

Humboldt-Universität zu Berlin  
Institut für Informatik



Studienarbeit

# **Evaluation von Constituent Parsern anhand von Dependency Graphen**

Stefan Pietschmann

pietschm@informatik.hu-berlin.de

25. April 2008

Betreuer: Prof. Dr. Ulf Leser

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Hintergrund und Motivation . . . . .	1
1.2	Ziel und Aufbau dieser Arbeit . . . . .	2
<b>2</b>	<b>Linguistische Grundlagen</b>	<b>3</b>
2.1	Constituent Trees . . . . .	3
2.2	Dependency Graphen . . . . .	4
<b>3</b>	<b>Vorgehen</b>	<b>6</b>
3.1	Ablauf und beteiligte Komponenten . . . . .	6
3.2	Treebank strippen . . . . .	7
3.3	Part Of Speech Tagging . . . . .	7
3.4	Parsing . . . . .	8
3.4.1	Bikel Parser . . . . .	9
3.4.2	Stanford Parser . . . . .	10
3.5	Transformation der Constituent Trees in Dependency Graphen . . . . .	11
3.5.1	Konsistenz schaffen . . . . .	11
3.5.2	Transformation . . . . .	12
<b>4</b>	<b>Evaluation</b>	<b>14</b>
4.1	Evaluation bezüglich genereller Genauigkeit . . . . .	15
4.2	Evaluation bezüglich bestimmter Wörter und Dependencies . . . . .	19
4.2.1	Evaluation bezüglich Interaktionswörtern . . . . .	19
4.2.2	Evaluation bezüglich Präpositionen . . . . .	20
4.3	Evaluation bezüglich Geschwindigkeit . . . . .	20
<b>5</b>	<b>Diskussion und Ausblick</b>	<b>21</b>
	<b>Literatur</b>	<b>22</b>

# 1 Einleitung

## 1.1 Hintergrund und Motivation

Die Menge an Publikationen im biomedizinischen Bereich wächst stetig und damit einhergehend auch der Bedarf an Verfahren, welche in der Lage sind, aus den dadurch entstehenden großen Fachtext-Korpora nur die Informationen zu liefern, die bestimmten Sachverhalten entsprechen. Dies sind oftmals Fragen nach bestimmten Wort-Wort-Beziehungen, wie beispielsweise Protein-Protein-Interaktionen, Protein-Funktions-Zusammenhänge oder Gen-Krankheits-Beziehungen.

Für solche Arten der Informationsextraktion (IE) wurden bisher vor allem Co-occurrence- [1][2][3] oder Pattern Matching-Verfahren [4][5][6][7] aus dem Natural Language Processing (NLP) eingesetzt. Co-occurrence-Verfahren gehen davon aus, dass eine Wort-Wort-Beziehung existiert, wenn sich beide Wörter gleichzeitig *im selben Satz* befinden. Pattern Matching-Verfahren basieren darauf, dass sich zwei Wörter nicht nur im selben Satz befinden, sondern dass sie außerdem gemeinsam in einem *Pattern* vorkommen müssen, welches extern (beispielsweise in einer Lexikon-Datenbank) gespeichert ist. In jüngerer Zeit versucht man zudem, Parsing-Verfahren einzubinden [8][9][10], welche die *grammatikalische Struktur* eines Satzes analysieren. Parsing-Verfahren sind in der Lage, auch verschachtelte Wort-Wort-Beziehungen zu finden, bei denen die beiden zuerst genannten Verfahren versagen. Diese arbeiten nur auf der Satzoberfläche, es findet also keinerlei Untersuchung der grammatikalischen Beziehungen der Wörter zueinander statt.

Für diese Zwecke eingesetzte Parser lassen sich in die zwei großen Gruppen der *Constituent Parser* und *Dependency Parser* einteilen. Ein Constituent Parser liefert bei Eingabe eines Satzes einen *Constituent Tree* und ein Dependency Parser einen *Dependency Graphen*. Beim Betrachten dieser Grapharten lassen sich anhand der durch sie kodiert dargestellten Syntax des Satzes sofort Aussagen darüber machen, welche Wörter in welcher Beziehung zueinander stehen. Vergleicht man wiederum Constituent Trees und Dependency Graphen miteinander, dann enthalten erste zwar mehr linguistische Informationen als zweite, dafür sind in Dependency Graphen Wort-Wort-Beziehungen expliziter dargestellt. Denn ein Dependency Graph ist *direkt aus den Wörtern* des Satzes aufgebaut und kommt ohne eine *abstrakte Superstruktur* aus, wie sie die Constituents in einem Constituent Tree bilden. Aus diesem Grund sind Dependency Graphen potentiell nützlicher für IE-Aufgaben der oben beschriebenen Art. [11, S. 430]

Andererseits sind von den beiden vorgestellten Syntax-Beschreibungsformen Consti-

tuent Trees ohne Zweifel die häufiger in der Linguistik und im NLP angewandte Form. Dies hat drei entscheidende Auswirkungen. Zum ersten existieren wesentlich mehr mit Constituent Trees als mit Dependency Graphen hand-annotierte Korpora (sogenannte *Treebanks*). Zum zweiten wurden bisher mehr Constituent Parser als Dependency Parser entwickelt und damit einhergehend wurden Constituent Parser häufiger in der Praxis eingesetzt und getestet. Zum dritten hat sich mit der Zeit ein de facto Standard für die Annotation von Constituent Trees herauskristallisiert, nämlich die Topologie und Constituent-Bezeichnungen der Penn Treebank [12], welche hand-annotierte Zeitungsartikel des Wall Street Journals umfasst. Für Dependency Graphen existiert solch ein Standard bisher nicht, d. h. jeder Parser und jedes Korpus benutzt eigene Annotationen.

Aus IE-Sicht ist dies ein Dilemma: Zum einen möchte man gern Dependency Graphen benutzen, was vor allem auch Performanz-Vergleiche verschiedener Dependency Parser einschliessen muss. Zum anderen existieren für solche Zwecke benötigte Standards und in der Praxis häufig eingesetzte Parser nur auf Constituent-Ebene.

## 1.2 Ziel und Aufbau dieser Arbeit

In dieser Arbeit werden zwei Constituent Parser auf Basis von Dependency Graphen evaluiert, es werden also beide Parsing-Verfahren miteinander kombiniert. Damit soll demonstriert werden, dass sich die im vorangegangenen Abschnitt 1.1 beschriebenen Probleme in gewisser Hinsicht lösen lassen, denn es wird bei diesem Vorgehen sowohl der für Constituent Trees existierende Standard ausgenutzt, als auch dem Wunsch Rechnung getragen, verschiedene Parser auf Ebene der Dependency Graphen zu evaluieren. Die Parser werden bezüglich ihrer Genauigkeit und Geschwindigkeit evaluiert. Darüber hinaus wird an Beispielen gezeigt, dass sich die Parser mit Hilfe von Dependency Graphen auch relativ leicht bezüglich konkreter biomedizinischer IE-Aspekte evaluieren lassen. Diese Vorgehensweise orientiert sich stark an der von CLEGG und SHEPHERD, welche bereits ein Paper zu selbiger Thematik veröffentlicht haben [13].

Die einzelnen für diesen Vorgang benötigten Schritte werden in Abschnitt 3 dieser Arbeit detailliert beschrieben, worauf im Abschnitt 4 die Evaluation folgt. Zuvor werden in Abschnitt 2 die linguistischen Grundlagen von Constituent Trees und Dependency Graphen erläutert und schließlich findet in Abschnitt 5 eine Zusammenfassung und Diskussion der Ergebnisse statt.

## 2 Linguistische Grundlagen

### 2.1 Constituent Trees

Ein *Constituent Tree* (auch Syntax-Baum oder Parsebaum) ist eine Darstellungsform für die grammatikalische Struktur (die Syntax) eines Satzes. Formal ist dies ein Baum, bei dem die Menge der Label der Blattknoten der Menge der Wörter der Sprache (in dieser Arbeit das englische Vokabular) und die Menge der Label der Nicht-Blattknoten der Menge von bestimmten linguistischen Komponenten – sogenannten *Constituents* (auch Phrasen) – entspricht. Der Wurzelknoten ist dem Constituent *S* zugeordnet. Eine Beispielauswahl von Constituent-Typen der Penn Treebank sieht man in Tabelle 1 und ein Beispiel für einen Constituent Tree in Abbildung 1 (für die vollständige Beschreibung aller Constituents der Penn Treebank siehe BIES et al. [14]).

Constituent-Typ	Bedeutung
S	Simple Clause
NP	Noun Phrase
PP	Prepositional Phrase
VP	Verb Phrase
ADJP	Adjective Phrase
CONJP	Conjunction Phrase

Tabelle 1: Auswahl an Constituents-Typen der Penn Treebank

Anhand des Baumes in Abbildung 1 lässt sich erkennen, dass jedes Wort als Vaterknoten immer einen Constituent besitzt, dieser wiederum selbst einen Constituent als Vaterknoten hat usw. bis schließlich die Wurzel erreicht wird. Dies entspricht der linguistischen Auffassung, welche erstmalig 1947 von WELLS [15] vorgestellt wurde, dass sich Wörter zu einer Art Einheit – einem Constituent – zusammenfassen lassen. Der größte gemeinsame Constituent ist demnach *S*, welcher den kompletten Satz umfasst. In der Abbildung lassen sich die Wörter „T-cell“ und „activation“ zu einem Constituent des Typs *NP* (Noun Phrase) zusammenfassen, stellt man das Wort „during“ noch davor, lassen sich alle drei Wörter zu einem Constituent des Typs *PP* (Prepositional Phrase) zusammenfassen usw.

