Modellierung natürlicher Sprache mit Hilfe von Topologischer Dependenzgrammatik

Abschlussbericht des Fortgeschrittenenpraktikums
von Mathias Möhl
betreut von Ralph Debusmann
unter der Aufsicht von Prof. Gert Smolka

3.5.2004

Einleitung

Das Ziel dieses Fortgeschrittenenpraktikums war es, eine deutsche Grammatik für das TDG-System (*Topological Dependency Grammar*) [1] bzw. das sich daraus entwickelnde allgemeinere XDG-System (*Extensible Dependency Grammar*) zu erzeugen.
Zu diesem Grammatikformalismus, der sich besonders dadurch auszeichnet, dass er mehrere unabhängige Ebenen der Strukturierung erlaubt, existieren bisher nur einige handgeschriebene Grammatiken, die diverse Phänomene der deutschen Sprache abdecken, aber nur über einen sehr geringen Wortschatz verfügen. Die in diesem Praktikum erzeugte Grammatik wird im Gegensatz dazu mit Hilfe einer umfangreichen Baumdatenbank automatisch generiert. Bei dieser Datenbank handelt es sich um den *Negra-Korpus*, der aus etwa 20.000 Sätzen deutscher Zeitungstexte besteht.
Die grammatikalischen Informationen, die in dem Korpus zu jedem Satz gespeichert sind, reichen aus, um die erste Ebene der TDG-Grammatik, die ID-Ebene (*Immediate Dominance*), relativ direkt herzuleiten. Die zweite Ebene der TDG-Grammatik, die LP-Ebene (*Linear Precedence*), unterteilt den Satz in unterschiedliche topologische Felder. Da der Negra-Korpus keine Informationen zu den topologischen Feldern der einzelnen Sätze enthält, sollte in dem Praktikum das in [2] vorgestellte System "Unsupervised Learning of Word Order Rules" verwendet werden, um die zur Erzeugung der LP-Ebene benötigten Informationen zu erhalten. Da die Struktur dieses Systems sehr modular ist, man es leicht erweitern kann und es neben den gelernten topologischen Feldern auch noch die benötigte Funktionalität zum Einlesen des Korpus zur Verfügung stellt, wurde das System direkt um die Funktionalität erweitert, eine XDG-Grammatik zu erzeugen. Diese Variante erwies sich als deutlich einfacher und flexibler als die Entwicklung eines komplett eigenständigen Systems.

1 Der XDG-Grammatik-Formalismus

Bei dem XDG-Grammatikformalismus (*Extensible Dependency Grammar*) handelt es sich um einen deklarativen Grammatikformalismus für Dependenz-Grammatiken. XDG ist eine verallgemeinerte Fortentwicklung von TDG, und erlaubt im Speziellen das Schreiben von Grammatiken mit einer beliebigen Anzahl von Ebenen der Strukturierung. TDG ist in XDG enthalten. Da wir in diesem Projekt nur den TDG-Teil von XDG benutzen, können wir die beiden Abkürzungen bei Bedarf gegen einander austauschen.Um den Aufbau und das Prinzip des Systems zu verstehen, sollte man zunächst einmal die generelle Idee von Dependenz-Grammatiken betrachten.

1.1 Dependenz-Grammatiken im Allgemeinen

Dependenz-Grammatiken gehen von der Annahme aus, dass bestimmte Teile innerhalb eines Satzes direkt von anderen Teilen abhängig sind. Wie diese

Abhängigkeiten oder Dependenzen genau aussehen, variiert innerhalb der verschiedenen Dependenz-Grammatiken. So könnte beispielsweise das Subjekt des Satzes vom Prädikat abhängen oder ein Artikel von dem Nomen, dem er vorangestellt ist.
Diese Abhängigkeiten werden mit Hilfe von Dependenz-Relationen formalisiert. Dependenz-Relationen sind gerichtete und getypte Relationen: gerichtet, da die Abhängigkeiten in aller Regel nur in einer Richtung bestehen (der Artikel ist vom Nomen abhängig, aber nicht umgekehrt) und getypt, da über den Typ die Art der Abhängigkeit festgelegt und unterschieden wird. So sind etwa in dem Satz "Maria liebt Peter" sowohl "Maria" als auch "Peter" von "liebt" abhängig, wobei "Maria" das zu "liebt" gehörige Subjekt und "Peter" das zu "liebt" gehörige Objekt ist. Die Relation zwischen "Maria" und "liebt" ist also vom Typ Subjekt und die Relation zwischen "Peter" und "liebt" vom Typ Objekt.
Die letzte wesentliche Gemeinsamkeit von Dependenz-Grammatiken besteht in der Annahme, dass einige Wörter bestimmte abhängige Wörter benötigen, um einen Sinn zu ergeben. Dieses Phänomen wird als Valenz bezeichnet. So benötigt "liebt" immer ein Subjekt und ein Objekt. Andere Abhängigkeiten können optional sein. So können Verben zum Beispiel in der Regel mit Adverbien näher beschrieben werden; das Weglassen der Adverbien führt aber nicht dazu, dass der Satz keinen Sinn mehr ergibt. Ein Beispiel hierfür ist der Satz "Maria liebt Peter abgöttisch". "liebt Peter abgöttisch" ist kein sinnvoller vollständiger Satz, "liebt" benötigt ein von ihm abhängiges Subjekt. Das Wort "abgöttisch" kann dagegen weggelassen werden, ohne dass der Satz seinen Sinn verliert.
Alle Abhängigkeiten innerhalb eines Satzes können mit Hilfe eines Dependenz-Baumes dargestellt werden. Dabei entspricht jeder Knoten des Baums einem Wort. Die ausgehenden Kanten eines Knotens führen zu den von dem Knoten gemä den Dependenz-Relationen abhängigen Knoten und sind mit dem Typ der Relation beschriftet. Zusätzlich gibt es gepunktete Linien, die jeden Knoten mit dem dazugehörigen Wort verbinden. Ein Beispiel-Dependenzbaum zu dem Satz "Maria wird einen Mann lieben können" ist in Abbildung 1 dargestellt.

Welche Abhängigkeiten innerhalb eines Satzes erlaubt sind, wird in den meisten Formalismen durch ein Lexikon festgelegt. Das Lexikon legt zu jedem Wort die Valenz fest, also welche abhängigen Elemente benötigt werden oder optional sind und auch von welchen anderen Elementen das Wort seinerseits abhängig sein kann.

1.2 Besonderheiten des XDG-Grammatik-Formalismus

Wie bereits erwähnt, gibt es für Dependenz-Grammatiken verschiedene Möglichkeiten, die Abhängigkeiten zwischen den einzelnen Teilen eines Satzes festzulegen. So kann man mit Hilfe der Dependenz zum Beispiel entweder nur die grammatikalischen Rollen einzelner Satzteile wie Subjekt und Objekt unterscheiden, was zu relativ grob strukturierten Dependenz-Bäumen führt, oder man kann etwa das Subjekt selbst noch weiter aufspalten in das zentrale Nomen, einen davon abhängigen Artikel und ergänzende Adjektive. Neben dem Grad der Genauigkeit oder Feinheit, mit der die Abhängigkeiten innerhalb einzelner Sätze

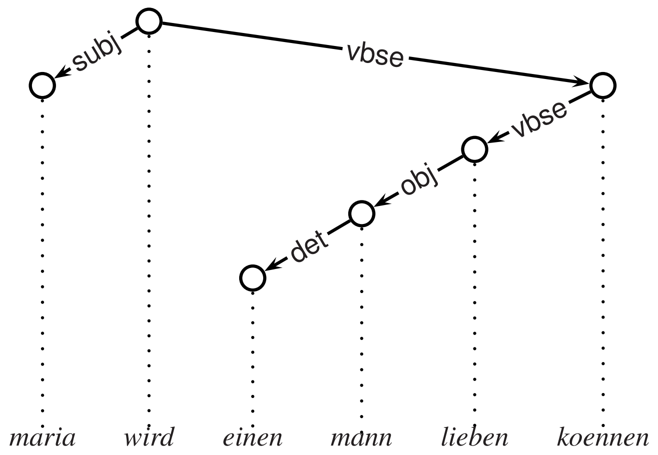

Abbildung 1: Ein Beispiel-Dependenzbaum

aufgeschlüsselt werden, kann man auch nach der Funktion eine Unterscheidung treffen, die durch die Dependenzen repräsentiert wird. So können die Dependenzen zum Beispiel die rein syntaktische Struktur des Satzes widerspiegeln oder im Gegensatz dazu auch eine semantische Struktur beschreiben.

Ein zentrales Merkmal des XDG-Systems besteht nun darin, dass es nicht auf die Festlegung einer einzigen solchen Ebene beschränkt ist, sondern die parallele Definition mehrerer Ebenen erlaubt. Für die Analyse eines Satzes werden dann dementsprechend mehrere Dependenz-Bäume generiert - ein Baum für jede Ebene der Grammatik. So kann zum Beispiel ein Baum die syntaktische Struktur des Satzes bis ins Detail aufschlüsseln, während ein zweiter Baum nur die gröbere syntaktische Struktur repräsentiert und ein dritter Baum Informationen über die semantischen Bezüge innerhalb des Satzes enthält.

Definiert werden die einzelnen Ebenen mit Hilfe sogenannter Prinzipien, die mit Hilfe von *Constraints* formalisiert werden. Die Prinzipien können entweder die Dependenz-Relationen innerhalb einer Ebene charakterisieren oder auch Beziehungen zwischen den einzelnen Ebenen beschreiben. So können in einer Grammatik zum Beispiel bestimmte Übereinstimmungen in der Baumstruktur verschiedener Ebenen gefordert werden.

