Roche & Y. Schabes (Hrsg.): Finite-state language processing. Cambridge: Bradford, 329-354

Gross, M. (1999): A Bootstrap Method for Constructing Local Grammars. In: N. Bakan (Hrsg.): Proceeding of the Symposium on Contemporary Mathematics, University of Belgmde, 229-250

Harbordt, S. (1974): Computersimulation in den Sozialwissenschaften, 2 Bände. Reinbek bei Hamburg: Rowohlt.

Harris, Zellig S. (1954): Distributional structure. Word, Vol. 10. New York: NY, Assoc, 146-162

Hasan, M.; Manian, V. & Kemke, C. (2007): HCP with PSMA: A Robust Spoken Language Parser. In: Proceedings of NeSy'2007. (Online: http://ftp.informatik.rwth-aachen.de /Publications/CEUR-WS/Vol-230/06-hasan.pdf; Stand: 03.12.2012)

Hellwig, P. (2011): Wissensrepräsentationen (Online: http://www.cl.uni-heidelberg.de /courses/archiv/ws06/ecl/folien/f_s83.pdf; Stand: 03.12.2012)

Hemphill, C.; Godfrey, J. & Doddington, G. (1990): The ATIS Spoken Language Systems Pilot Corpus. In: Proceedings of a Workshop on Speech and Natural Language. Pennsylvania: Hidden Valley, 96-101

Hess, M. (2004): Methoden der Künstlichen Intelligenz in der Sprachverarbeitung, Universität Zürich. (Online: https://files.ifi.uzh.ch/cl/hess/classes/mki/mki.0.pdf; Stand: 04.12.2012)

Hjelmslev, L. (1974): Langue und parole. In: Aufsätze zur Sprachwissenschaft. Stuttgart: Klett, 44-55

Hwa, R. (2000): Sample selection for statistical grammar induction. In: Proceedings of the 2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora, 45 - 52

Kaplan, D. (2009): Statistical Modeling, A Fresh Approach. (Online: http://www.macalester .edu/~kaplan/ISM/StatModeling-Review.pdf; Stand: 04.12.2012)

Karttunen, L.; Chanod, J.P.; Grefenstette, G. & Schiller, A. (1996): Regular expressions for language engineering. In: Natural Language Engineering, Vol. 2(4), 305-328

Kiss, G. R. (1973): Grammatical word classes: A learning process and its simulation. In: Gordon H. Bower (Hrsg.): Psychology of Learning and Motivation. New York: Academic Press, l-41.

Klein, D. & Manning, C.D. (2004): Natural language grammar induction with a generative constituent-context model. In: Pattern Recognition, Volume 38(9), 1407-1419

Klinge, K. (2007): Interkulturelles Training mit synthetischen Kulturen. Dissertation, Psychologisches Institut IV, Eestfälischen Wilhelms-Universität, Münster. (Online: http://d-nb.info/985918551/34; Stand: 04.12.2012)

Kruskal, J. (1983): An Overview of Sequence Comparison: Time Warps, String Edits, And Macromolecules. In: SIAM Review, Vol. 25, 201-237

Lee, Y. S. (2008): Website-Klassifikation und Informationsextraktion aus Informationsseiten einer Firmenwebsite. Dissertation, Centrum für Informations- und Sprachverarbeitung, Ludwig-Maximilians-Universität München. (Online: http://edoc.ub.uni-muenchen.de /8952/1/Lee_Yeong-Su.pdf; Stand: 04.12.2012)

Levenshtein, V. (1966): Binary codes capable of correcting deletions, insertions, and reversals. In: Soviet Physics Dokl, Vol. 10(8), 707–710

Lezius, W. (2001): Baumbanken. In: K.U. Carstensen; C. Ebert; C. Endriss; S. Jekat, R. Klabunde & H. Langer (Hrsg.): Computerlinguistik und Sprachtechnologie – Eine Einführung. Heidelberg: Spektrum Akademischer Verlag, 377-385

Lin D. (1998): Dependency-based evaluation of MINIPAR. In: Workshop on the Evaluation of Parsing Systems. (Online: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1 .99.6217&rep=rep1&type=pdf; Stand: 08.12.2012)

Lucas, S.M. (1993): New directions in grammatical inference. In IEE Colloquium on Grammatical Inference: Theory, Applications and Alternatives 1, 1-7