Zur automatischen Generierung solcher Bäume für einen vorhandenen Korpus werden *Constituent Parser* eingesetzt. Ein Parser ist eine Implementation von verschiedenen Ableitungsregeln, die ausdrücken, aus welchen Constituents andere Constituents bzw. Wörter abgeleitet werden können. Diese Regeln kann man den Parsern entweder mit-

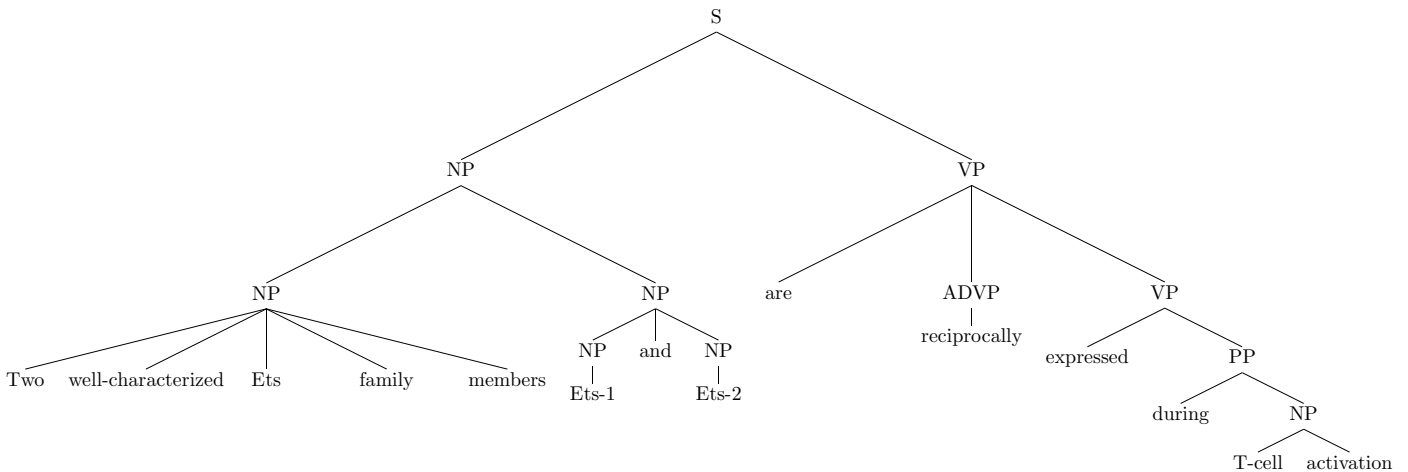


Abbildung 1: Constituent Tree des Satzes „Two well-characterized Ets family members, Ets-1 and Ets-2, are reciprocally expressed during T-cell activation.“ aus der GENIA Treebank (ohne Berücksichtigung von POS Tags und Satzzeichen)

geben oder man lässt sie die Regeln lernen, indem auf einem hand-annotierten Korpus (einer Treebank) die Häufigkeiten gezählt werden, wie oft welche Regel zutrifft. Bekannte im biomedizinischen NLP bereits zum Einsatz gekommene Constituent Parser der zweiten Variante sind beispielsweise die in dieser Arbeit verwendeten Bikel [16] und Stanford Parser [17] oder der Charniak Lease Parser [18]. Beispiele für Constituent Treebanks sind die bereits erwähnte Penn Treebank [12], die in dieser Arbeit verwendete GENIA Treebank [19] oder das deutsche TIGER Korpus [20].

## 2.2 Dependency Graphen

Eine alternative Beschreibungsform der grammatikalischen Struktur eines Satzes, welche ihren Ursprung 1959 in einer Arbeit von LUCIEN TESNIÈRE hat [21], ist ein *Dependency Graph*. Anders als ein Constituent Tree setzt dieser nicht verschiedene abstrakte syntaktische Einheiten (in Form von Constituents), sondern die Wörter selbst in Beziehung zueinander. Demnach stellen in einem Dependency Graphen die Knoten die Wörter des Satzes und die Kanten die Beziehungen bzw. Abhängigkeiten (*Dependencies*) zwischen den Wörtern dar. Ein Dependency Graph ist zudem *gerichtet*, was bedeutet, dass ein Wort von einem anderen Wort abhängt, aber nicht umgekehrt. Ein Auszug der vom Stanford Tool benutzten Dependencies ist in der Tabelle 2 zu sehen und ein Beispiel für einen Dependency Graphen in Abbildung 2 (für die vollständige Auflistung aller vom Stanford Tool benutzten Dependencies siehe MARNEFFE et al. [22]).

Dependency-Typ	Bedeutung
subj	Subject
nsubj	Nominal Subject
obj	Object
conj	Conjunction
auxpass	Passive Auxiliary
prep	Prepositional Modifier

Tabelle 2: Auswahl an Dependency-Typen des Stanford Tools

Ein Dependency Graph ist in der Regel ein gerichteter Baum, das heißt er ist zyklenfrei und zwischen zwei Knoten existiert nur ein Pfad (allerdings gibt es auch Ausnahmen mit mehr als einem Pfad, wie in Abschnitt 3.5 dieser Arbeit noch zu sehen sein wird, weshalb hier im Allgemeinen von Graphen gesprochen werden muss). Die Wurzel bildet das Verb des Satzes, was der linguistischen Vorstellung entspricht, dass vom Verb alle anderen Wörter entweder direkt oder indirekt über weitere Wörter abhängen. Im Beispielgraphen in Abbildung 2 hängen die Wörter „members“, „are“, „reciprocally“ und „during“ direkt vom Verb „expressed“ ab, alle anderen Wörter hängen entweder direkt oder indirekt von diesen vier Wörtern ab usw.

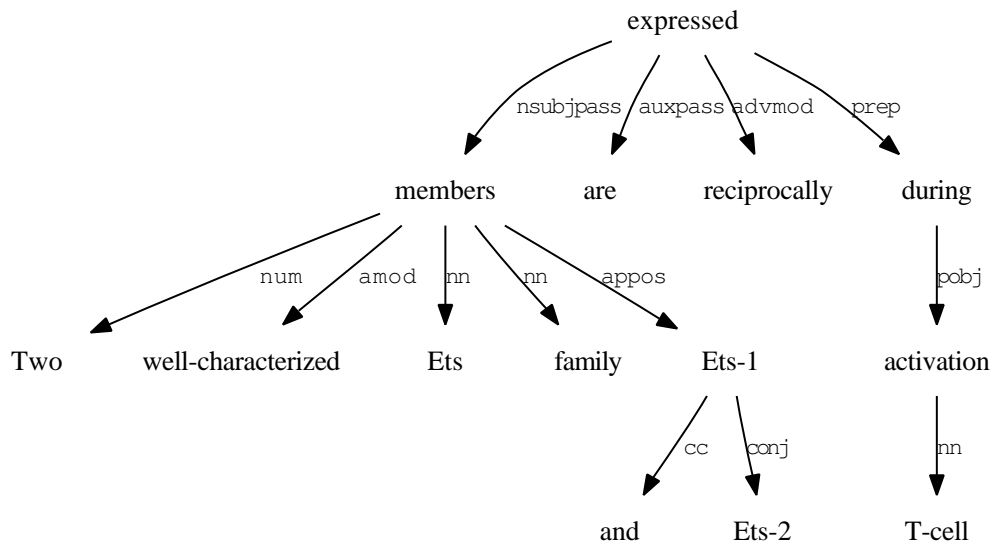


Abbildung 2: Dependency Graph des Satzes „Two well-characterized Ets family members, Ets-1 and Ets-2, are reciprocally expressed during T-cell activation.“ aus der GENIA Treebank

Die automatische Generierung eines Dependency Graphen für einen Satz erfolgt auch hier wieder über einen Parser – diesmal über einen *Dependency Parser*. Ähnlich wie ein

Constituent Parser ist dies eine Implementation von verschiedenen Regeln, nur diesmal eben nicht auf Ebene von Constituents, sondern von Wörtern. Diese können dem Programm auch wieder entweder per Hand mitgegeben werden oder aus einem hand-annotierten Korpus gelernt werden, wobei jeder Wort-Wort-Beziehung eine bestimmte Wahrscheinlichkeit zugewiesen wird. Beispiele für Dependency Parser sind der in dieser Arbeit verwendete Stanford Parser (bzw. seine Transformations-Komponente) [17], der Link Grammar Parser [23] oder der Minipar Parser [24]. Beispiele für mit Dependency Graphen annotierte Korpora sind die Prague Dependency Treebank [25] oder die PARC 700 Dependency Bank [26].

## 3 Vorgehen

### 3.1 Ablauf und beteiligte Komponenten

Wie eingangs beschrieben, werden in dieser Arbeit beide gängigen Parsing-Verfahren kombiniert in der Art, dass die auf Constituent Ebene vorhandenen Standards, Korpora und Praxis-bewährte Parser ausgenutzt werden, die Endanwendung (beispielsweise eine IE-Aufgabe) aber auf Ebene von Dependency Graphen stattfindet. Dieser Ebenenwechsel wird möglich durch die im Stanford Parser [27][28][17] integrierte Constituent-Tree-nach-Dependency-Graph-Transformations-Komponente (im Folgenden als Stanford Tool bezeichnet), welche einen Constituent Tree anhand von bestimmten Mapping-Regeln in einen Dependency Graph umwandelt [22]. Als Beispielend Anwendung wird im Folgenden eine Evaluation durchgeführt.