Das XDG-System ist eine Weiterentwicklung und Verallgemeinerung des TDG-Systems (*Topological Dependency Grammar*). Während das TDG-System noch auf zwei konkrete Ebenen - die ID- und die LP-Ebene - festgelegt ist, kann man im XDG-System relativ frei Grammatiken mit beliebig vielen unterschiedlichen Ebenen entwickeln. Dabei geht die Flexibilität sogar so weit, dass die Dependenz-Bäume in den einzelnen Ebenen nicht zwangsläufig auf Baumstrukturen beschränkt sind. Vor allem in Ebenen, die semantische Bezüge repräsentieren, kann es sinnvoll sein, allgemeine Dependenz-Graphen zu erlauben. Die Grammatik, die im Rahmen dieser Arbeit entwickelt wurde, ist für den Einsatz mit dem XDG-System gedacht und wurde mit diesem getestet. Die Struktur der Grammatik entspricht allerdings noch genau der Struktur von Grammatiken des TDG-Systems mit der Unterteilung in ID-Ebene und LP-Ebene mit Hilfe der jeweiligen Prinzipien des TDG-Systems. Von den erweiterten Möglichkeiten des XDG-Systems wird in dieser Grammatik also noch kein Gebrauch gemacht. Deswegen werden im Folgenden auch nur die vom TDG-System übernommenen Ebenen des Systems erklärt.

1.3 Die ID-Ebene

Die ID-Ebene (*Immediate Dominance*) entspricht der traditionellen Dependenz-Ebene, in der die Relation zwischen den einzelnen Knoten im Dependenz-Baum den grammatikalischen Rollen der einzelnen Wörter innerhalb des Satzes entspricht. Da ausschlielich die grammatikalischen Bezüge in dieser Ebene analysiert werden, spielt die Reihenfolge der Wörter im ursprünglichen Satz keine Rolle. Eine direkte Konsequenz dieser Tatsache ist, dass die Dependenz-Bäume ungeordnet sind.
Insgesamt gibt es drei Prinzipien in der ID-Ebene: das *Treeness Principle*, das *Accepted Edge Labels Principle* und das *Valency Principle*. Das *Treeness Principle* garantiert einfach, dass nur Dependenzbäume und keine allgemeinen Dependenz-Graphen auf dieser Ebene erlaubt sind. Dieses Prinzip ist bei allen Grammatiken, die es nutzen, identisch und hängt nicht von irgendwelchen Parametern ab. Die anderen beiden Prinzipien legen die Dependenz-Relation mit Hilfe des Lexikons fest. Das Lernen der ID-Ebene aus einem Korpus besteht im Wesentlichen aus der Aufgabe, dieses Lexikon aufzubauen.
Das *Accepted Edge Labels Principle* definiert für jedes Wort, welche eingehenden Kanten für den Knoten eines Wortes erlaubt sind, und nutzt dazu den $Labels_{ID}$-Parameter. Zu jedem Wort kann eine Menge von gültigen eingehenden Kantenbeschriftungen angegeben werden. So könnte der Name "Maria" zum Beispiel in einem Satz als Subjekt, aber auch als Objekt vorkommen. Unter der Annahme, dass diese grammatikalischen Funktionen durch die Kantenbeschriftungen "Subj" und "Obj" repräsentiert werden, sollte das Lexikon also einen Eintrag zum Wort "Maria" enthalten, der den Wert $Labels_{ID} = \{Subj, Obj\}$ enthält.
Das *Valency Principle* definiert, welche ausgehenden Kanten für den Knoten eines Wortes erlaubt sind, und verwendet dazu den $Valency_{ID}$-Parameter. Die ausgehenden Kanten entsprechen genau den Valenzen, also den benötigten - oder auch optionalen - abhängigen Elementen eines Wortes. Für die Valenzen reicht es nicht aus, eine Menge der erlaubten ausgehenden Kantenlabel anzugeben, da nicht nur die Information benötigt wird, welche Kantenlabel erlaubt sind, sondern auch, ob sie zwingend benötigt werden oder optional sind und ob genau ein Element oder mehrere dieses Typs erlaubt sind. Deswegen wird zu jedem erlaubten Kantenlabel noch das Attribut *card* (für *Cardinality*) angegeben, das genau einen der folgenden Werte hat: *opt* für optionale ausgehende Kanten, die genau einmal oder gar nicht vorkommen dürfen, *one* für ausgehende Kanten, die genau einmal vorkommen müssen, *geone* für aus-

gehende Kanten, die mindestens einmal, aber auch öfter vorkommen können, und *any* für Kanten, die beliebig oft vorkommen, aber auch komplett fehlen dürfen. Im Lexikon ist ein Eintrag zu dem Wort "liebt" mit dem Wert $Valency_{ID} = \{(Subj, card = one), (Obj, card = one), (Adv, card = any)\}$ also so zu interpretieren, dass "liebt" auf jeden Fall genau ein von ihm abhängiges Subjekt und genau ein von ihm abhängiges Objekt benötigt und zusätzlich mit beliebig vielen Adverbien ausgeschmückt werden kann. Wie generell in der ID-Ebene ist auch hier zu beachten, dass die Reihenfolge - wie durch die Mengennotation angedeutet ist - keine Rolle spielt. Durch die Reihenfolge in der Schreibweise ist also nicht festgelegt, dass das Subjekt im Satz vor dem Objekt und dieses wiederum vor den Adverbien stehen muss.

1.4 Die LP-Ebene

Im Gegensatz zur ID-Ebene, die die einzelnen Komponenten eines Satzes nach ihren grammatikalischen Rollen unterscheidet, besteht die Absicht der LP-Ebene (*Linear Precedence*) darin, den Satz nach der *Theorie der topologischen Felder* [2] zu untergliedern. Die zentrale Idee dieser Theorie besteht darin, einen Satz in einzelne Felder zu unterteilen, wobei jedes Feld ein zusammenhängendes Teilstück des Satzes darstellt. Die Unterteilung in die einzelnen Felder soll Aufschluss darüber geben, an welcher Stelle innerhalb eines Satzes in der Regel Elemente mit bestimmten Funktionen liegen. Die Reihenfolge der Wörter innerhalb des Satzes spielt hier also eine zentrale Rolle. Die Theorie der topologischen Felder schlägt vor, innerhalb eines Satzes folgende Felder zu unterscheiden: das Vorfeld, die linke Satzklammer, das Mittelfeld, die rechte Satzklammer und das Nachfeld. Dabei befindet sich das konjugierte Verb immer in der linken Satzklammer und andere von ihm abhängige Verben befinden sich immer in der rechten Satzklammer. Alle anderen Teile des Satzes befinden sich dementsprechend je nach der Reihenfolge der Wörter im Satz im Vor-, Mittel- oder Nachfeld. Innerhalb eines Aussagesatzes befindet sich zum Beispiel das Subjekt in der Regel im Vorfeld und das Objekt im Mittelfeld. In einem Fragesatz befindet sich das Subjekt dagegen im Mittelfeld, wie das folgende Beispiel zeigt:

Vorfeld	(	Mittelfeld	)	Nachfeld
Maria	hat	Peter	geliebt	
	Hat	Maria Peter	geliebt ?	

Das Nachfeld enthält in der Regel Nebensätze:

Vorfeld	(	Mittelfeld	)	Nachfeld
Maria	hat	Peter	geliebt	,als sie ihn heiratete

Doch nicht nur vollständige Sätze lassen sich in einzelne Felder aufspalten, auch einzelne Segmente des Satzes können wiederum in detailliertere Felder auf-