Luckhard, H. D. (1996): Automatische und intellektuelle Indexierung, Universität des Saalandes. (Online: http://scidok.sulb.uni-saarland.de/volltexte/2007/870/html/fiz1.fh-potsdam.de/volltext/saarland/03038.html; Stand: 06.12.2012)

Mallchok, F. (2004): Automatic Recognition of Organization Names in English Business News. Dissertation, Centrum für Informations- und Sprachverarbeitung, Ludwig-Maximilians-Universität München

Maratsos, M. (1988). The acquisition of formal word classes. In: Y. Levy, I. M. Schlesinger & M. D. S. Braine (Hrsg.), Categories and Processes in Language Acquisition. New Jersey: Hillsdale, 31-44

McDonald, D. (1993): Internal and External Evidence in the identification and Semantic Categorization of Proper Names. In: Proceedings of the Workshop on Acquisition of lexical Knowledge from Text, 32-43

Michalski, R. S. (1986): Understanding the Nature of Learning. In: R.S. Michalski, J.G. Carbonell & T.M. Mitchell (Hrsg.): Mashine Learning – An Artificial Intelligence Approach, Vol. 2 California: Morgen Kaufmann Publishers, 3-25

Minsky, M. (1974): A Framework for Representing Knowledge. In: International Journal of Cognitive Ergonomics, Vol. 3(1), 1999, 37-49

Mitchell, T.M. (1982): Generalization as Search. In: Artificial Intelligence, Vol. 18: 203-226

Mintz, T. H. (2003): Frequent frames as a cue for grammatical categories in child directed speech. In: Cognition, Vol. 90(1), 91 - 117

Mohri, M. (1994): Syntactic analysis by local grammars automata: an efficient algorithm. In: Proceedings of COMPLEX ’94. (Online: http://xxx.lanl.gov/pdf/cmp-lg/9407002; Stand: 05.12.2012)

Mohri, M. & Sproat, R. (2006): On a common fallacy in computational linguistics. In: Festschrift in Honour of Fred Karlsson, 432-439

Morik, K. (1993): Maschinelles Lernen, Universität Dortmund. (Online: https://eldorado.tu-dortmund.de/bitstream/2003/2574/1/morik_93c.pdf; Stand: 06.12.2012)

Nagel, S. (2008): Lokale Grammatiken zur Beschreibung von lokativen Sätzen und Ihre Anwendung im Information Retrieval, Dissertation, Computerlinguistik, Ludwig-Maximilians-Universität München.

Nowell, A. (1973): Production Systems: Models of Control Structures. In: W.G. Chase (Hrsg.): Visual information processing. Toronto: Academic Press, 463-562

Ortolf, C. (2008): Effiziente Text Suche. (Online: http://www.informatik.uni-freiburg.de/~ipr/ipr_teaching/ss_08/homework/28_05_08_Suchmaschinen.pdf; Stand: 07.12.2012)

Paumier, S. (2003): Unitex 2.1 User Manual. Université Paris-Est Marne-la-Vallée. (Online: http://www-igm.univ-mlv.fr/~unitex; Stand: 10.12.2012)

Rabiner, L.R. & Juang, B.H. (1986): An Introduction to Hidden Markov Models. In: IEEE ASSP Magazine, January 1986, 4-16

Redington, M.; Chater N. & Finch, S. (1998): Distributional Information: A Powerful Cue for Acquiring Syntactic Categories. In: Cognitive Science, Vol. 22(4), 425-469

Reimer, U. (1991): Einführung in die Wissensrepräsentation – Netzartige und schemabasierte Repräsentationsformate. Stuttgart: Teubner.