Folgende Schritte werden dafür benötigt: Zunächst müssen die zu vergleichenden Constituent Parser jeweils auf ein biomedizinisches Korpus angewandt werden. Dadurch entsteht pro Parser eine Menge von Constituent Trees. Das Korpus soll als *Goldstandard* fungieren, d. h. die Parser Outputs sollen gegen dieses Korpus evaluiert werden. Dies bedeutet, dass es bereits in Form von hand-annotierten Constituent Trees vorliegen muss und davon ausgegangen werden kann, dass diese Annotationen korrekt sind. Der nächste Schritt ist der Wechsel auf Dependency-Graph-Ebene, d. h. die Anwendung des Stanford Tools auf die Constituent Trees der Parser und des Goldstandards. Auf Basis der entstehenden Dependency Graphen erfolgt dann schließlich die eigentliche Evaluation.

Die für die konkrete Durchführung benutzten Komponenten sind der Bikel Parser in der Version 0.9.9c [29][16] und der oben bereits erwähnte Stanford Parser in der Version 1.6 als die zu vergleichenden Constituent Parser, die GENIA Treebank Beta 2 Version [19] – welche 1761 Sätze in 200 biomedizinischen Abstracts aus der

MEDLINE-Datenbank enthält – als hand-annotierte Treebank und das bereits erwähnte Stanford Tool als Transformations-Tool. Alle Komponenten sind frei verfügbar, die Parser und das Stanford Tool sind Open Source Software. Das Stanford Tool ist im Stanford Parser enthalten, kann aber extern aufgerufen werden. Sämtliche in den Zwischenschritten benötigte Modifikationen der In- und Outputs wurden vom Autor durch eigene Skripte in der Programmiersprache Perl realisiert.

### 3.2 Treebank strippen

Damit die Parser auf die Treebank angewandt werden können (und ihre eigenen Trees erstellen), muss diese zunächst im Klartext vorliegen, es müssen also sämtliche Annotationen entfernt werden, d. h. alle Constituents und POS Tags (Part Of Speech Tags – siehe Abschnitt 3.3). In der GENIA Treebank steht ein Constituent immer direkt hinter einer öffnenden Klammer. Sein Geltungsbereich endet an der dazu gehörenden schließenden Klammer und umfasst alle zwischen diesen Klammern stehenden Wörter. Die hinter jedem Wort des Satzes durch einen Slash gekennzeichneten Annotationen sind die POS Tags. Beide Konventionen entsprechen denen der Penn Treebank. In Abbildung 3 sieht man die typische Darstellung eines annotierten GENIA-Satzes (Constituents sind blau, POS Tags rot hervorgehoben) und sein gestripptes Klartext-Pendant.

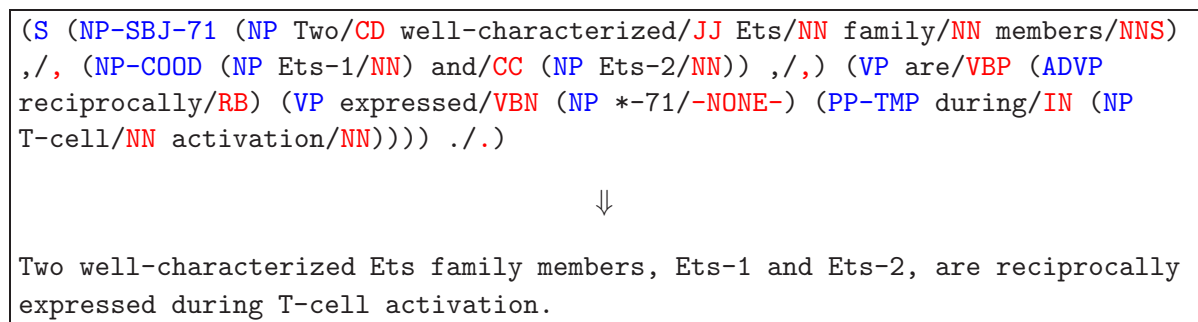


Abbildung 3: In GENIA hand-annotierte (oben) und gestrippte Klartext-Version (unten) des Satzes „Two well-characterized Ets family members, Ets-1 and Ets-2, are reciprocally expressed during T-cell activation.“

### 3.3 Part Of Speech Tagging

Das separate POS Tagging des Klartextes ist ein optionaler Schritt, da viele Parser selbst über ein POS-Tag-Modul verfügen. Allerdings erwarten manche, wie beispielsweise der in dieser Arbeit benutzte Bikel Parser, bereits vorgetaggten Text. Deshalb

soll dieser Zwischenschritt hier auch Beachtung finden. Ein *POS Tagger* ordnet jedem Wort/Satzzeichen seine Wortart (Part Of Speech) zu. Für biomedizinische Texte eignet sich der Medpost Tagger [30], welcher auf einer Auswahl von biomedizinischen Texten aus der MEDLINE-Datenbank trainiert wurde. Neben einem eigenen Output-Format bietet Medpost an, die POS Tags im Penn-Treebank-Format zu annotieren, was hier auch getan wurde. Der entstehende Output hat die Form `Wort/POS Tag`. In Abbildung 4 sieht man den Klartext-Beispielsatz und den entstandenen Text mit POS Tags, welche blau hervorgehoben sind.

<p>Two well-characterized Ets family members, Ets-1 and Ets-2, are reciprocally expressed during T-cell activation.</p> <p style="text-align: center;">↓</p> <p>Two/<b>CD</b> well-characterized/<b>JJ</b> Ets/<b>NN</b> family/<b>NN</b> members/<b>NNS</b> ,/, Ets-1/<b>NN</b> and/<b>CC</b> Ets-2/<b>NN</b> ,/, are/<b>VBP</b> reciprocally/<b>RB</b> expressed/<b>VCN</b> during/<b>IN</b> T-cell/<b>NN</b> activation/<b>NN</b> ./.</p>
--

Abbildung 4: Klartext-Version (oben) und POS-getaggte Version (unten) des Satzes „Two well-characterized Ets family members, Ets-1 and Ets-2, are reciprocally expressed during T-cell activation.“

Trotz Benutzung des Penn-Output-Formats muss der von Medpost generierte Text in einigen kleinen Punkten nachbearbeitet werden. Dies betrifft hauptsächlich Klammern- und Sonderzeichen-Behandlung. So werden beispielsweise in der Penn Treebank (und GENIA Treebank) öffnende Klammern als `-LRB-`, geschlossene Klammern als `-RRB-` usw. dargestellt, Medpost zeigt sie aber als einzelne ASCII-Zeichen an (Siehe Abbildung 5 zur Veranschaulichung).

<p>... expression/<b>NN</b> of/<b>IN</b> the/<b>DT</b> 1,25-(<b>OH</b>)2D3/<b>NN</b> receptor/<b>NN</b> ...</p> <p style="text-align: center;">↓</p> <p>... expression/<b>NN</b> of/<b>IN</b> the/<b>DT</b> 1,25--<b>LRB</b>-<b>OH</b>-<b>RRB</b>-2D3/<b>NN</b> receptor/<b>NN</b> ...</p>
--

Abbildung 5: Medpost-POS-Format (oben) und modifiziertes Medpost-POS-Format (unten) des Satzauszuges „expression of the 1,25-(OH)2D3 receptor“

### 3.4 Parsing

Wie in Abschnitt 2.1 bereits angesprochen, müssen die beiden hier benutzten Constituent Parser vor ihrem Einsatz trainiert werden, d. h., dass sie die Regeln, nach denen sie

später einen Klartext in verschiedene Constituents teilen, lernen müssen. Dies tun sie durch Betrachten eines bereits korrekt annotierten Korpus. Je häufiger eine Regel für bestimmte Constituent-Wort-Ableitungen in diesem Korpus zutrifft, desto höher wird ihre zugehörige Wahrscheinlichkeit und desto wahrscheinlicher wird diese Ableitung bei einem späteren Parsvorgang angewendet, wenn das Wort gesehen wird.

Beide Parser dieser Arbeit wurden auf einem Auszug der bereits angesprochenen Penn Treebank trainiert. Da diese aus Artikeln des Wall Street Journals besteht, ist sie sicherlich nicht die optimale Trainingsgrundlage für einen Parser, welcher später einen biomedizinischen Korpus parsen soll. Allerdings existieren zur Zeit noch keine biomedizinischen hand-annotierten Korpora, die groß genug wären, um die Parser darauf zu trainieren [13]. So enthält der hier zum Training verwendete Auszug der Penn Treebank (Section 02-21) 39 832 Sätze, die größten bisher vorhandenen biomedizinischen Treebanks umfassen lediglich 500 (GENIA Treebank) bzw. 642 Abstracts (PennBioIE Treebank [31]), was bei einer angenommenen Anzahl von 9 Sätzen pro Abstract nicht mehr als ca. 4500 bzw. ca. 5800 Sätzen entspricht.

### 3.4.1 Bikel Parser

Der Bikel Parser verlangt ein bestimmtes Input-Format, weshalb der vom Medpost Tagger generierte und modifizierte Output zunächst transformiert werden muss. Konkret wird das Medpost-POS-Format der Form `Wort/POS Tag` ins Bikel-POS-Format der Form `(Wort (POS Tag))` umgewandelt und zusätzlich um jeden Satz Klammern gesetzt. Der so entstehende Text kann dann vom Bikel Parser geparkt werden (Siehe Abbildung 6 für die Bikel Parser Pipeline).