gespalten werden. So könnte man den Teil eines Satzes, der sich auf ein bestimmtes Nomen bezieht, wie zum Beispiel den Ausdruck "... das alte, baufällige Haus, das in der Parkstrae steht,...", ähnlich untergliedern. Eine geeignete Einteilung in Felder kann Regelmäigkeiten in der Stellung der Wörter ausnutzen, wie etwa die Tatsache, dass der Artikel des Nomens grundsätzlich vor ergänzenden Adjektiven steht oder ergänzende Nebensätze immer nach dem Nomen stehen, auf das sie sich beziehen.
Genau wie die ID-Ebene nutzt die LP-Ebene das *Treeness Principle*, um sicherzustellen, dass nur echte Baumstrukuren als Dependenzbaum erlaubt sind. Analog zur ID-Ebene gibt es ebenfalls ein *Accepted Edge Labels Principle* und ein *Valency Principle*, die die gültigen eingehenden und ausgehenden Kanten der Knoten des Dependenzbaumes festlegen. Die eingehenden Kanten eines Wortes entsprechen in der LP-Ebene den Feldern, in denen das Wort untergebracht werden darf, und die ausgehenden Kanten entsprechen den Feldern, die das Wort seinerseits für die Unterbringung abhängiger Wörter zur Verfügung stellt. Im Lexikon werden diese Angaben analog zur ID-Ebene über die Parameter $Labels_{LP}$ und $Valency_{LP}$ angegeben.
Ein Beispiel: Das Wort "Maria" kann als Subjekt eines Aussagesatzes im Vorfeld (abgekürzt durch *vf*) und als Subjekt eines Fragesatzes im Mittelfeld (*mf*) untergebracht werden. Selbst bietet es Felder für einen Artikel, ergänzende Adjektive und nachstehende Nebensätze, die wir mit *nvf* (nominales Vorfeld), *nmf* (nominales Mittelfeld) und *nxf* (nominales extraponiertes Feld) bezeichnen. Das Lexikon enthält also einen Eintrag mit den Werten $Labels_{LP} = \{vf, mf\}$ und $Valency_{LP} = \{(nvf, card = opt), (nmf, card = any), (nxf, card = any)\}$. Die Kardinalitätsangaben des $Valency_{LP}$-Parameters haben die gleiche Funktion wie innerhalb der ID-Ebene und sagen in diesem Fall aus, dass im nominalen Vorfeld genau ein oder gar kein Element untergebracht werden darf, während in den anderen Feldern beliebig viele Elemente stehen dürfen. Die Einschränkung des Vorfeldes ist hier deshalb sinnvoll, weil dieses Feld für den Artikel vorgesehen ist und sich grundsätzlich maximal ein Artikel auf ein Nomen bezieht. Adjektive im nominalen Mittelfeld kann es dagegen durchaus mehrere geben.
Da die Dependenz-Bäume der LP-Ebene geordnete sind, reichen die bisher genannten Prinzipien noch nicht aus, um die LP-Ebene ausreichend zu charakterisieren. Wie in der ID-Ebene legt das *Valency Principle* auch in der LP-Ebene noch keine Ordnung zwischen den ausgehenden Kanten (in dem Fall also den durch sie repräsentierten Feldern) fest. Diese Ordnung wird durch das *Order-Principle* festgelegt. Doch nicht nur die ausgehenden Kanten eines Knotens werden in die Ordnung einbezogen, sondern auch der Knoten selbst. Dazu bekommt jedes Wort über den zusätzlichen Parameter $Labels_N$ eine Menge möglicher Knotenbeschriftungen zugeordnet. Der Knoten eines Nomens erhält zum Beispiel in der Regel die Beschriftung "n". Um nun auszudrücken, dass das nominale Vorfeld (*nvf*) vor dem nominalen Mittelfeld (*nmf*) und dieses vor dem Nomen selbst steht, muss für die Ordnung gelten $nvf < nmf < n$. Man beachte, dass es sich bei *nvf* und *nmf* um Kantenbeschriftungen und bei n um eine Knotenbeschriftung handelt. Für das *Order Principle* muss jede Grammatik also eine totale Ordnung über alle erlaubten Kanten- und Knotenbeschriftungen in der

LP-Ebene enthalten. Im Prinzip würde auch eine partielle Ordnung ausreichen, die es lediglich erlaubt, die Felder miteinander zu vergleichen, die innerhalb der Grammatik unterhalb eines gemeinsamen Elternknotens vorkommen. Der Einfachheit halber und aus Effizienzgründen, vor allem in Kombination mit lexikalischer Ambiguität,erwartet das *Order Principle* allerdings eine totale Ordnung, die man aus der partiellen Ordnungen leicht erzeugen kann: Der partiellen Ordnung $vf < v12 < mf < vcf < nf$ und $nvf < nmf < n < nxf$ entspricht die totale Ordnung $vf < v12 < mf < vcf < nf < nvf < nmf < n < nxf$ oder auch $nvf < nmf < n < nxf < vf < v12 < mf < vcf < nf$. Welche der beiden Ordnungen man wählt, spielt keine Rolle, da ja nie eines der ersten fünf Felder mit einem der virt anderen Felder verglichen wird, da es keine Wörter gibt, die zum Beispiel gleichzeitig ein nominales Vorfeld und ein Vorfeld zur Verfügung stellen.

Während das *Order Principle* garantiert, dass sich die Felder unterhalb eines Knotens im Dependenzbaum in der korrekten Reihenfolge befinden, stellt es noch nicht sicher, dass es sich bei den Feldern um zusammenhängende Fragmente des Satzes handelt. So wäre es mit den bisher genannten Prinzipien möglich, dass sich ein Nomen im Mittelfeld befindet, der dazugehörige Artikel allerdings im Vorfeld untergebracht ist. Um solche Lösungen für die Analyse des Satzes auszuschlieen, benötigt man noch das *Projectivity Principle*, das keine weiteren Parameter von der Grammatik benötigt.

Die Struktur der Bäume der LP-Ebene ist nicht völlig unabhängig von der Struktur der ID-Bäume. Das *Climbing Principle* und das *Subtreeness Principle* legen im Prinzip fest, dass die Struktur der LP-Bäume immer eine abgeflachte Version der Bäume der ID-Ebene darstellt. Das *Climbing Principle* legt dabei fest, dass Knoten im Baum entlang der Kanten nach oben wandern dürfen. Das *Subtreeness Principle* fordert, dass der komplette Teilbaum unterhalb des Knotens mit nach oben klettert. Durch ein *Barriers Principle* kann zusätzlich dieser Klettervorgang begrenzt werden, indem es Knoten mit bestimmten grammatikalischen Funktionen explizit verboten wird, bestimmte Knoten zu "überklettern". Das genaue Verhalten des *Barriers Principle* wird über das Lexikon definiert; auf eine genaue Beschreibung wird hier verzichtet, da das *Barriers Principle* in der aktuellen Version der in diesem Bericht vorgestellten Grammatik noch nicht verwendet wird.

2 Das Felder-Lernsystem

Das Lernen einer Grammatik für den XDG-Formalismus mit den in den vorigen Abschnitten beschriebenen Prinzipien lässt sich in zwei gröere Aufgabenbereiche untergliedern: Erstens die Generierung eines Lexikonabschnitts, der die $Labels_{ID}$ und die $Valency_{ID}$ Werte für die ID-Ebene enthält, und zweitens die Erzeugung eines Lexikonabschnitts mit den $Labels_{LP}$-, $Valency_{LP}$- und $Labels_{N}$-Werten für die LP-Ebene und der damit verbundenen totalen Ordnung über die Knoten- und Kantenbeschriftungen. Bei dem Negra-Korpus, mit dessen Hilfe in diesem Praktikum die Grammatik gelernt werden sollte, handelt es sich um einen Korpus, der die Satzstrukturen der enthaltenen Sätze in Form von Bäumen repräsentiert. Diese Bäume entsprechen in ihrem Aufbau aber weder genau den Bäumen der ID-Ebene noch den Bäumen der LP-Ebene. Das in [3] dargestellte Felderlernsystem bietet die Möglichkeit, die Sätze des Negra-Korpus in Baumstrukturen umzuwandeln, wie sie in der ID-Ebene vorkommen, und auf Basis dieser umgewandelten Bäume mögliche Felder zu identifizieren, die als Basis für die Generierung der LP-Ebene genutzt werden können.

2.1 Korpus-Transformation

Die Umwandlung der Bäume des Negra-Korpus in Dependenzbäume, wie sie in der ID-Ebene vorkommen, stellt innerhalb des Lernsystems nur einen vorbereitenden Schritt für den eigentlichen Lernvorgang dar. Für die Erzeugung des ID-Teils der Grammatik können diese Zwischenergebnisse allerdings schon direkt genutzt werden. Die im aufwendigen darauf folgenden Lernvorgang identifizierten Felder können dann schlielich zur Erzeugung der LP-Ebene genutzt werden.
Der zentrale Unterschied zwischen den Bäumen des Negra-Korpus und den Dependenzbäumen, in die sie umgewandelt werden sollen, besteht darin, dass in den Negra-Bäumen jedes Wort des Satzes einem Blatt entspricht und die inneren Knoten deren Abhängigkeiten repräsentieren, während im Dependenzbaum eine Eins-zu-eins-Beziehung zwischen Wörtern und Knoten existiert, also auch innere Knoten genau einem Wort entsprechen. Um einen Negra-Baum in einen Dependenz-Baum umzuwandeln, muss man also alle inneren Knoten des Negra-Baumes eliminieren. Der Algorithmus, der die Unwandlung durchführt, lässt rekursiv bestimmte Wörter von den Blättern in höher gelegene Knoten wandern, bis die gewünschte Eins-zu-eins-Beziehung erreicht ist. Welche Wörter dabei innerhalb des Baumes aufsteigen, ist von den Kantenbeschriftungen abhängig: Unter mehreren Geschwister-Knoten steigt jeweils das Wort auf, dessen grammatikalische Funktion erkennen lässt, dass die anderen Geschwister von ihm abhängig sind.

