Jörg Didakowski

Robustes Parsing und Disambiguierung mit gewichteten Transduktoren

Linguistics in Potsdam | 23

Linguistics in Potsdam

Series Editors:

Susann Fischer
Institut für Linguistik Potsdam

Ruben van de Vijver
Institut für Linguistik Potsdam

Ralf Vogel
Institut für Linguistik Potsdam

Linguistics in Potsdam | 23

Jörg Didakowski

Robustes Parsing und Disambiguierung mit gewichteten Transduktoren

Universitätsverlag Potsdam 2005

Bibliografische Information Der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.

LiP issues do not appear according to strict schedule.

Series Editors: Susann Fischer
Ruben van de Vijver
Ralf Vogel

Institut für Linguistik
Universität Potsdam
Postfach 60 15 53
D-14415 Potsdam
{sfisher, ruben, rvogel}@ling.uni-potsdam.de

Published by Universitätsverlag Potsdam
Postfach 60 15 53, 14415 Potsdam
Fon +49 (0) 331 977 4517
Fax +49 (0) 331 977 4625
e-mail: ubpub@uni-potsdam.de
http://info.ub.uni-potsdam.de/verlag.htm

Printed by Audiovisuelles Zentrum der Universität Potsdam
Published 2005

ISBN 3-937786-87-2
ISBN 978-3-937786-87-2
ISSN 1616-7392

Vorwort

In dieser Arbeit wird ein Verfahren für robustes Parsing von uneingeschränktem natürlichsprachlichen Text mit gewichteten Transduktoren erarbeitet. Es werden zwei linguistische Theorien, das Chunking und das syntaktische Tagging, vorgestellt, die sich besonders für die praktische Anwendung mit Finite-State Maschinen eignen. Über die formalen Grundlagen, die es möglich machen, Finite-State Maschinen zu modellieren, werden existierende Ansätze vorgestellt, die diese linguistischen Theorien mit Finite-State Maschinen realisieren. Jedoch sind diese Ansätze in vieler Hinsicht problematisch. Es wird gezeigt, dass sich Probleme lösen lassen, indem Disambiguierungsstrategien durch Constraints realisiert werden, die als Gewicht bzw. Semiring vorliegen. Durch die Bestimmung des besten Pfades ist dann eine Disambiguierung möglich. Das Verfahren bewegt sich zwischen einem Low- und High-Level Parsing und behandelt flache Dependenzstrukturen. Für die Analyse wird eine rudimentäre Grammatik für das Deutsche entwickelt. Durch eine Implementierung wird letztlich der Ansatz getestet.

Es handelt sich hierbei um meine Diplomarbeit im Fach Computerlinguistik am Institut für Linguistik, an der Universität Potsdam, die ich Anfang 2005 fertigstellte. Ich danke meinem Betreuer Thomas Hanneforth für seine tatkräftige Unterstützung und seine zahlreichen Anregungen. Ohne Ihn wäre eine Implementierung meines Ansatzes nicht ohne weiteres möglich gewesen. Ich danke auch Jette Klein-Berning, die mir für heikle Fragen stets zur Seite stand. In zahlreichen Gesprächen wurde ich so immer auf neue Ideen gebracht.

Inhaltsverzeichnis

1 Einführung

Im Rahmen meiner Arbeit beim *Digitalen Wörterbuch der deutschen Sprache des 20. Jahrhunderts*[1] (DWDS) sollte ein Parser zur Unterstützung der lexikographischen Analyse entwickelt werden. Dabei sollte eine strikte Trennung zwischen Grammatik und Parser herrschen.

Die syntaktische Analyse von Belegen setzt ein robustes Parsingverfahren voraus, das die Verarbeitung eines großen Belegkorpus in kürzester Zeit ermöglicht. Dabei ist man vor allem an den Beziehungen zwischen Wörtern interessiert. Eine erschöpfende syntaktische Analyse ist hierbei allerdings nicht zwingend notwendig.

Für eine flache Analyse eignen sich besonders endliche Automaten und Transduktoren. In der maschinellen Sprachverarbeitung haben sich diese Maschinen auf vielen Gebieten bewährt. Richtig eingesetzt machen sie eine robuste und effiziente Verarbeitung natürlicher Sprache möglich. Über reguläre Ausdrücke ist in diesem Zusammenhang die gewünschte Trennung von Grammatik und Parser möglich.