Im Bikel Parser Output folgt auf jede öffnende Klammer entweder ein Constituent oder ein POS Tag, dessen Geltungsbereich an der dazugehörigen schließenden Klammer endet. Folgt auf eine Annotation – getrennt durch ein Leerzeichen – direkt ein Wort, handelt es sich um einen POS Tag, sonst um einen Constituent. Im Beispielsatz in Abbildung 6 sind somit alle blau hervorgehobenen Annotationen Constituents, alle anderen POS Tags.

Zu erwähnen ist hier außerdem, dass der Bikel Parser in seiner Grundeinstellung keine Sätze parst, die eine Satzlänge über 100 Wörter haben. Dies ist beim verwendeten GENIA Korpus zweimal der Fall, dementsprechend schlug der Parsvorgang bei diesen Sätzen fehl.

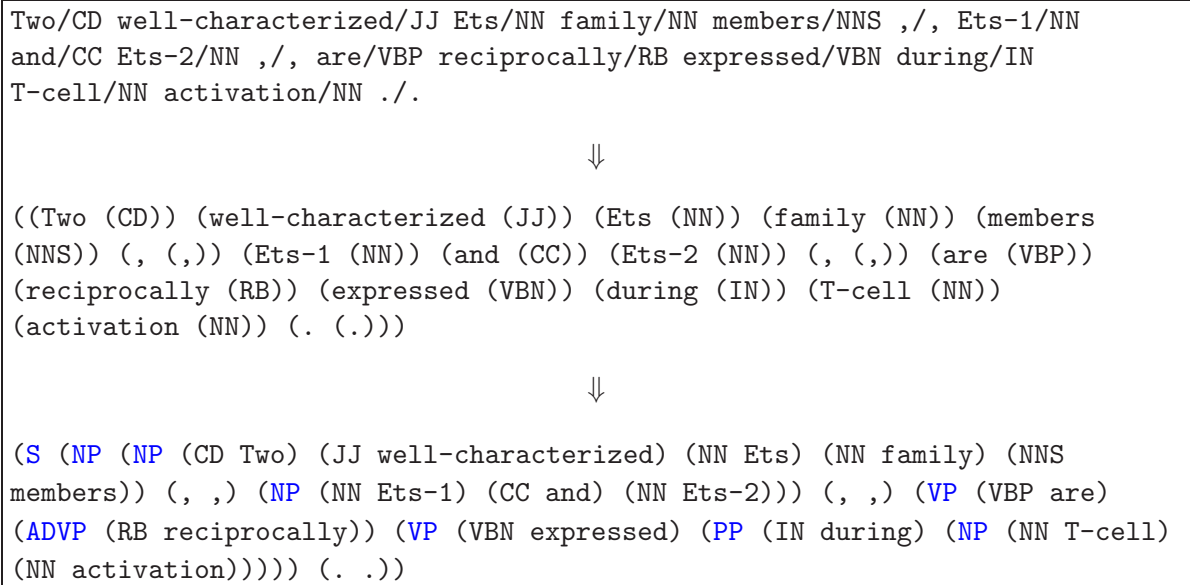


Abbildung 6: Medpost-POS-Format (oben), Bikel-POS-Format (mittig) und vom Bikel Parser erstellte Constituent Trees (unten) des Satzes „Two well-characterized Ets family members, Ets-1 and Ets-2, are reciprocally expressed during T-cell activation.“

### 3.4.2 Stanford Parser

Dem Stanford Parser kann der von Medpost generierte und anschließend modifizierte Text direkt ohne weitere Format-Transformationen übergeben werden. Er erstellt das selbe Output-Format wie der Bikel Parser mit der Ausnahme, dass er einen zusätzlichen ROOT Constituent erzeugt, welcher den kompletten Satz umfasst. Siehe dazu Abbildung 7 (Die Constituents sind blau hervorgehoben.).

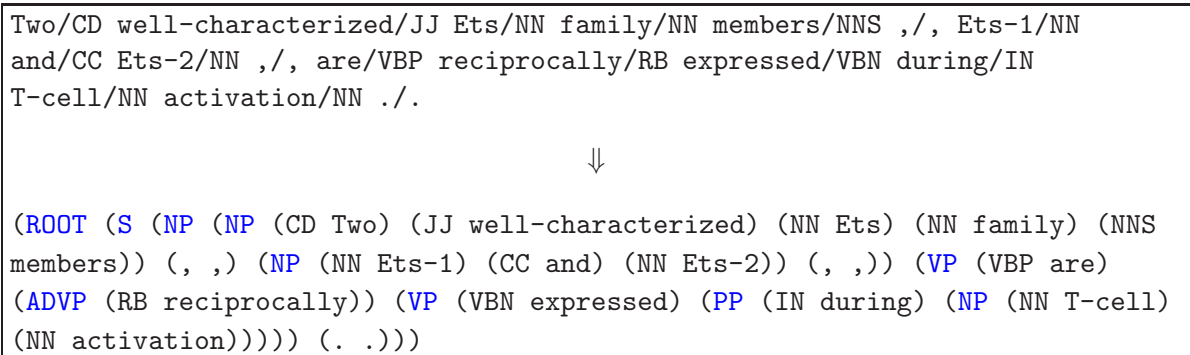


Abbildung 7: Medpost-POS-Format (oben) und vom Stanford Parser erstellte Constituent Trees (unten) des Satzes „Two well-characterized Ets family members, Ets-1 and Ets-2, are reciprocally expressed during T-cell activation.“

## 3.5 Transformation der Constituent Trees in Dependency Graphen

### 3.5.1 Konsistenz schaffen

Bevor die im vorherigen Abschnitt erzeugten Constituent Trees und die GENIA Treebank in Dependency Graphen transformiert werden, muss Konsistenz zwischen den Parser Outputs und den GENIA-Bäumen geschaffen werden, damit die Parser bei der späteren Evaluation nicht eventuell für Dinge bestraft werden, für die sie nicht verantwortlich sind (aus demselben Grund wurde auch der Medpost Tagger Output noch einmal modifiziert).

In der GENIA Treebank werden einige Constituents zusätzlich mit Attributen versehen, welche den jeweiligen Constituent näher spezifizieren, beispielsweise bezüglich Zeit (TMP-Attribut) oder Ort (LOC-Attribut). Da die verwendeten Parser keinerlei solcher Attribute generieren, werden diese bei GENIA entfernt. Ebenso werden leere Constituents, d. h. Constituents, welche nur Zeichen mit `-NONE-` POS Tags enthalten, aus GENIA entfernt. Zudem muss das GENIA-Format an sich transformiert werden in die vom Stanford Tool gewünschte Form. Zur Veranschaulichung siehe Abbildung 8, in der die zu entfernenden Teile blau hervorgehoben sind und das Format angepasst wurde.

Bei den Parser Outputs müssen alle `NX` und `NAC` Constituents durch `NP` ersetzt werden, da GENIA die beiden zuerst genannten nicht benutzt<sup>1</sup>. Beim Output des Stanford Parsers werden die erzeugten `ROOT` Constituents wieder entfernt, da diese keinen speziellen syntaktischen Zweck haben, sondern lediglich als zusätzlicher Überbau fungieren.

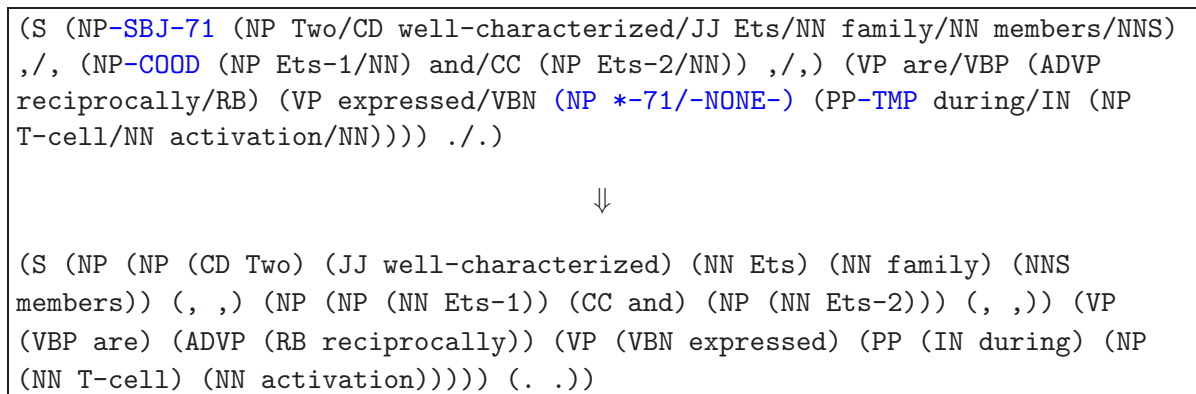


Abbildung 8: GENIA-Formatierung mit (oben) und ohne (unten) Constituent-Attribute und leeren Constituents für den Satz „Two well-characterized Ets family members, Ets-1 and Ets-2, are reciprocally expressed during T-cell activation.“

<sup>1</sup>`NX` und `NAC` sind Spezialisierungen des `NP` Constituents. So kennzeichnet beispielsweise `NX` den Anfang einer Aufzählung von mehreren NPs.