2.2 Identifizierung der Felder

Die Suche nach einer geeigneten Unterteilung von Sätzen in verschiedene Felder ist unmittelbar mit der Aufgabe verbunden, gewisse Regelmäigkeiten in der Reihenfolge der Wörter innerhalb eines Satzes ausfindig zu machen. Wenn man zum Beispiel festgestellt hat, dass generell der auf ein Nomen bezogene Artikel vor ergänzenden Adjektiven steht und diese wiederum vor dem Nomen selbst, dann kann man daraus eine Felderstruktur für ein Nomen ableiten, die ein erstes Feld für den Artikel und ein darauf folgendes Feld für Adjektive zur Verfügung stellt. In manchen Fällen lässt sich keine eindeutige Reihenfolge feststellen: Eine adverbiale Ergänzung kann vor oder hinter einem Objekt stehen, wie die unterschiedliche Satzstellung bei “Ich mache Urlaub am Meer” bzw. “Ich mache am Meer Urlaub” zeigt. In so einem Fall kann man ein gemeinsames Feld vorsehen, in dem beide Teile des Satzes zusammen untergebracht werden können, so dass

die genaue Reihenfolge nicht festgelegt ist.
Das Lernsystem geht zum Lernen der Felder von einer zentralen Annahme aus: Die Reihenfolge von zwei Wörtern, die sich im Dependenzbaum unterhalb eines gemeinsamen Elternknotens befinden, hängt ausschlielich von den beiden Wörtern selbst und dem gemeinsamen Elternknoten ab. Da der Rest eines Baumes laut Annahme keine Rolle spielt, werden für den Lernvorgang auch nur solche Baumfragmente betrachtet. Alle Bäume des Korpus, auf dessen Basis die Felder generiert werden sollen, werden also in einem ersten Schritt in Tripel aufgespaltet, die jeweils aus einem Elternknoten und zwei seiner Töchterknoten bestehen. Dabei stehen in dem Tripel nicht die Wörter selbst, die den einzelnen Knoten entsprechen, sondern nur bestimmte Merkmale der Wörter, so genannte *Features*. Innerhalb des Negra-Korpus sind zu jedem Wort neben der grammatischen Funktion auch diverse Eigenschaften annotiert, wie etwa Wortart, Kasus, Genus, Numerus und ähnliches. Alle diese Werte sind potentielle *Features*; welche von ihnen in die Tripel übernommen werden, entscheidet ein so genannter *Feature-Selector*. Dieser *Feature-Selector* hat also im Prinzip die Aufgabe, von den konkreten Wörtern zu abstrahieren und sie auf ihre wesentlichen Merkmale zu reduzieren und somit unterschiedliche Wörter zu verschiedenen Wortklassen zusammenzufassen. Die richtige Entscheidung des *Feature-Selectors*, welche *Features* er für ein Wort auswählt, sind also entscheidend für den Erfolg des Lernvorgangs. Aus dem Satzfragment "Das alte Haus" entsteht zum Beispiel das Tripel *("Haus", "Das", "alte")*, das der *Feature-Selector* zu *((Nomen), (Artikel), (Adjektiv))*, aber auch zu *((Nomen, Singular, Neutrum), (Singular), (Singular))* reduzieren kann. Aus dem ersten Tripel kann man die sehr sinnvolle Regel ableiten, dass unterhalb eines Nomens der Artikel vor dem Adjektiv steht. Das zweite Tripel würde dagegen besagen, dass unterhalb eines Nomens im Singular mit dem Genus Neutrum ein Wort im Singular immer vor einem Wort im Singular steht, was offensichtlich kein sehr vernünftiges Ergebnis ist.
Insgesamt verfügt das System über zwei Alternativen bezüglich des *Feature-Selectors*: Den manuellen *Feature-Selector*, in dem zu jeder möglichen Zusammenstellung von *Features* fest vorgegeben ist, welche davon ausgewählt werden sollen und den automatischen *Feature-Selector*, der einen *Decision-Tree-Learner* verwendet, um seine Entscheidung zu treffen. Des Weiteren wird zwischen dem *Feature-Selector* für den Elternknoten (also das erste Element der Tripel) und dem *Feature-Selector* für die Töchterknoten (also die beiden anderen Elemente der Tripel) unterschieden. Zur Unterscheidung werden sie als *Head-Feature-Selector* beziehungsweise *Dependent-Feature-Selector* bezeichnet. Der automatische *Feature-Selector* steht nur für den letzteren zur Verfügung. Auerdem muss zum *Dependent-Feature-Selector* erwähnt werden, dass dieser vom jeweiligen Elternknoten abhängig ist. Ein und dasselbe Wort kann also, wenn es sich unterhalb verschiedener Elternknoten befindet, durch den *Dependent-Feature-Selector* unterschiedlichen Wortklassen zugeordnet werden.
Um aus den einzelnen Tripeln eine allgemeine Ordnung abzuleiten, die letztlich eine Felderstruktur festlegt, werden jeweils alle Tripel gemeinsam betrachtet, die einen identischen Elterknoten enthalten. Diese Partitionierung der Gesamtmenge aller Tripel dient dazu, für jeden Elternknoten-Typ getrennt eine Felder-

struktur ableiten zu können. Zu jeder Partition wird im nächsten Schritt ein *Order Graph* erstellt. Dieser fasst alle Ordnungen zwischen den einzelnen Wortklassen unterhalb des jeweiligen Elternknotens zusammen. Jeder Knoten im Graph ist mit einer Wortklasse beschriftet, die gerichteten Kanten repräsentieren eine in den Beispiel-Tripeln beobachtete Reihenfolge. Das Tripel *((Nomen), (Artikel), (Adjektiv))* würde zum Beispiel dafür sprechen, dass im *Order Graph* zu der Wortklasse *Nomen* eine Kante vom Knoten *Artikel* zum Knoten *Adjektiv* eingefügt wird. Um eine gewisse Robustheit gegenüber fehlerhaften Tripeln zu erreichen, die aus Sätzen mit falscher oder sehr ungewöhnlicher Wortstellung resultieren, wird allerdings nicht jede Kante direkt auf Basis eines einzelnen Tripels generiert. Welche Kanten in den *Order Graph* übernommen werden, entscheidet eine *Edge Selection Function.* Diese lässt solche Kanten weg, die nur durch sehr wenige Tripel prognostiziert werden - also seltene Ausnahmefälle darstellen - und solche, die sich gegenseitig wiedersprechen. Wenn also gleichzeitig die Reihenfolge Artikel vor Adjektiv und umgekehrt vorkommen würde, würde die *Edge Selection Function* nicht beide, sondern keine der beiden Kanten in den Graph einfügen.
Im Ordergraph entsprechen die einzelnen Felder genau den *Strongly Connected Components*, also den Teilgraphen, innerhalb derer jeder Knoten von jedem anderen aus über die gerichteten Kanten erreichbar ist. Der Grund dafür liegt auf der Hand: Wenn zwei Knoten a und b vom jeweils anderen aus im Graph erreichbar sind, bedeutet dies, dass im Satz sowohl a vor b als auch b vor a stehen kann. Somit sollten sie in ein gemeinsames Feld zusammengefasst werden.
Die *Strongly Connected Components* können in linearer Zeit berechnet werden; was der Algorithmus natürlich nicht leisten kann, ist den so identifizierten Feldern einen aussagekräftigen Namen wie "Vorfeld" oder "nominales Mittelfeld" zuzuordnen. Statt dessen werden die einzelnen Felder einfach durchnummeriert.

3 Erweiterungungen des Lernsystems

In diesem Abschnitt werden alle Erweiterungen dargestellt, die im Rahmen dieses Praktikums an dem Felder-Lernsystem vorgenommen wurden. Ziel der Erweiterungen war es, das System um die Funktionalität zu erweitern, direkt an Hand einer Baumdatenbank eine funktionsfähige Grammatik für den XDG-Parser zu generieren. Die Grammatik sollte aus einer ID-Ebene und einer LP-Ebene bestehen, nach dem Vorbild der Grammatiken für das TDG-System.

3.1 Export der ID-Ebene

Die Generierung der ID-Ebene beschränkt sich im Wesentlichen auf die Erzeugung der $\mathit{Labels}_{ID}-$ und $\mathit{Valency}_{ID}-$Einträge zu jedem Wort des Lexikons. Das Felder-Lernsystem bietet ja bereits die Funktionalität, die Baumstrukturen des Negra-Korpus, der als Eingabe dient, in Dependenzbäume, wie sie in der ID-Ebene vorkommen sollen, umzuwandeln (siehe Abschnitt 2.1). Aus diesen ID-Bäumen kann man die Labels_{ID}-Werte, also die Kantenbeschriftungen der eingehenden Kanten zu jedem Knoten, und die $Valency_{ID}$-Werte, also die Beschriftungen der ausgehenden Kanten zu jedem Knoten und deren Anzahl, mehr oder weniger direkt ablesen.

Da die Negra-Bäume zu jedem Wort verschiedenste Informationen enthalten, muss festgelegt werden, welche davon für die Kantenbeschriftung genutzt werden. Diese Aufgabe kann analog zur Aufgabe des *Feature-Selectors* aus Abschnitt 2.2 betrachtet werden. Für die Kantenbeschriftungen der ID-Ebene wird grundsätzlich ein Tupel aus dem *POS-Tag* und dem *REL-Tag* verwendet, was im Prinzip einem einfachen, manuellen *Feature-Selector* entspricht. Das *POS-Tag* entspricht der Wortart eines Wortes, wobei diese relativ genau aufgeschlüsselt ist und zum Beispiel zwischen attributivem Adjektiv (abgekürzt durch das Tag "ADJA") und adverbialem oder prädikativem Adjektiv ("ADJD") unterschieden wird. Das *REL-Tag* entspricht dagegen der syntaktischen Funktion und kann Werte wie "AG" für "Genitiv-Attribut" oder "SB" für "Subjekt" aufweisen. Diese beiden Tags wurden ausgewählt, da sie die wesentlichen Eigenschaften beschreiben, die ein Wort auf der Dependenz-Ebene klassifizieren. Prinzipiell könnte man auch noch mehr Attribute für die Kantenbeschriftung heranziehen: Zum Beispiel ein Dreitupel, das zusätzlich noch das Genus eines Wortes enthält, sofern es sich um ein Wort mit Genus handelt. So könnte auch ausgedrückt werden, dass sich auf ein Nomen nur Artikel beziehen dürfen, die das gleiche Genus haben wie das Nomen selbst. Ähnlich könnte man auch eine Kasus- und Numeruskongruenz fordern. Dies würde allerdings zu einer Explosion der Anzahl möglicher Kantenbeschriftungen führen. Um den damit verbundenen Speicher- und Rechenaufwand in erträglichen Grenzen zu halten, wurde auf diese möglichen Erweiterungen verzichtet.