Ristov, S. (2002): Using Inverted Files to Compress Text. In: Journal of Computing and Information Technology - CIT 10, 2002, 3, 157-161

Roberts, A.; Atwell, E. (2003): The use of corpora for automatic evaluation of grammar inference systems. In: Proceedings International Conference on Corpus Linguistics. (Online: http://www.andy-roberts.net/res/writing/andyrCL03a.pdf; Stand: 07.12.2012)

Roche, E. (1992): Text disambiguation by finite state automata, an algorithm and experiments an corpora. In: Proceeding of the 14th conference on Computational linguistics, Vol. 3. (Online: http://acl.ldc.upenn.edu/C/C92/C92-3153.pdf; Stand: 07.12.2012)

Roche, E. & Schabes, Y. (1996): Introduction to Finite-State Devices in Natural Language Processing. Mitsubishi Electric Research Laboratories. (Online: http://www.merl.com /publications/docs/TR96-13.pdf; Stand: 08.12.2012)

Roth, J. (2002): Der Stand der Kunst in der Eigennamen-Erkennung - Mit einem Fokus auf Produktennamen-Erkennung. Universität Zürich. (Online: http://www.cl.uzh.ch/studies /theses/lic-master-theses/lizjeannetteroth.pdf; Stand: 08.12.2012)

Rössler, M. (2006): Korpus-adaptive Eigennamenerkennung, Dissertation, Computerlinguistik, Universität Duisburg-Essen. (Online: http://duepublico.uni-duisburg-essen.de /servlets/DerivateServlet/Derivate-16089/diss_final2007_DS.pdf; Stand: 09.12.2012)

Rubenstein, H.; Goodenough, J.B. (1965): Contextual correlates of synonymy. Communications of the ACM, Vol. 8(10), 627-633

Sahlgren, M. (2006): The distributional hypothesis. (Online: http://soda.swedish-ict.se/3941/1 /sahlgren.distr-hypo.pdf; Stand: 07.12.2012)

Samuel, A. L. (1959): Some Studies in Machine Learning Using the Game of Checkers. In: IBM Journal of Research and Development, Vol. 3, 211-229

Samuelsson, C. & Voutilainen, A. (1997): Comparing a Linguistic and a Stochastic Tagger. In: Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics. (Online: http://acl.ldc.upenn.edu/P/P97/P97-1032.pdf; Stand: 02.12.2012)

Schmalhofer, F. (1994): Maschinelles Lernen: Eine kognitionswissenschaftliche Betrachtung. Deutsches Forschungszentrum für Künstliche Intelligenz. (Online: http://www.dfki.uni-kl.de/dfkidok/publications/D/94/03/D-94-03.pdf; Stand: 04.12.2012)

Schmidt, R. (1990): Das Konzept einer Lerner-Grammatik. In: Gross, Harro & Fischer, Klaus (Hrsg.), Grammatikarbeit im Deutsch-als-Fremdsprache-Unterricht. München: ludicium, 153-161

Schütze, H. & Pedersen, J. O. (1995): Information Retrieval based on Word Senses. (Online: ftp://parcftp.parc.xerox.com/pub/qca/papers/sdair95.senseir.ps.Z; Stand: 02.12.2012)

Scott, P. (1983): Learning: The Construction of a Posteriori Knowledge Structures. In: Proceedings of the Third National Conference on Artificial Intelligence, AAAI-83, 359-363

Senellart, J. (1998): Locating noun phrases with finite state transducers. In: Proceedings of the 17th International Conference on Computational Linguistics (COLING-98), 1212–1219.

Senellart, J.; Dienes, P. & Váradi, T. (2001): New generation Systran translation system. In: Proceeding of MT Summit VII. (Online: http://citeseerx.ist.psu.edu/viewdoc /download?doi=10.1.1.68.568&rep=rep1&type=pdf; Stand: 07.12.2012)

Sester, M. (1995): Lernen struktureller Modelle für die Bildanalyse, Dissertation, Universität Stuttgart. (Online: http://elib.uni-stuttgart.de/opus/volltexte/2001/839/pdf/moni_diss.pdf; Stand: 07.12.2012)

Sester, M. (2005): Künstliche Intelligenz. (Online: http://www.ikg.uni-hannover.de/geosensor /lehre/katalog/gis/ki-intro.pdf; Stand: 07.12.2012)

Shannon, C. (1951): Prediction and entropy of printed English. In: Bell Systems Technical Journal 30, 50-64.