### 3.5.2 Transformation

Das Stanford Tool [22][17] gibt, angewandt auf einen Constituent Tree, einen Dependency Graphen in Form einer Menge von Dependencies aus. Dabei hat eine Dependency die Form `Dependency-Typ(Wort1-Wort1position, Wort2-Wort2position)`, was bedeutet, dass `Wort2` an Position `Wort2position` im Satz von `Wort1` an Position `Wort1position` im Satz bezüglich `Dependency-Typ` abhängt. In Abbildung 9 kann man den Constituent Tree des Beispielsatzes und seinen daraus generierten Dependency Graphen, d. h. die Menge seiner Dependencies sehen.

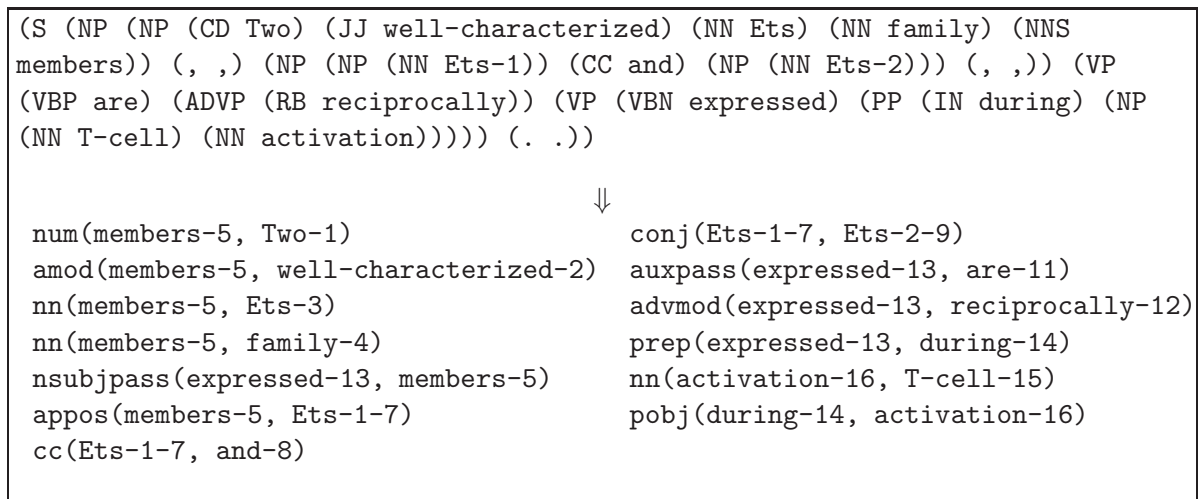


Abbildung 9: Constituent Tree aus der GENIA Treebank (oben) und durch das Stanford Tool erzeugter Dependency Graph (unten) des Satzes „Two well-characterized Ets family members, Ets-1 and Ets-2, are reciprocally expressed during T-cell activation.“

Das Stanford Tool bietet zudem zwei Möglichkeiten die entstehenden Dependency Graphen weiter zu verarbeiten und auszugeben – in „Collapsed“ oder „Collapsed And Conjunction Processed“ Form. Dabei werden bei einem „Collapsed“ Graph an bestimmten Wörtern (z. B. Konjunktionen, Präpositionen oder Possessivpronomen) die ein- und ausgehenden Dependencies zu einer neuen Misch-Dependency zusammengefasst und der Graph damit gewissermaßen zusammengeklappt. Dies kann für spätere IE-Anwendungen vielleicht nützlich sein, da dadurch der Graph insgesamt weniger Dependencies bekommt und bestimmte gesuchte Pfade zwischen Subjekten und Objekten kleiner werden.

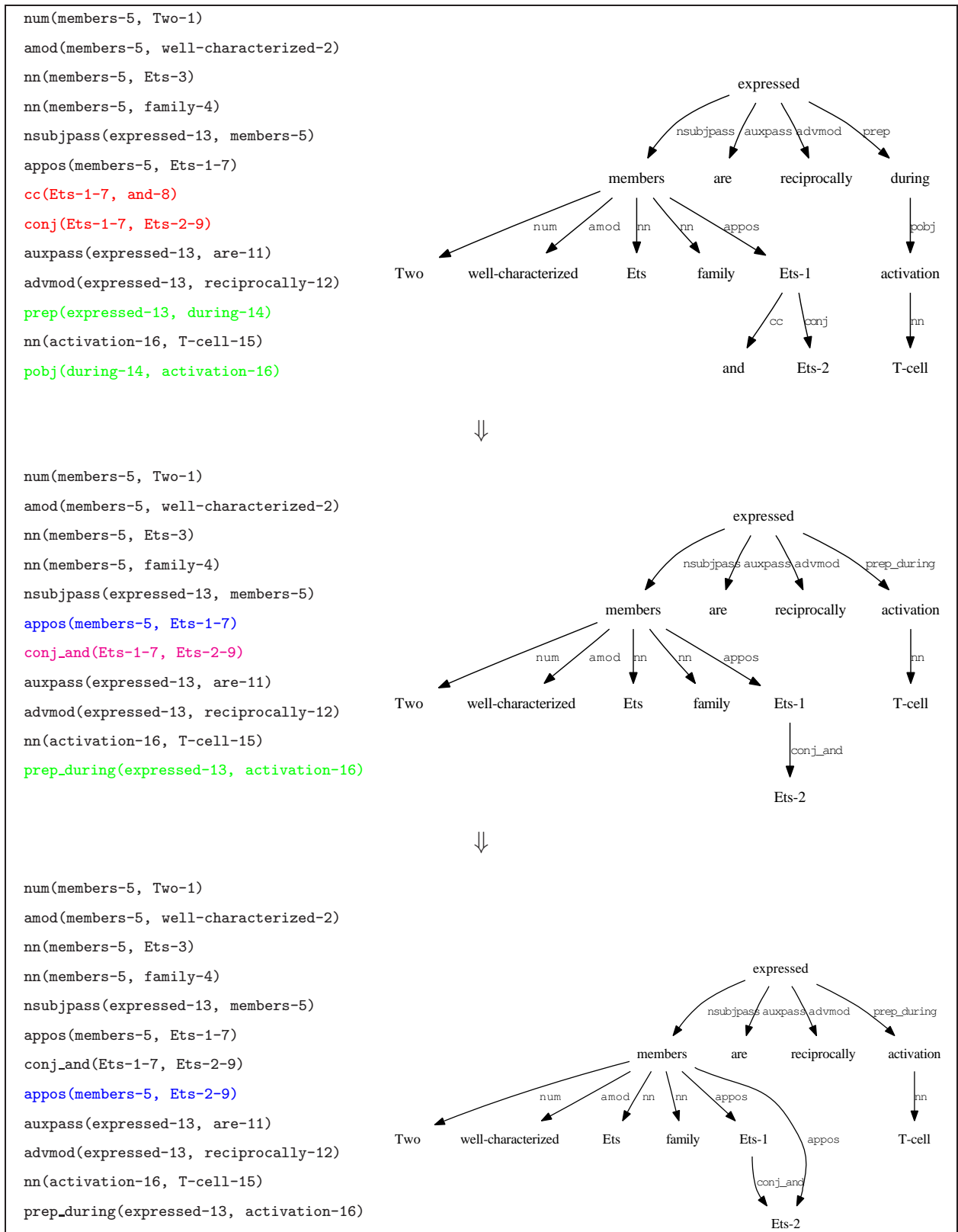


Abbildung 10: „Uncollapsed“ (oben), „Collapsed“ (mittig) und „Collapsed And Conjunction Processed“ (unten) Dependency Graph des Satzes „Two well-characterized Ets family members, Ets-1 and Ets-2, are reciprocally expressed during T-cell activation.“

Die zweite Variante „Collapsed And Conjunction Processed“ ist, wie der Name schon sagt, eine Erweiterung der ersten, und zwar in der Art, dass nach dem Zusammenklappen des Graphen an den Wörter mit neu entstandenen Konjunktions-Misch-Dependencies noch weitere Dependencies hinzugefügt werden. Auch diese Option ist aus IE-Sicht potentiell nützlich, da dadurch beispielsweise ein Subjekt nicht nur mit dem ersten Objekt, sondern mit allen anderen mit diesem Objekt durch eine Konjunktion verbundenen Objekten, verbunden wird (beispielweise in einer Aufzählung). In Abbildung 10 kann man sehen, wie aus den ursprünglichen Dependencies zuerst eine „Collapsed“ Version entsteht (aus den beiden grünen Dependencies wird eine grüne Dependency und aus den beiden roten Dependencies eine magenta Dependency) und daraus wiederum eine „Collapsed And Conjunction Processed“ Version des Graphen (mit Hilfe der magenta und blauen Dependency wird eine neue blaue Dependency hinzugefügt).

## 4 Evaluation

Evaluiert werden die beiden Parser bezüglich ihrer generellen Genauigkeit, bezüglich für biomedizinische IE wichtiger Wörter und Dependencies und bezüglich ihrer Geschwindigkeiten. Für den ersten Vergleich werden alle drei Dependency-Graph-Varianten verglichen und zudem auch die Constituent Trees mit evaluiert. Bei allen anderen späteren Vergleichen wird die „Uncollapsed“-Version der Dependency Graphen benutzt.

Als Vergleichsmaß werden die für Parser-Evaluationen gängigen *PARSEVAL-Meßgrößen* [32] benutzt (Siehe Abbildung 11).