Das XDG-System erlaubt es, Grammatiken hierarchisch aufzubauen, indem man bestimmte Wortklassen definiert und von ihnen dann andere Wortklassen oder konkrete Wörter ableitet. Von einer Klasse "Nomen" könnte also eine Klasse "Eigenname" und von dieser das Wort "Maria" abgeleitet werden. So müssen nicht alle Parameter wie $Labels_{ID}$ und $Valency_{ID}$ direkt in dem Lexikon-Eintrag von "Maria" definiert werden, sondern können von einer Klasse geerbt werden. Die Nutzung dieser Strukturierung macht die Grammatik zwar übersichtlicher, ändert an der internen Darstellung der Daten innerhalb des XDG-Parsers jedoch nichts und spielt für die Effizienz des Parsers deshalb keine Rolle. Um die Erzeugung der Grammatik nicht unnötig kompliziert zu gestalten, wurde bei der ID-Ebene deshalb auf eine hierarchische Ordnung vollkommen verzichtet.

Der Algorithmus gestaltet sich dementsprechend einfach: Jedes Wort in jedem Baum wird unabhängig vom kompletten Rest des Korpus betrachtet und zu jedem Auftreten jedes Wortes wird ein eigener Lexikoneintrag erzeugt. Dabei entspricht der $Labels_{ID}$-Parameter des Lexikoneintrags genau einem Tupel aus dem *POS-Tag* und dem *REL-Tag* des Wortes. Der $Valency_{ID}$-Parameter setzt sich aus einer Menge von Tupeln der *POS-Tags* und *REL-Tags* der Töchterknoten zusammen. Kommt ein solches Tupel bei genau einem Tocherknoten vor, so bekommt es die Kardinalität "one", kommt ein Tupel gleichzeitig bei mehreren Töchterknoten vor, so wird seine Kardinalität auf "geone" ("greater or equal one") gesetzt. In Abbildung 2 ist das Fragment eines ID-Baumes zu dem

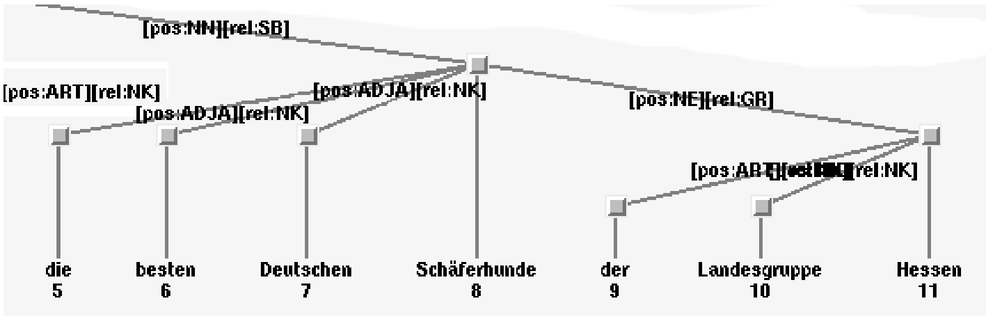

Abbildung 2: Fragment eines Beispiel-Dependenzbaums der ID-Ebene

Satzstück “... die besten Deutschen Schäferhunde der Landesgruppe Hessen ...” dargestellt. Basierend auf diesem Baumfragment würde zum Beispiel zu dem Wort “Schäferhunde” ein Lexikoneintrag mit den Werten $Labels_{ID} = \{[pos : NN][rel : SB]\}$ und $Valency_{ID} = \{([pos : ART][rel : NK], card = one), ([pos : ADJA][rel : NK], card = geone), ([pos : NE][rel : GR], card = one)\}$ generiert. Die Kardinalität des Elementes zu *[pos:ADJA][rel:NK]* in der $Valency_{ID}$-Menge hat eine Kardinalität von *geone*, weil sowohl die ausgehende Kante zu dem Wort “besten” als auch die zu “Deutschen” mit diesem Label beschriftet sind.

Da zu jedem Wortvorkommen im Korpus exakt ein Lexikoneintrag generiert wird, werden zu Wörtern, die mehrfach im Korpus vorkommen, auch mehrere Lexikoneinträge erzeugt. Dieser Umstand stellt jedoch kein Problem dar, da der XDG-Parser, der die fertige Grammatik einliet und in ein internes Format kompiliert, über die Funktionalität verfügt, mehrere Einträge, die sich auf das gleiche Wort beziehen, zusammenzufassen. Absolut identische Einträge werden dabei auf einen Eintrag reduziert; gibt es zu einem Wort mehrere sich unterscheidende Einträge innerhalb der Grammatik, so werden zu dem Wort mehrere $Valency_{ID}$- und $Labels_{ID}$-Mengen getrennt gespeichert und nicht etwa deren Vereinigung gebildet. Eine solche Vereinigung würde den Speicherbedarf des Lexikons zwar erheblich reduzieren, würde aber gleichzeitig eine viel zu starke Verallgemeinerung bedeuten, da sämtliche Interdependenzen zwischen den einzelnen Valenzen ignoriert würden. So würde die Information darüber verloren gehen, dass bestimmte ausgehende Kanten nur in Kombination mit bestimmten anderen ausgehenden Kanten auftreten oder voraussetzen, dass bestimmte andere Kanten fehlen.

3.2 Export der LP-Ebene

Der Export der LP-Ebene gestaltet sich etwas komplexer als der Export der ID-Ebene, obwohl statt der $Labels_{ID}$- und $Valency_{ID}$-Mengen nur die $Labels_{LP}$- und $Valency_{LP}$-Mengen sowie zusätzlich die $Labels_N$-Mengen und die totale Ordnung über die Kanten- und Knotenbeschriftungen für das *Order Principle*

generiert werden müssen. Den Ausgangspunkt für den Export der LP-Ebene stellen die topologischen Felder dar, die im Laufe des Lernvorgangs generiert wurden. Innerhalb des Lernvorgangs wurden über einen *Head-Feature-Selector* die einzelnen Wörter zu bestimmten Elternklassen zusammengefasst und zu jeder dieser Klassen eine Folge von Feldern generiert. In jedem dieser Felder können Wörter bestimmter Tochterklassen untergebracht werden. Zu welcher Tochterklasse ein Wort gehört, entscheidet ein *Dependent-Feature-Selector*. Da die Grammatiken des XDG-Systems eine hierarchische Gliederung erlauben, wird jede dieser Eltern- und Tochterklassen innerhalb der Grammatik auch direkt durch eine Klasse repräsentiert. Jedes Wort des Lexikons erbt dann die Eigenschaften von den Eltern- und Tochterklassen, zu denen es durch die *Head-* und *Dependent-Feature-Selectoren* reduziert wird.
Die Elternklassen müssen die Information darüber enthalten, welche Felder (bzw. ausgehenden Kanten) ein Wort dieser Klasse für die von ihm abhängigen Wörter zur Verfügung stellt. Dafür wird der $Valency_{LP}$-Parameter verwendet, der aus einer Menge der Namen dieser Felder besteht. Über den $Labels_N$-Parameter wird zusätzlich die Knotenbeschriftung des Elternknotens selbst angegeben, die für die Ordnung zwischen dem Eltern-Knoten und seinen Feldern benötigt wird. Innerhalb des Lernvorgangs wird der Eltern-Knoten genauso behandelt wie die ihm untergeordneten Wörter. Demnach wird auch ein Feld generiert, in dem der Elternknoten positioniert ist. Der Name dieses Feldes wird demnach als $Labels_N$-Wert gewählt. Eine mögliche Elternklassen-Bezeichnung ist zum Beispiel "[pos:NN]", zu der Wörter reduziert werden, deren *POS-Tag* den Wert *NN* hat. Wenn das Lernsystem zu dem Ergebnis gekommen ist, dass Wörter dieser Klasse neun verschiedene Felder anbieten, dann werden diese mit "[pos:NN]0",..., "[pos:NN]8" bezeichnet. Demnach enthält die Definition der Klasse den Wert $Valency_{LP} = \{([pos : NN]0, card = any), ..., ([pos : NN]8, card = any)\}$ und $Labels_N$ enthält die Bezeichnung des Feldes, in dem der Elternknoten untergebracht ist, also zum Beispiel $[pos : NN]4$. Die Kardinalitäten wurden in einem ersten einfachen Versuch, wie im Beispiel angedeutet, alle auf den allgemeinsten Wert "any" gesetzt. Später wurde das System an dieser Stelle noch verbessert, indem die Kardinalitäten zu jedem Feld jeder Klasse genau berechnet wurden. Da diese Berechnung etwas komplexer ist, wird sie separat im Abschnitt 3.3 beschrieben.
Die Töchterklassen legen fest, innerhalb welcher Felder ein Wort der jeweiligen Tochterklasse untergebracht werden darf. Dies wird direkt über den Parameter $Labels_{ID}$ angegeben. Die Generierung der Namen der Tochterklassen durch den *Dependent-Feature-Selector* erfolgt analog zum Verfahren bei den Elternklassen durch Aneinanderreihung der relevanten Features. Da diese relevanten Features bei Eltern- und Töchter-Knoten durchaus identisch sein können, ist diese Namensgebung nicht eindeutig, da zum Beispiel sowohl eine Elternklasse als auch eine Tochterklasse mit dem Namen "[pos:NN]" existieren kann. Deshalb bekommen alle Töchterklassen den Präfix dependent vorangestellt.
Da die Klassen bereits alle zu einem Lexikoneintrag nötigen Parameter enthalten, beschränkt sich die Erzeugung der Einträge der einzelnen Wörter darauf, die Klassen anzugeben, von denen das entsprechende Wort diese Parameter er-