Simon, H. A. (1983): Why Should Machine Learn? In: R.S. Michalski, J. Carbonell, & T.M. Mitchell (Hrsg.): Machine Learning - An artificial Intelligence Approach. Palo Alto, CA: Tioga Publishing, 25-38

Solan, Z.; Rupping, E.; Horn, D. & Edelman, S. (2003): Automatic Acquisition and Efficient Representation of Syntactic Structures. In: S. Becker, S. Thrun & K. Obermayer (Hrsg.): Advances in Neural Information Processing Systems 15. Cambridge: The MIT Press, 107-114

Solan, Z. (2006): Unsupervised Learning of Natural Languages. PhD Thesis. Tel Aviv University. (Online: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1187953/; Stand: 08.12.2012)

Solomonoff, R. (1997): The Discovery of Algorithmic Probability. In: Journal of Computer and System Sciences, Vol. 55, 73-88.

Stachowiak, H. (1973): Allgemeine Modelltheorie. Wien, New York: Springer.

Stangl, W. (2008): Arbeitsblätter von Werner Stangl. (Online: http://arbeitsblaetter.stangl-taller.at/LERNEN/; Stand: 07.12.2012)

Stolcke, A. & Omohundro, S. (1994): Inducing probabilistic grammars by Bayesian model merging. In: Proceedings of the Second International Colloquium on Grammatical Inference and Applications, ICGI'94, 106-118

Stumper, B.; Bannard, C.; Lieven, E. & Tomasello, M. (2009): "Frequent Frames" in German Child-Directed Speech: A Limited Cue to Grammatical Categories. In: Cognitive Science, Vol. 35 (6), 1190-1205

Tarvainen, K. (1981): Einführung in die Dependenzgrammatik. Tübingen: Max Niemeyer Verlag, 2000

Turing, A. M. (1947): Lecture on the automatic computing engine. In: B. Jack Copeland (Hrsg.): The Essential Turing: Seminal Writings in Computing, Logic, Philosophy, Artificial Intelligence, and Artificial Life: Plus The Secrets of Enigma. Oxford University Press, 363-394

Turing A. M. (1950): Computing Machinery and Intelligence. In: Mind – a quarterly review of Psychology and Philosophy, Vol. 59. Oxford University Press, 443-460

Valiant, L.G. (1984): A Theory of the Learnable. In: Communication of the ACM, Vol. 27(11), 1134-1142

Van Zaanen, M. (2000): ABL: Alignment-Based Learning. In: Proceedings of the 18th conference on Computational linguistics, Vol. 2, 961-967

van Zaanen, M. (2001): Bootstrapping structure into language: alignment-based learning. PhD thesis, School of Computing, University of Leeds. (Online: http://arxiv.org/pdf/cs /0205025.pdf; Stand: 11.12.2012)

van Zaanen, M. & Geertzen, J. (2008): Problems with Evaluation of Unsupervised Empirical Grammatical Inference Systems. In: ICGI 2008, 301-303

Wagner, R. & Fischer, M. (1974): The String-to-String Correction Problem. In: Journal of the Association for Computing Machinery, Vol. 21(1), 168-173

Weisgerber, B. (1982): Über die Roller der Grammatik beim Erwerb von Muttersprache und Fremdsprache. In: C. Gnutzmann & D. Stark (Hrsg.): Grammatikunterricht: Beiträge zur Linguistik und Didaktik des Fremdsprachenunterrichts. Tübingen: Narr, 101-126

Winograd, T. (1983): Language as a Cognitive Process, Vol. 1: Syntax. United States: Addison-Wesley

Wolff, G. (2003): Information Compression by Multiple Alignment, Unification and Search as a Unifying Principle in Computing and Cognition. In: Artificial Intelligence Review, Vol. 19(3), 193-230

Woods, W. A. (1970): Transition Network Grammars for Natural Language Analysis. In: Communications of the ACM, Vol. 13(10), 591-606

Zimmermann, H. (1974): Ein Konzept zur syntaktischen Oberflächenanalyse. In: G. Nickel, A. Raasch (Hrsg.): IRAL-Sonderband: Kongressbericht der 5. Jahrestagung der Gesellschaft für Angewandte Linguistik GAL e.V. Heidelberg: Groos, 172-179

Zimmermann, H. (1979): Ansätze einer realistischen automatischen Indexierung unter Verwendung linguistischer Verfahren. In: Kuhlen, R. (Hrsg.): Datenbasen, Datenbanken, Netzwerke, Bd. 1: Aufbau von Datenbasen. München: Saur, 311-338

Zimmermann, H. (1990): Computer und Sprache im Zeitalter der Fachinformation. In: Lebende Sprachen, Band 35, Heft 1, 1-5