- Precision  $P = \frac{True\ Positives}{True\ Positives + False\ Positives}$
- Recall  $R = \frac{True\ Positives}{True\ Positives + False\ Negatives}$
- F-Measure  $F = \frac{2 * P * R}{P + R}$
- Macro F-Measure  $F_{macro} = \frac{\sum_{S\ae tze} F_{S\ae tze}}{\# S\ae tze}$
- Micro F-Measure  $F_{micro} = \frac{2 * P_{alle\ S\ae tze} * R_{alle\ S\ae tze}}{P_{alle\ S\ae tze} + R_{alle\ S\ae tze}}$

Abbildung 11: Die für die Evaluation benutzten PARSEVAL-Meßgrößen

*True Positives* ist die Anzahl der durch den Parser korrekt generierten, *False Positives* die Anzahl der durch den Parser falsch generierten und *False Negatives* die vom Parser

nicht generierten, aber in der Treebank vorhandenen Dependencies bzw. Constituents. Damit gibt *Precision* also das Verhältnis von korrekt generierten zu insgesamt generierten, *Recall* das Verhältnis von korrekt generierten zu in der Treebank vorhandenen Dependencies/Constituents an und *F-Measure* bildet das harmonische Mittel aus diesen beiden Werten. Dabei werden zwei Varianten des F-Measures unterschieden: *Micro F-Measure* berechnet das F-Measure global über alle Dependencies bzw. Constituents und *Macro F-Measure* gibt das durchschnittlichen Pro-Satz-F-Measure wieder.

Für die Berechnungen muss festgelegt werden, was unter einem True Positive zu verstehen ist. Hier gilt bei der Evaluation von Constituent Trees, dass ein True Positive genau dann vorliegt, wenn der Constituent-Typ und die von ihm umfassten Wörter gleich sind im Constituent Tree des Parsers und der GENIA Treebank. Zur Veranschaulichung betrachte man den Constituent Tree der GENIA Treebank in Abbildung 1. Existiert im vom Parser erzeugten Constituent Tree für denselben Satz beispielsweise ein PP Constituent, welcher ebenfalls nur genau die Wörter „during“, „T-cell“ und „activation“ in derselben Reihenfolge umfasst, gilt dies als True Positive. Bei der Evaluation von Dependency Graphen liegt ein True Positive genau dann vor, wenn der Dependency-Typ, der Start- und der Endknoten (also die beiden in Beziehung stehenden Wörter) im Dependency Graph des Parsers und der GENIA Treebank gleich sind. Als Beispiel betrachte man den Dependency Graphen der GENIA Treebank in Abbildung 2. Wenn im zugehörigen Dependency Graphen des Parsers ebenfalls das Wort „members“ an Satzposition 5 vom Wort „expressed“ an Satzposition 13 bezüglich des Dependency-Typs nsubjpass abhängt, dann gilt dies als True Positive.

Die Vergleiche wurden vom Autor mit Hilfe selbst implementierter Skripte in der Programmiersprache Perl realisiert.

## 4.1 Evaluation bezüglich genereller Genauigkeit

Bei der Evaluation bezüglich genereller Genauigkeit werden alle Constituent Trees und Dependency Graphen beider Parser mit denen des Goldstandards bezüglich der im vorherigen Abschnitt genannten Kriterien verglichen. Bei den Dependency Graphen wurden alle drei möglichen Varianten evaluiert. Die Ergebnisse sieht man in Tabelle 3.

Wie man anhand der Tabelle sieht, unterscheiden sich die Parser in ihren Ergebnissen bezüglich Constituent Trees kaum, wohingegen ein deutlicherer Abstand zwischen ihnen bezüglich Dependency Graphen zu sehen ist. Die Werte der „Uncollapsed“ Dependency Trees sind generell höher als die der zugehörigen Constituent Trees. Ein Grund dafür könnte sein, dass bestimmte Constituents, die vorher negativ gewertet wurden, durch die

Parser	Constituents		Dependencies					
	$F_{macro}$	$F_{micro}$	Uncollapsed		Collapsed		Coll.&Conj.Proc.	
	$F_{macro}$	$F_{micro}$	$F_{macro}$	$F_{micro}$	$F_{macro}$	$F_{micro}$	$F_{macro}$	$F_{micro}$
Bikel	0,80	0,79	0,83	0,81	0,80	0,79	0,80	0,78
Stanford	0,80	0,78	0,81	0,80	0,78	0,76	0,78	0,76

Tabelle 3: Ergebnisse der Evaluation der Parser bezüglich Constituent Trees und Dependency Graphen („Uncollapsed“, „Collapsed“ und „Collapsed And Conjunction Processed“) über den gesamten Korpus

Transformation wegfallen oder mit anderen Constituents verschmelzen. Die  $F_{macro}$ -Werte sind stets höher als die  $F_{micro}$ -Werte, was bedeutet, dass kürzere Sätze insgesamt besser geparkt werden als längere.

Bei den verschiedenen Dependency-Graph-Varianten fällt auf, dass die „Collapsed“-Version kleinere Werte bekommt als die „Uncollapsed“. Die Ursache hierfür könnte sein, dass beim Kollabieren der Graphen zwei potentielle True Positives wegfallen und durch nur einen neuen ersetzt werden. Dass sich die Werte zwischen der „Collapsed“ und „Collapsed And Conjunction Processed“ Version wiederum so gut wie nicht unterscheiden, liegt sicherlich daran, dass die Anzahl der neu hinzukommenden Dependencies zu gering ist, um ernsthafte Auswirkungen auf die Ergebnisse zu haben.

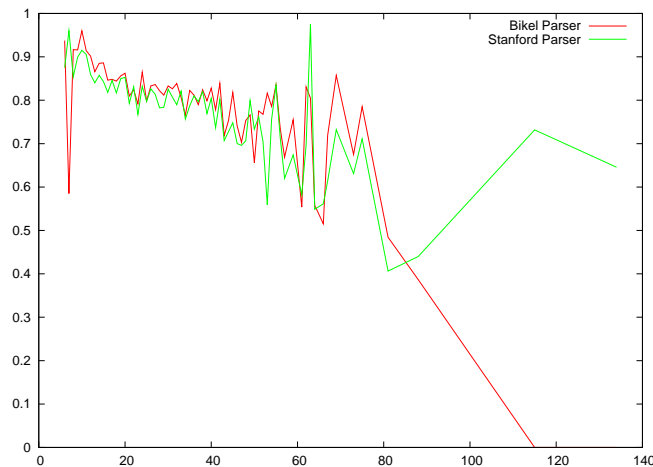


Abbildung 12: Durchschnittliches F-Measure pro Satzlänge des Bikel und Stanford Parsers sortiert nach Satzlängen (6-134)

In Abbildung 12 ist das durchschnittlichen F-Measure der beiden Parser für jede Satzlänge angezeigt (die kleinste Satzlänge ganz links ist 6, die größte ganz rechts ist

134). Wie man sieht, erreicht der Bikel Parser zwar meistens bessere Ergebnisse, allerdings gibt es immer wieder Ausnahmen, wo der Stanford Parser offensichtlich besser abschneidet.

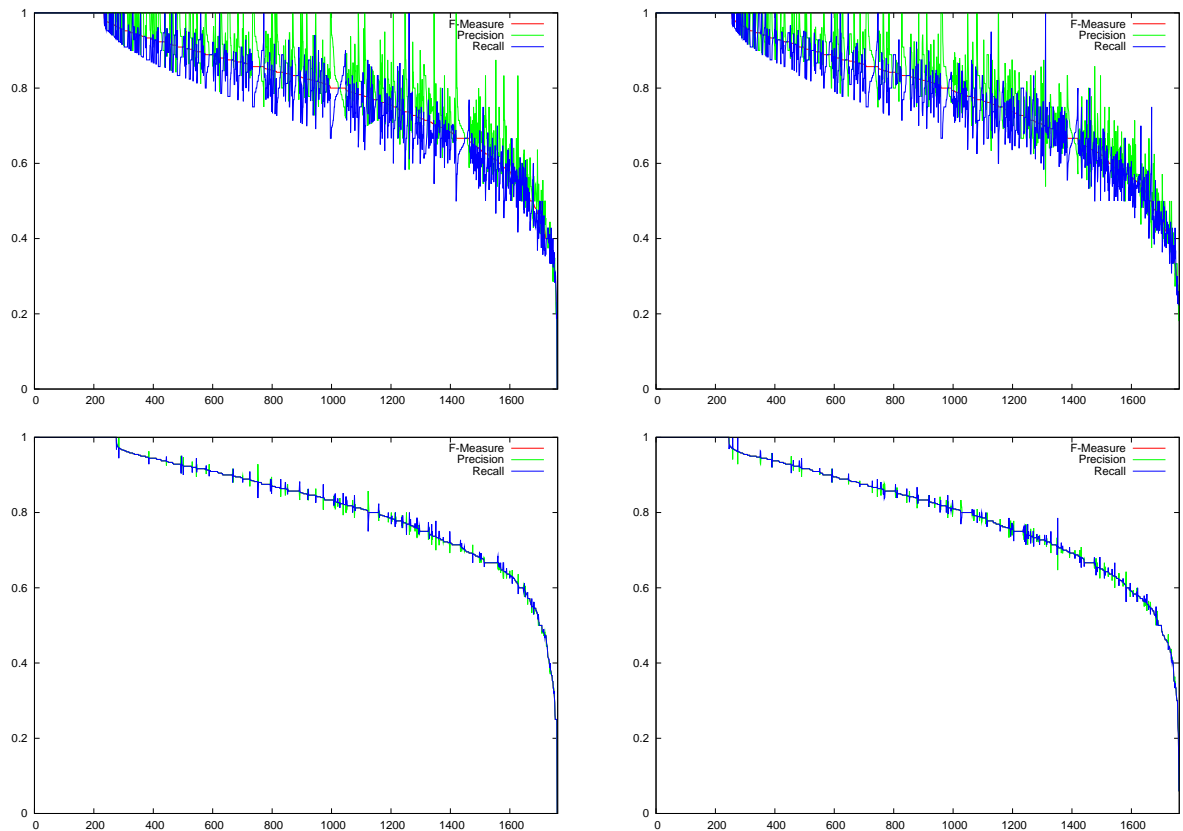


Abbildung 13: Satzweises Precision, Recall und F-Measure des Bikel (links) und Stanford Parsers (rechts) bezüglich Constituent Trees (oben) und Dependency Graphen (unten) jeweils sortiert nach F-Measure