ben soll. Dazu wird wie für die Generierung der ID-Ebene einmal die komplette Baumdatenbank durchlaufen und jedes vorkommende Wort getrennt betrachtet. Jedes Wort erbt dann die Eigenschaften von einer Elternklasse und einer oder mehrerer Töchterklassen. Um festzustellen, von welcher Elternklasse geerbt werden soll, wird der *Head-Feature-Selector* genutzt. Dieser reduziert das Wort zu genau einer Elternklasse, von der das Wort dann die $Valency_{LP}$- und $Labels_N$-Werte erbt. Zur Feststellung der geeigneten Tochterklasse kann man analog dazu den *Dependent-Feature-Selector* verwenden. Hier stellt sich nur das Problem, dass dieser vom Elternknoten abhängt, der nicht eindeutig feststeht. Da der durchlaufene Korpus aus ID-Bäumen besteht und diese sich in ihrer Struktur von den LP-Bäumen unterscheiden können, kann nicht davon ausgegangen werden, dass der Elternknoten im ID-Baum auch immer der Elternknoten im dazu passenden LP-Baum ist. Da keine passenden LP-Bäume zur Verfügung stehen, bleibt keine andere Möglichkeit, als alle potentiellen Elternknoten aus dem ID-Baum in Betracht zu ziehen. Der Unterschied zwischen den LP- und den ID-Bäumen besteht darin, dass Knoten des ID-Baumes im LP-Baum weiter nach oben geklettert sein können (siehe Abschnitt 1.4). Deshalb kommt als potentieller Elternknoten im LP-Baum jeder Knoten des ID-Baums in Frage, der auf dem Pfad von dem Knoten selbst zur Wurzel liegt. Der *Dependent-Feature-Selector* wird also für jeden dieser möglichen Elternknoten einmal aufgerufen und liefert jeweils eine Tochterklasse zurück. Der Lexikoneintrag erbt dann einfach von jeder dieser Tochterklassen.
Wie in der ID-Ebene kann es auch in der LP-Ebene vorkommen, dass zu einem Wort mehrere Lexikoneinträge generiert werden, sofern das Wort mehrfach im Korpus vorkommt. Genau wie in der ID-Ebene muss auch hier dieser Umstand nicht weiter berücksichtigt werden, da der Parser, der die Grammatik dann kompiliert, diese Einträge in geeigneter Weise zusammenfasst.

3.3 Berechnung der Kardinalitäten

Um die LP-Ebene der Grammatik noch etwas restriktiver und genauer zu gestalten, wurde das Felderlernsystem noch dahingehend erweitert, dass es zu den gelernten Feldern auch Kardinalitäten ermittelt. Die Kardinalität sagt aus, wie viele Elemente in dem jeweiligen Feld gleichzeitig untergebracht werden dürfen. Die Bestimmung der Kardinalitätswerte kann nicht in den Lernvorgang zur Bestimmung der Felder selbst integriert werden. Der Grund dafür ist die Tatsache, dass die Bestimmung der Felder auf Tripeln basiert, die jeweils aus einem Elternknoten und zwei seiner Töchterknoten bestehen, die in den Feldern des Elternknotens untergebracht werden sollen. Um festzustellen, wie viele Wörter innerhalb eines Feldes gleichzeitig vorkommen dürfen, reicht es nicht aus, sich immer nur zwei dieser Wörter anzusehen. Aus diesem Grund erfolgt die Bestimmung der Kardinalitäten separat, nachdem die Bestimmung der einzelnen Felder abgeschlossen ist.
Die Berechnung der Kardinalitäten baut auf einem bereits im Lernsystem vorhandenen Modul auf, das es ermöglicht, die ID-Bäume eines Korpus in LP-Bäume zu transformieren, die auf den erlernten Feldern basieren. Da es sich

bei dieser Transformation um einen einfachen Algorithmus handelt, der vollkommen autonom und ohne die Korrektur durch einen menschlichen Benutzer arbeitet, ist natürlich nicht sichergestellt, dass es sich bei den erzeugten LP-Bäumen um die linguistisch sinnvollsten zu den ID-Bäumen passenden LP-Bäume handelt. Der Algorithmus versucht lediglich, irgendeinen Baum zu finden, der allen LP-Prinzipien genügt und dessen Kanten mit den gelernten Feldnamen beschriftet sind. Das Verfahren dazu funktioniert wie folgt: In einem ersten Schritt klettern die einzelnen Knoten eines Baumes nach den Vorschriften des *Climbing Principles* solange nach oben, bis alle Kanten projektiv geworden sind, das heit, bis der Baum dem *Projectivity Principle* genügt. Danach werden die ID-Kantenbeschriftungen durch LP-Kantenbeschriftungen ersetzt. Da die Kantenbeschriftungen der LP-Ebene die Namen der topologischen Felder sind, entspricht das der Aufgabe, alle Knoten in passenden Feldern ihrer Elternknoten unterzubringen. Dabei wird eine *Greedy*-Strategie erfolgt, bei der jeder Knoten in dem jeweils ersten Feld untergebracht wird, in dem er untergebracht werden darf. Natürlich darf kein Knoten in einem Feld vor seinen Vorgängerknoten untergebracht werden. Wenn der n-te Knoten also im i-ten Feld untergebracht wurde, dann wird zunächst versucht, den (n+1)-ten Knoten auch im i-ten Feld unterzubringen. Wenn das auf Grund der Tochterklassen-Zugehörigkeit des Knotens nicht möglich ist, dann wird als nächstes das (i+1)-te , (i+2)-te, ... Feld in Betracht gezogen. Für den Fall, dass der Knoten in keinem der Felder untergebracht werden darf, wird er entgegen der gelernten Regeln im i-ten Feld untergebracht. Das führt dazu, dass es sich bei dem erzeugten Baum nicht mehr um einen gültigen LP-Baum handelt. Die so erzeugte Baumstruktur ist als Notlösung zu betrachten, die einem wirklichen LP-Baum möglichst nahe kommt und in den Fällen erzeugt wird, in denen es nicht einfach ist, einen korrekten LP-Baum zu konstruieren. Das bedeutet jedoch nicht, dass es zu dem entsprechenden ID-Baum keinen gültigen LP-Baum gibt. Durch geschickte zusätzliche Climbing-Vorgänge könnte es prinzipiell möglich sein, einen gültigen LP-Baum zu konstruieren. Verbesserungen des Algorithmus in dieser Hinsicht wären durchaus denkbar, wenn auch recht komplex.

Obwohl die so konstruierten LP-Bäume im ursprünglichen System nur der Bewertung der erzeugten Felderstrukturen dienten, lassen sie sich relativ direkt auch für die Berechnung der Kardinalitäten verwenden. Da fehlerhafte Baumstrukturen keine gute Basis für eine derartige Berechnung sind, werden nur die Bäume in Betracht gezogen, die korrekte LP-Bäume sind, also bei denen alle Wörter in gültigen Feldern untergebracht werden konnten. Zu jedem Feld werden nun drei Werte ermittelt: $Count_0$, $Count_1$ und $Count_{>1}$. Dabei gibt $Count_0$ die Anzahl der Fälle an, in denen gar kein Element in dem Feld untergebracht wurde; $Count_1$ und $Count_{>1}$ repräsentieren analog dazu die Anzahl der beobachteten Fälle, in denen genau ein Wort beziehungsweise mehr als ein Wort in dem Feld untergebracht war. Mit Hilfe dieser drei Werte wird nun versucht, dem Feld einen der Kardinalitätswerte *one*, *opt*, *geone* und *any* zuzuordnen. Die Kardinalitätswerte lassen sich nach ihrer Restriktivität ordnen, wobei die oben angegebene Reihenfolge genau einer absteigenden Restriktivität entspricht. Jedem Feld wird der Kardinalitätswert zugeordnet, der möglichst

restriktiv ist, und gleichzeitig durch die in $Count_0$, $Count_1$ und $Count_{>1}$ gespeicherten Werte gerechtfertigt wird. Gerechtfertigt bedeutet in dem Fall, dass es weniger als *thresh* Ausnahmen gibt, die nicht dem Kardinalitätswert entsprechen, wobei *thresh* ein einstellbarer Parameter ist, der sinnvoller Weise einen Wert zwischen 0 und 2 haben sollte. Damit zum Beispiel der Kardinalitätswert *one* gerechtfertigt ist, muss gelten $Count_0 + Count_{>1} \leq thresh$, da für diesen Kardinalitätswert das Feld weder leer bleiben, noch mehr als ein Element enthalten darf; für *opt* genügt dagegen die Bedingung $Count_{>1} \leq thresh$.
Damit verhindert wird, dass die Kardinalität von Feldern berechnet wird, für die zu wenige Beispiele gesehen wurden und für die somit keine vernünftige Prognose möglich ist, gibt es einen zweiten Schwellenwert, der die Anzahl der minimal gesehenen Vorkommen für dieses Feld festlegt. Felder, von denen zu wenige Vorkommen in den LP-Bäumen enthalten sind, erhalten grundsätzlich den am wenigsten restriktiven Kardinalitätswert *any*.
Die ermittelten Kardinalitätswerte werden, wie im Abschnitt 3.2 erwähnt wird, im $Valency_{LP}$-Parameter zu jedem Feld angegeben.

4 Auswertung

4.1 Einschränkungen durch Speicherprobleme

Um eine vernünftige Grammatik zu erhalten, braucht man einen relativ groen Korpus, der möglichst viele Phänomene der Sprache abdeckt und auf dessen Basis die topologischen Felder erlernt werden können. Eine an Hand von 9730 Sätzen des Negra-Korpus erlernte Grammatik ist in Form der vom System erzeugten XML-Datei bereits knapp 100MB gro. Der XDG-Parser versucht, diese Grammatikdatei einzulesen und in ein internes Format zu kompilieren. Dafür braucht er noch einmal ein Vielfaches dieses Speichers, so dass selbst der leistungsfähigste zur Verfügung stehende Rechner mit 1GB Arbeitsspeicher nicht ausreichte, um diese Aufgabe erfolgreich zu beenden. Deshalb wurde bei allen Tests so verfahren, dass zwei verschiedene Korpora genutzt wurden: Ein groer Lernkorpus mit 9.730 Sätzen, auf dessen Basis die Felderstruktur und die Einteilung in die einzelnen Wortklassen gelernt wurde, und ein kleiner Testkorpus, auf dessen Basis das Lexikon der ID- und LP-Ebene erzeugt wurde. Die so entstandene Grammatik deckt also nur einen relativ kleinen Wortschatz ab und verbraucht wenig Speicher, bietet den für den Lernerfolg zentralen Komponenten aber trotzdem eine ausreichende Zahl an Beispielsätzen. Möglicherweise sind die Probleme beim Kompilieren der Grammatiken auch nicht alleine auf Speicherprobleme, sondern auch auf einen unauffälligen Bug in der *Constraint-Engine* der zu Grunde liegenden Programmiersprache Mozart zurückzuführen. Eine Tatsache, die diese Vermutung stützt, ist, dass der XDG-Parser auch ab und zu beim Kompilieren kleinerer Grammatiken nicht terminiert, obwohl mehr als genug Speicher zur Verfügung steht. Versuche von Ralph Debusmann und Denys Duchier, den Fehler aufzuspüren, haben bis jetzt noch kein endgültiges Resultat erbracht. Es scheint jedoch relativ klar zu sein, dass die Ursache nicht in der Grammatik oder dem XDG-Parser, sondern innerhalb des *Constraint-Systems* liegt.

4.2 Testergebnisse

Mit Hilfe des XDG-Parsers wurde die Grammatik auf Testsätze angewendet, um zu überprüfen, ob der Parser sinnvolle Lösungen zu den Testsätzen findet. Dabei wurde, wie im Abschnitt 4.1 erwähnt wird, jeweils eine Grammatik erzeugt, die nur den Wortschatz dieser Testsätze abdeckt, um den Speicherbedarf in Grenzen zu halten. Da zu den Testsätzen keine korrekten LP-Baumsstrukturen als Vergleichsmaterial zur Verfügung stehen und es auch durchaus mehrere als sinnvoll geltende Lösungen zu einem Satz geben kann, ist es nicht ohne weiteres möglich, für die durch den Parser generierten Lösungen zu entscheiden, ob sie korrekt beziehungsweise sinnvoll sind. Deshalb wurden die Sätze nur nach folgenden drei Kriterien untersucht:

- Wie viele Lösungen wurden gefunden ?
- Wurde überhaupt eine Lösung gefunden ?
- Wie lange dauert der Parsvorgang ?

Wenn die erzeugte Grammatik nicht restriktiv genug ist, erlaubt sie auch nicht sinnvolle Lösungen und dadurch ergibt sich eine sehr hohe Gesamtzahl an möglichen Lösungen. Ist die Grammatik dagegen zu restriktiv, dann werden auch korrekte Lösungen durch sie teilweise nicht erlaubt und für viele Sätze wird gar keine Lösung gefunden. Das dritte Kriterium, die benötigte Zeit, spielt keine ganz so wichtige Rolle wie die ersten beiden. Wenn allerdings das Parsen eines Satzes mehrere Minuten benötigt oder gar nicht terminiert, spielt das durchaus eine relevante Rolle.
Für die Resultate ist die Einstellung diverser Parameter des Felderlernsystems ebenfalls ein entscheidender Faktor. Verschiedene getestete Parameterkombinationen wurden bei früheren Tests mit bestimmten Modellnamen bezeichnet. Da das Testen mit allen möglichen Parameterkombinationen den Rahmen gesprengt hätte, wurden nur die erfolgreichsten Modelle früherer Tests, FULL-D2 und etwas eingeschränkter auch MAN-E4, verwendet. Dabei liegt der wesentliche Unterschied der beiden darin, dass das FULL-D2-Modell den automatischen *Feature Selector* und das MAN-E4-Modell den manuellen *Feature Selector* verwendet.
Die insgesamt 495 verwendeten Testsätze stammen genau wie die Trainingssätze aus dem Negra-Korpus, es wurden aber Teile des Korpus verwendet, die nicht zum Lernen der Felderstruktur verwendet wurden. Da die Anzahl der Wörter in einem Satz wesentlich die untersuchten Faktoren beeinflusst, wurden die Testsätze in Gruppen unterschiedlicher Länge unterteilt. Dabei wurde zwischen Sätzen mit 1 bis 3 Wörtern, Sätzen mit 4 bis 10 Wörtern, Sätzen mit 11 bis 15 Wörtern und Sätzen mit mehr als 15 Wörtern unterschieden. Die Ergebnisse für das FULL-D2-Modell sind in Tabelle 1 dargestellt. Um zu überprüfen,

ob die Berechnung der Kardinalitäten zu einer Verbesserung führt, wurden die Tests zweimal durchgeführt: Einmal mit den berechneten Kardinalitätswerten und einmal ohne. Der Wert in einem Tabelleneintrag hinter dem Schrägstrich ist dabei immer der Wert des Durchgangs ohne die Kardinalitätswerte.

Tabelle 1: Testergebnisse des Parsers mit dem FULL-D2-Modell

Anzahl der Wörter des Satzes	Anteil am gesamten Korpus	Anzahl der Lösungen (Durchschnitt)	Sätze ohne Lösungen	Dauer (Sekunden)
1-3	12,52%	1,02 / 1,02	9,68% / 9,68%	0,78 / 0,78
4-10	23,84%	2,81 / 3,23	67,80% / 66,95%	7,36 / 6,76
11-15	21,82%	6,56 / 7,62	92,59% / 92,59%	19,99 / 15,16
>15	41,82%	nicht genug Speicher		

Bei den Sätzen mit 1 bis 3 Wörtern handelt es sich meist gar nicht um vollständige Sätze, sondern um Fragmente, die in dem Korpus vorkommen, da es sich bei den Korpus-Texten um Zeitungstexte handelt. Beispiele hierfür sind "Frankfurt A. M." "Spanisches Konsulat verwüstet" oder "11. Juni". Das Finden einer korrekten Lösung stellt sich dementsprechend einfach dar, so dass für die meisten Sätze das Optimum erreicht wird, nämlich das Generieren einer einzigen Lösung. Doch selbst hier scheitert der Parser bei fast zehn Prozent der Sätze bereits und findet gar keine Lösung. Je mehr Wörter ein Satz enthält, desto schlechter werden die erzielten Resultate. Die durchschnittliche Anzahl der Lösungen macht für sich alleine betrachtet gar keinen so schlechten Eindruck. Etwa 3 Lösungen bei Sätzen mit 4 bis 10 Wörtern beziehungsweise 7 Lösungen bei 11 bis 15 Wörtern sind keine schlechten Werte. Die Werte sind allerdings nur deshalb so niedrig, weil die Anzahl der Sätze ohne eine einzige Lösung dramatisch ansteigt. Bei den Sätzen mit 11 bis 15 Wörtern wird zu über 90 Prozent der Sätze gar keine Lösung gefunden. Wenn zu einem Satz dann doch Lösungen gefunden werden, sind es gleich sehr viele, so dass sich die im Vergleich dazu recht hohe Anzahl durchschnittlicher Lösungen ergibt. So haben - was sich aus der Tabelle nicht ergibt - bei den Sätzen mit 11 bis 15 Wörtern mehr als ein Drittel der Sätze, zu denen Lösungen gefunden werden, gleich mehr als 100 Lösungen. Zu den Sätzen mit mehr als 15 Wörtern werden sogar oft so viele Lösungen generiert, dass der Speicher nicht ausreicht und vernünftige Tests nicht möglich sind. Über die benötigte Zeit kann nur gesagt werden, dass sie sich in vernünftigen Grenzen bewegt, solange der Speicher ausreicht. Wenn die Anzahl der möglichen Lösungen gro wird, steigt gleichzeitig auch die benötigte Zeit, da alle möglichen Lösungen aufgezählt werden. Der begrenzende Faktor ist hier aber in der Tat nicht die Dauer der Berechnung, sondern der benötigte