In Abbildung 13 sind Precision, Recall und F-Measure für jeden Satz angezeigt (sortiert nach F-Measure). Man sieht anhand der Streuung der Werte sehr gut, dass Precision und Recall (und damit auch F-Measure) bei Dependency Graphen wesentlich mehr korrelieren als bei Constituent Trees, d. h. wenn ein Satz für einen Dependency Graphen eine hohe Precision erreicht, kann davon ausgegangen werden, dass auch sein Recall und F-Measure ähnlich hoch sein werden. Bei Constituent Trees ist dies sehr oft nicht der Fall, d. h. es treten viele Fälle auf mit hoher Precision und niedrigem Recall bzw. umgekehrt. Dies bedeutet, dass aus dem resultierenden F-Measure nicht ohne weiteres auf die zugehörige Precision bzw. Recall geschlossen werden kann.

Um festzustellen, ob beide Parser bei denselben Sätzen gut bzw. schlecht abschneiden,

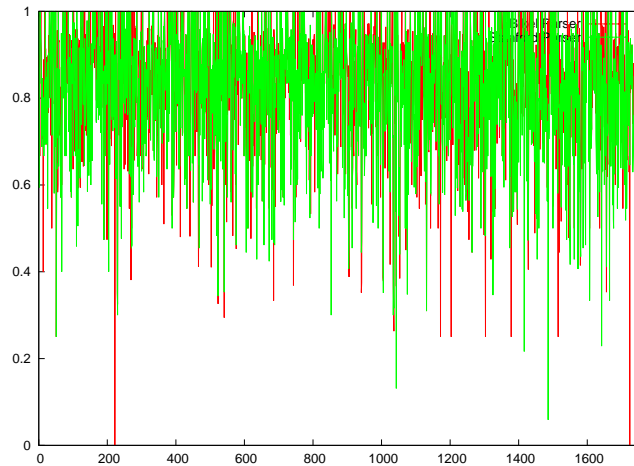


Abbildung 14: Satzweises F-Measure des Bikel und Stanford Parsers sortiert nach Vorkommen im Korpus

wurde in Abbildung 14 das satzweises F-Measure noch einmal für beide Parser eingetragen – diesmal nicht der Größe nach sortiert, sondern nach Vorkommen im Korpus. Wie man sieht, existiert keine Korrelation zwischen den Werten. Es scheint also so zu sein, dass die Parser nicht dieselben Schwächen und Stärken haben, was ja Abbildung 12 auch schon andeutete.

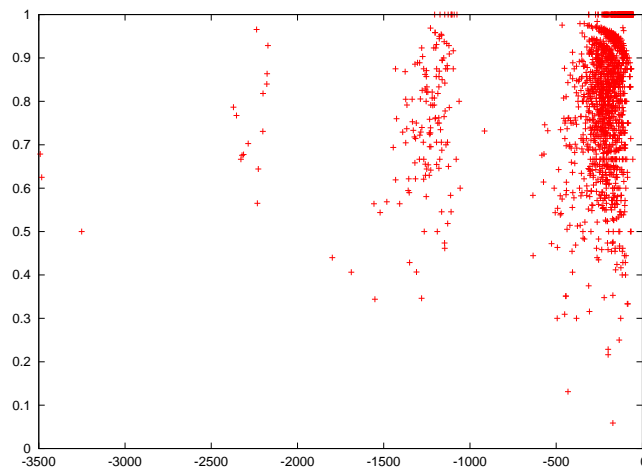


Abbildung 15: Verhältnis von F-Measure und Eigen score des Stanford Parsers

Der Stanford Parser bietet zudem die Möglichkeit für jeden Constituent Tree einen Eigen score auszugeben, der in der FAQ des Parsers als „log probability“ bezeichnet wird und den man als eine Art Maß für die Sicherheit des Parsers beim Parsvorgang

sehen kann. In Abbildung 15 sieht man diesen Score (X-Achse) in Beziehung zum jeweils zugehörigen F-Measure gesetzt. Dieses Diagramm gibt dementsprechend Auskunft, ob die Parsersicherheit mit der Korrektheit des entstehenden Graphen korreliert. Wie man sieht, tut er dies nicht. Tatsächlich ist der Korrelationskoeffizient nur 0,2. Auffällig ist allerdings, dass es drei ziemlich voneinander abgegrenzte Bereiche im Diagramm gibt.

## 4.2 Evaluation bezüglich bestimmter Wörter und Dependencies

Durch die Struktur der Dependency Graphen lassen sich die Parser mit einfachen Mitteln bezüglich bestimmter – im biomedizinischen Kontext wichtiger – Wörter und Dependencies evaluieren.

### 4.2.1 Evaluation bezüglich Interaktionswörtern

Eine typische IE-Aufgabe ist die Suche nach bestimmten Interaktionen zwischen Wörtern (beispielsweise zwischen Proteinen oder Genen) der Form „Agent Interaction Target“. In Tabelle 4 sieht man die Ergebnisse der Untersuchung drei solcher potentieller Interaktionswörter. Es handelt sich dabei um die Verben 'induce', 'activate' und 'express' und ihren Varianten (z. B. 'induces', 'activated', 'expressing' etc.). Dabei wurden nur Dependencies des Typs *Argument* verglichen, da nur diese potentiell für eine Interaktion in Frage kommen. Zudem wurde das Kriterium für ein True Positive abgeschwächt in der Art, dass ein Match zwischen zwei Dependencies vorliegt, wenn sie in derselben *Argument*-Kategorie sind, d. h. entweder in der *Subject*-Kategorie (entspricht Agent) oder *Complement*-Kategorie (entspricht Target). Dabei soll dem Rechnung getragen werden, dass es in einer IE-Anfrage nicht interessiert, ob der Agent vom Interaktionswort in Form eines *Nominal Subject*, *Passive Nominal Subject* oder *Clausal Subject* abhängt, sondern lediglich, dass der Agent hier eine Subjekt-Funktion hat.

Parser	induce		activate		express	
	$F_{macro}$	$F_{micro}$	$F_{macro}$	$F_{micro}$	$F_{macro}$	$F_{micro}$
Bikel	0,84	0,85	0,87	0,88	0,85	0,86
Stanford	0,80	0,81	0,85	0,87	0,78	0,82

Tabelle 4: Ergebnisse der Evaluation der Parser bezüglich der Verben 'induce', 'activate' und 'express'

Wie man sieht, sind die Ergebnisse meistens wesentlich höher als bei der Gesamtevaluation (vor allem beim Bikel Parser).

### 4.2.2 Evaluation bezüglich Präpositionen

Für die richtige Interpretation eines Satzes ist die korrekte Annotation von Präpositionen unerlässlich, d. h. es muss klar hervorgehen, welches Wort der Modifikator der Präposition ist und welches Wort durch die Präposition modifiziert wird. In Tabelle 5 sieht man die Auswertung beider Gesichtspunkte.

Parser	Modifikator		Modifiziertes Wort	
	$F_{macro}$	$F_{micro}$	$F_{macro}$	$F_{micro}$
Bikel	0,81	0,80	0,91	0,92
Stanford	0,79	0,78	0,91	0,91

Tabelle 5: Ergebnisse der Evaluation der Parser bezüglich korrekter Annotation von Präpositionen

Während beide Parser bezüglich korrekter Annotation der Präposition am Modifikator ungefähr dieselben Werte wie bei der Gesamtevaluation erreichen, sind die Werte bezüglich korrekter Annotation der Präposition an das zu modifizierende Wort erheblich größer.

### 4.3 Evaluation bezüglich Geschwindigkeit

Beide Parser wurden in ihren Default-Einstellungen benutzt, es wurden also keine speziellen Optionen bezüglich Performanz eingestellt, sondern nur solche, die das gewünschte Input- und Outputformat der Daten charakterisieren. Die Auswertung der Geschwindigkeiten findet sich in Tabelle 6. Die dort aufgeführten Vorgänge fanden alle auf einem Solaris X86 8 Dual-Core-AMD 8218 mit 32 GB RAM (gruenau1.informatik.hu-berlin.de) statt. Für die Zeitberechnung wurde das GNU-Programm *time* verwendet.

Parser	Parsing		Transformation
	Gesamt	Pro Satz	
Bikel	2h 16m 13,5s	4,6s	2m 6,9s
Stanford	51m 29,8s	1,7s	2m 12,1s

Tabelle 6: Geschwindigkeiten der Parsvorgänge, des durchschnittlichen Pro-Satz-Parsings und des Transformationsvorgangs

Wie man sieht, war der Stanford Parser etwa 2,5-mal schneller als der Bikel Parser. Für die Constituent-Tree-nach-Dependency-Graph-Transformation wurde erwartungsgemäß etwa gleich viel Zeit benötigt.