Arbeitsspeicher. Zu den Auswirkungen der Kardinalitätsberechnung kann man feststellen, dass die dadurch bewirkten Änderungen minimal sind. Wie beabsichtigt sinkt die Anzahl der durchschnittlichen Lösungen, allerdings steigt die Anzahl der Sätze ohne Lösung auch minimal. Insgesamt sind die Auswirkungen der Kardinalitäten jedoch so gering, dass das schlechte Ergebnis durch sie nicht wirklich verbessert wird. Da das Hauptproblem die vielen Sätze ohne Lösung sind, kann auch nicht mit einer wesentlichen Verbesserung gerechnet werden, da die Kardinalitätsangaben die Menge möglicher Lösungen ausschlielich verkleinern können. Abschlieend ist zu den ermittelten Werten des FULL-D2 Modells noch zu erwähnen, dass die genauen Zahlenwerte mit Vorsicht zu betrachten sind. Während der Tests fiel ein gewisser Indeterminismus auf, der auf den automatischen *Feature Selector* zurückgeführt werden konnte. Der hier verwendete *Decision Tree Learner* C4.5 liefert Ergebnisse, die offenbar nicht deterministisch sind. Bei verschiedenen Lernvorgängen mit den gleichen Eingabedaten können bei Verwendung dieses *Feature-Selectors* unterschiedliche Topologische Felderstrukturen erlernt werden. Bei mehrfacher Wiederholung der Tests können sich also durchaus leicht abgewandelte Werte ergeben.
Zum FULL-D2 Modell wurde zusätzlich untersucht, wie effizient festgestellt wird, dass keine Lösung existiert. Hierzu wurde die Anzahl der *choice points* gezählt, die im Suchbaum auftreten, bis festgestellt wird, dass keine Lösung existiert. Wenigstens hier sind die Ergebnisse als positiv anzusehen: Von den insgesamt 186 Sätzen ohne Lösung war bei 183 gar keine Aufspaltung des Suchraumes notwendig, so dass sich keine *choice points* ergaben. Bei den drei Sätzen, die *choice points* benötigten, handelt es sich um Sätze mit 11 bis 15 Wörtern.
Die Ergebnisse des Tests mit dem MAN-E4 Modell sind in Tabelle 2 dargestellt.

Tabelle 2: Testergebnisse des Parsers mit dem MAN-E4-Modell

Länge der Sätze	Anzahl der Lösungen (Durchschnitt)	Sätze ohne Lösungen	Dauer (Sekunden)
4-10 Wörter	1,05 / 1,05	54,7% / 54,7%	1,94 / 2,20

Das MAN-E4-Modell wurde aus Zeitgründen nur mit Sätzen getestet, die 4 bis 10 Wörter enthalten. Hier schneidet es im Vergleich zum FULL-D2-Modell zwar etwas besser ab, insgesamt sind die Resultate aber immer noch als noch nicht brauchbar zu bewerten, da auch hier für über die Hälfte aller Sätze keine Lösung gefunden wird. Die Berechnung der Kardinalitäten beeinflusst das Ergebnis des MAN-E4 Modells überhaupt nicht, lediglich die benötigte Zeit geht bei Verwendung der Kardinalitätsangaben leicht zurück. Diese minimale Änderung kann aber durchaus auch auf externe Faktoren, wie die Belastung des Rechners durch andere laufende Prozesse während der Testdurchläufe, zurück-

zuführen sein.
Um die Ursache der schlechten Ergebnisse genauer einzugrenzen, wurde nach den direkten Tests der Grammatik mittels des XDG-Parsers noch ein weiterer Test durchgeführt, der ausschlielich die Qualität der erlernten Felderstrukturen bewerten sollte. Wie im Abschnitt 3.3 beschrieben wird, wird zum Berechnen der Kardinalitäten der einzelnen Felder versucht, aus den Sätzen des Korpus LP-Bäume zu konstruieren, die auf den gelernten Feldstrukturen basieren. Dabei kann es vorkommen, dass zu einem Wort kein passendes Feld gefunden wird, in dem es untergebracht werden kann. Dies kann sogar dann passieren, wenn der Satz ein Satz des Trainingskorpus ist, der zum Erlernen der Felderstrukturen verwendet wird. Das Lernsystem versucht, wie in Abschnitt 2.2 erklärt wird, die häufigste beziehungsweise wahrscheinlichste Wortstellung an Hand der Beispielsätze zu finden und nicht alle möglichen Wortstellungen. Selten beobachtete Wortstellungen werden daher als Ausnahmen betrachtet und ignoriert. Genau diese Ausnahmen können dann bei der oben erwähnten Konstruktion der LP-Bäume nicht korrekt in einem Feld untergebracht werden. Im Test wurde deshalb untersucht, wie viele Fehler bei dem Versuch auftreten, aus den Sätzen des Trainingskorpus LP-Bäume zu konstruieren. Der Traingskorpus besteht aus 9.727 Sätzen mit insgesamt 146.252 Wörtern. 3,07 Prozent dieser Wörter konnten nicht korrekt in einem der erlernten Felder untergebracht werden. Auf den ersten Blick scheinen diese 3,07 Prozent ein sehr guter Wert zu sein, da immerhin fast 97 Prozent der Wörter korrekt untergebracht werden können. Andererseits muss beachtet werden, dass etwa der XDG-Parser zu einem Satz nur dann eine Lösung berechnen kann, wenn alle Wörter des Satzes korrekt untergebracht werden können. Wenn man nun aber nicht die Anzahl der korrekt untergebrachten Wörter, sondern die Anzahl der fehlerfreien Sätze betrachtet, so erhält man das Ergebnis, dass nur 69,86 Prozent aller Bäume fehlerfrei sind. In fast jedem dritten Satz, den das System dazu verwendet, die Felderstrukturen zu erlernen, findet sich also mindestens ein Wort, das nicht in die generierten Felder passt. Der Grund dieser groen Differenz der Ergebnisse auf Wort- und Satzebene liegt darin, dass die nicht korrekt unterzubringenden Wörter sich sehr gleichmäig über die einzelnen Sätze verteilen: In 65,38 Prozent der fehlerhaften LP-Bäume konnte nur ein einziges Wort nicht untergebracht werden und nur 12 Prozent enthalten mehr als 2 Fehler. Dabei enthält jeder Satz durchschnittlich 15 Wörter.
Insgesamt scheint das Vorgehen des Lernsystems also nicht geeignet zu sein, um mit den gelernten Feldern eine Grammatik für den XDG-Parser zu generieren. Das System identifiziert zwar die am weitesten verbreitete Wortstellung zuverlässig, aber dennoch gibt es genug Ausnahmen, die dieser Wortstellung widersprechen, so dass fast jeder dritte Satz Ausnahmen enthält und nicht zu parsen ist. Andererseits erklärt das noch nicht vollständig, warum der XDG-Parser bei den meisten Tests weit mehr als 30 Prozent der Sätze nicht parsen konnte. Die Zahlen sind allerdings schwer direkt zu vergleichen, da im XDG-Parser fremde Sätze getestet wurden, die also nicht zum Lernen der Felderstrukturen verwendet wurden. Auerdem liegen zu den Sätzen mit mehr als 15 Wörtern bei den Parser-Tests gar keine Werte vor. Es ist durchaus möglich, dass es noch andere, nicht identifizierte Faktoren gibt, die die Resultate der Grammatik so

schlecht ausfallen lassen. Der letzte Test zeigt allerdings, dass eine auf dem Felderlernsystem basierende Grammatik nie mehr als etwa 70 Prozent Erfolgsquote haben kann. Die vom System generierten Felder stellen linguistisch interessante Informationen dar, sind aber als Basis für eine Grammatik zu ungenau.
Zusammenfassend kann man sagen, dass der erste Versuch, eine deutsche Grammatik für das XDG-System auf Basis einer Baumdatenbank zu generieren, nicht erfolgreich war. Auf der einen Seite müssen noch einige Probleme, die den vom Parser benötigten Speicher betreffen, gelöst werden, auf der anderen Seite muss eine bessere Möglichkeit gefunden werden, die topologischen Felder der LP-Ebene zu generieren. Ein denkbarer Ansatz, um die Speicherprobleme zu bewältigen, besteht darin, nicht alle möglichen Lösungen für die Analyse eines Satzes aufzuzählen, sondern nach bestimmten Kriterien nur eine der Lösungen zu generieren. Ein für die Generierung einer Grammatik besser geeignetes Felderlernsystem sollte auf jeden Fall so konzipiert sein, dass es Feldstrukturen erzeugt, die zumindest auf alle Sätze passen, auf denen der Lernvorgang basiert. Dazu wird es nötig sein, nicht nur die wahrscheinlichste Wortreihenfolge, sondern alle möglichen Reihenfolgen zu identifizieren. Von der zentralen Annahme, dass die Reihenfolge zweier Wörter nur von ihnen selbst und ihrem gemeinsamen Elternknoten abhängt, muss sehr wahrscheinlich auch Abstand genommen werden. Schon die Reihenfolge von Subjekt und Prädikat eines Satzes hängt beispielsweise auch davon ab, ob es sich um einen Aussagesatz oder einen Fragesatz handelt.

Literatur

[1] Ralph Debusmann. A declarative grammar formalism for dependency grammar. Master's thesis, University of Saarland, 2001.

[2] Tilman Höhle. Der Begriff "Mittelfeld", Anmerkungen über die Theorie der topologischen Felder. In Walter Weiss, Herbert Ernst Wiegand, and Marga Reis, editors, *Akten des 7. Internationalen Germanisten-Kongresses*, pages 329–340, Tübingen/GER, 1986. Max Niemeyer Verlag.

[3] Christian Korthals. Unsupervised learning of word order rules. Master's thesis, Saarland University, 2003.