## 5 Diskussion und Ausblick

In dieser Studienarbeit wurde anschaulich in allen Zwischenschritten gezeigt, wie sich zwei Constituent Parser auf Basis von Dependency Graphen evaluieren lassen. Dabei wurde unter anderem zum Ausdruck gebracht, dass die Evaluation auf Basis von Dependency Graphen nicht nur allgemeine Vergleiche von Constituent Parsern möglich macht, sondern zudem auch speziellere Vergleiche bezüglich in biomedizinischen Texten wichtiger Aspekte. So wurde in dieser Arbeit beispielsweise das Verhalten der Parser bei bestimmten Interaktionswörtern oder Präpositionen untersucht, was in dieser Form auf Ebene von Constituent Trees nicht möglich ist.

Bei der Evaluation bezüglich genereller und vor allem bezüglich der untersuchten spezielleren Genauigkeiten erreichte der Bikel Parser meistens die klar besseren Ergebnisse. Dafür war der Bikel Parser bei zwei Sätzen nicht in der Lage, sie zu parsen. Zudem war der Stanford Parser mehr als 2,5-mal so schnell im eigentlichen Parsvorgang. Dies deckt sich in etwa mit den Ergebnissen von CLEGG und SHEPHERD [13].

Durch die insgesamt ziemlich guten Ergebnisse bezüglich der Genauigkeit der Parser (vor allem bezüglich der spezielleren Genauigkeiten), ist die Einbindung von Parsing-Verfahren in IE-Pipelines prinzipiell empfehlenswert. Dies unterstreichen auch erste Anwendungen von Dependency Graphen in der IE [10], bei denen mit Hilfe des Stanford Parsers bereits F-Measures von 78% – 82% im Finden von Protein-Protein-Interaktionen erreicht wurden. Einzig die Geschwindigkeiten für die einzelnen Parsvorgänge sind für bestimmte Anwendungen sicherlich zu langsam (beispielsweise in Online-Applikationen). Allerdings könnte man in diesen Fällen Parsing-Verfahren vielleicht mit anderen schnelleren NLP Verfahren, wie beispielsweise Pattern Matching, kombinieren in der Art, dass ein Parser nur dann zu Rate gezogen wird (auf Kosten von Zeit), wenn ein schnelleres Verfahren zu schlechte Ergebnisse geliefert hat.

## Literatur

- [1] JENSSEN, T.K. et al.: A literature network of human genes for high-throughput analysis of gene expression. In: *Nat. Genet.* (2001), Nr. 28, S. 21–28
- [2] DING, J. et al.: Mining medline: abstracts, sentences, or phrases? In: *Pac. Symp. Biocomput.* (2002), Nr. 7, S. 326–337
- [3] JELIER, R. et al.: Co-occurrence based meta-analysis of scientific texts: retrieving biological relationships between genes. In: *Bioinformatics* (2005), Nr. 21, S. 2049–2058
- [4] BLASCHKE, C. et al.: Automatic extraction of biological information from scientific text: protein-protein-interaction. In: *Proc. Int. Conf. Intell. Syst. Mol. Biol.* (1999), S. 60–67
- [5] BLASCHKE, C. ; VALENCIA, A.: The potential use of suiseki as a protein interaction discovery tool. In: *Genome Inform. Ser. Workshop Genome Inform.* (2001), Nr. 12, S. 123–134
- [6] LEROY, G. ; CHEN, H.: Filling preposition-based templates to capture information from medical abstracts. In: *Pac. Symp. Biocomput.* (2002), Nr. 7, S. 350–361
- [7] ONO, T. et al.: Automated extraction of information on protein-protein interactions from the biological literature. In: *Bioinformatics* (2001), Nr. 17, S. 155–161
- [8] TEMKIN, Joshua M. ; GILDER, Mark R.: Extraction of protein interaction information from unstructured text using a context-free grammar. In: *Bioinformatics* (2003). <http://bioinformatics.oxfordjournals.org/cgi/content/abstract/19/16/2046>, Abruf: 25. April 2008
- [9] AHMED, Syed T. ; CHIDAMBARAM, Deepthi ; DAVULCU, Hasan ; BARAL, Chitta: IntEx: A Syntactic Role Driven Protein-Protein Interaction Extractor for Bio-Medical Text. In: *Proceedings of the ACL-ISMB Workshop on Linking Biological Literature. Ontologies and Databases: Mining Biological Semantics* (2005). <http://www.public.asu.edu/~cbaral/papers/intex-camera.pdf>, Abruf: 25. April 2008
- [10] FUNDEL, Katrin ; KÜFFNER, Robert ; ZIMMER, Ralf: RelEx - Relation extraction using dependency parse trees. In: *Bioinformatics* (2006). <http://bioinformatics.oxfordjournals.org/cgi/content/full/23/3/365>, Abruf: 25. April 2008

- [11] MANNING, Christopher D. ; SCHÜTZE, Hinrich: *Foundations of Statistical Natural Language Processing*. Second Printing. Cambridge/London : The MIT Press, 2000
- [12] *The Penn Treebank Project*. <http://www.cis.upenn.edu/~treebank/>, Abruf: 25. April 2008
- [13] CLEGG, Andrew B. ; SHEPHERD, Adrian J.: Benchmarking natural-language parsers for biological applications using dependency graphs. In: *BMC Bioinformatics* (2007). <http://www.biomedcentral.com/1471-2105/8/24>, Abruf: 25. April 2008
- [14] BIES, A.: Bracketing Guidelines for Treebank II Style Penn Treebank Project. (1995). <http://www ldc.upenn.edu/Catalog/desc/addenda/LDC1999T42/PRSGUID1.PDF>
- [15] WELLS, Rulon S.: Immediate Constituents. In: *Language* (1947), Nr. 23, S. 81–117
- [16] *Bikel Parser*. <http://www.cis.upenn.edu/~dbikel/download.html>, Abruf: 25. April 2008
- [17] *Stanford NLP Tools*. <http://nlp.stanford.edu/software/index.shtml>, Abruf: 25. April 2008
- [18] *Charniak Lease Parser*. <http://bllip.cs.brown.edu/resources.shtml>, Abruf: 25. April 2008
- [19] *GENIA Treebank*. <http://www-tsujii.is.s.u-tokyo.ac.jp/GENIA/topics/Corpus/GTB.html>, Abruf: 25. April 2008
- [20] *TIGER PROJECT*. <http://www.ims.uni-stuttgart.de/projekte/TIGER/>, Abruf: 25. April 2008
- [21] TESNIÈRE, Lucien: *Éléments de Syntaxe Structurale*. Paris : Librairie C. Klincksieck, 1959
- [22] MARNEFFE, Marie-Catherine de ; MACCARTNEY, Bill ; MANNING, Christopher D.: Generating Typed Dependency Parses from Phrase Structure Parses. In: *Proceedings of 5th International Conference on Language Resources and Evaluation (LREC 2006)* (2006). [http://nlp.stanford.edu/pubs/LREC06\\_dependencies.pdf](http://nlp.stanford.edu/pubs/LREC06_dependencies.pdf), Abruf: 25. April 2008
- [23] *Link Grammar Parser*. <http://www.link.cs.cmu.edu/link/>, Abruf: 25. April 2008

- [24] *Minipar Parser*. <http://www.cs.ualberta.ca/~lindek/minipar.htm>, Abruf: 25. April 2008
- [25] *The Prague Dependency Treebank*. <http://ufal.mff.cuni.cz/pdt/>, Abruf: 25. April 2008
- [26] *The PARC 700 Dependency Bank*. <http://www2.parc.com/isl/groups/nlitt/fsbank/default.html>, Abruf: 25. April 2008
- [27] KLEIN, Dan ; MANNING, Christopher D.: Fast Exact Inference with a Factored Model for Natural Language Parsing. In: *Advances in Neural Information Processing Systems 15 (NIPS 2002)* (2003). <http://www-nlp.stanford.edu/~manning/papers/lex-parser.pdf>, Abruf: 25. April 2008
- [28] KLEIN, Dan ; MANNING, Christopher D.: Accurate Unlexicalized Parsing. In: *Proceedings of the 41st Meeting of the Association for Computational Linguistics* (2003). <http://nlp.stanford.edu/~manning/papers/unlexicalized-parsing.pdf>, Abruf: 25. April 2008
- [29] BIKEL, Dan: Design of a Multi-lingual, Parallel-processing Statistical Parsing Engine. In: *Proceedings of the Human Language Technology Conference 2002 (HLT2002)* (2002). <http://www.cis.upenn.edu/~dbikel/#research>, Abruf: 25. April 2008
- [30] SMITH, L. ; RINDFLESCH, T. ; WILBUR, W. J.: MedPost: a part-of-speech tagger for bioMedical text. In: *Bioinformatics* (2004). [http://www.ncbi.nlm.nih.gov/sites/entrez?cmd=Retrieve&db=PubMed&dopt=AbstractPlus&list\\_uids=15073016](http://www.ncbi.nlm.nih.gov/sites/entrez?cmd=Retrieve&db=PubMed&dopt=AbstractPlus&list_uids=15073016), Abruf: 25. April 2008
- [31] *Mining the Bibliome*. <http://bioie ldc.upenn.edu/>, Abruf: 25. April 2008
- [32] BLACK et al.: A procedure for quantitatively comparing the syntactic coverage of English grammars. In: *Proceedings, Speech and Natural Language Workshop* (1991), S. 306–311

## **Selbstständigkeitserklärung**

Hiermit erkläre ich, dass ich die Studienarbeit selbstständig und nur unter Zuhilfenahme der angegebenen Quellen angefertigt habe.

Berlin, 25. April 2008