Jedoch können einige Aufgaben durch endliche Automaten und Transduktoren nicht unbedingt gelöst werden. Eine Disambiguierungsstrategie muss hier in lineare Abfolgen kodiert werden. Es ist nicht möglich, Analysen miteinander zu vergleichen und nach ihrer Akzeptanz zu bewerten. Mit gewichteten Automaten und Transduktoren lässt sich dagegen ein paarweises Vergleichen von Analysen realisieren. So kann eine Disambiguierungsstrategie in Form von Analysebewertungen verwirklicht werden. Eine syntaktische Analyse kann dann als Disambiguierung mehrerer Alternativen angesehen werden.

[1] Der gegenwärtige Stand des Projektes ist einzusehen unter www.dwds.de.

Zunächst werde ich auf zwei linguistische Theorien eingeben, die im Bereich der robusten Sprachverarbeitung mit Finite-State Maschinen Anwendung gefunden haben. Einerseits werden dort lokale Konstituentenstrukturen, andererseits globale Dependenzstrukturen behandelt. Über lokale Dependenzstrukturen können dann beide Ansätze zusammengebracht werden.

Dann werde ich die formalen Grundlagen erläutern, die die Voraussetzung für die Realisierung der linguistischen Theorien mit Finite-State Maschinen bilden. Hierbei liegt der Fokus auf den regulären Ausdrücken, mit denen Finite-State Maschinen modelliert werden können. Ich werde auf Probleme eingehen, die bei der Realisierung der linguistischen Theorien auftreten. Zum einen sind dies Probleme der Berechnung und Robustheit, zum anderen durch eine bestimmte Parsingstrategie bedingte falsche Analysen.

Mit Hilfe von gewichteten Transduktoren werde ich über reguläre Ausdrücke ein Verfahren entwickeln, das es möglich macht, die genannten Probleme zu lösen. Die Begriffe *Semiring* und *bester Pfad* stehen dabei im Mittelpunkt. Es werden Semiringe entwickelt, die die Ermittlung der besten Analysen ermöglichen.

Auf der Basis des hier entwickelten Verfahrens werde ich eine rudimentäre Grammatik erstellen, die in einer Implementierung auf eine Anzahl von Testsätzen angewendet wird. Zuletzt erfolgt die Diskussion der Ergebnisse.

2 Grundlegende linguistische Theorien

Es existieren zwei Ansätze, die für robustes Parsing mit Finite-State Maschinen sehr entscheidend sind: das *Chunking* und das *syntaktische Tagging*. Beide Ansätze befassen sich mit der syntaktischen Analyse. Abney [3] führt den Begriff *Chunk* ein. Chunks werden konstruktiv aufgebaut, sie bilden syntakti-

sche Gruppierungen und können als lokale Bäume von globalen Parse-Bäumen angesehen werden. Karlsson [28] führt syntaktische Tags ein, um eine flache Abhängigkeitsstruktur über Sätze aufzubauen. Die Analyse ist reduktionistisch, das Prinzip der Disambiguierung steht dabei im Vordergrund. In diesem Kapitel sollen beide Ansätze vorgestellt werden und es soll darauf eingegangen werden, inwieweit es möglich ist, diese zusammenzuführen.

2.1 Syntaktisches Tagging

Karlsson schlägt in [29] eine flache dependenz-orientierte Analyse von Sätzen vor. Diese besteht aus der Festlegung von Satzgrenzen, der Auflösung von morphologischen Ambiguitäten und der Disambiguierung von syntaktischen Funktionen.

Es existieren mehrere Klassifizierungen von Ambiguitäten: semantisch, pragmatisch und strukturell. Bei dem von Karlsson vorgestellten Verfahren werden nur strukturelle Ambiguitäten berücksichtigt. Eine Erweiterung auf andere Klassen wäre aber durchaus denkbar. Strukturelle Ambiguitäten können in globale und lokale Ambiguitäten unterschieden werden. Lokale Ambiguitäten betreffen einige adjazente Wörter. Diese können durch ihren Satzkontext aufgelöst werden. Globale Ambiguitäten betreffen einen ganzen Satz, sie sind ohne weiteren Kontext über die Satzgrenzen hinaus nicht aufzulösen. Die Differenzierung von lokalen und globalen Ambiguitäten ist jedoch nicht immer so eindeutig. Ein Beispiel dafür sind Relativsätze, in denen von globalen Ambiguitäten gesprochen werden kann, obwohl Relativsätze keine unabhängigen Sätze darstellen. Die häufigsten lokalen Ambiguitäten, die Homonyme, in diesem Fall Homographen, beziehen sich nur auf eine Wortform. Wörter können *Paradigma Ambiguitäten* enthalten:

(1) receive:

- finites Verb im Subjunktiv
- finites Verb im Imperativ
- infinites Verb
- finites Verb im Präsens

Ebenso können Wörter auch *kategorielle Ambiguitäten* enthalten:

(2) saw:

- Nomen
- Verb

Diese verschiedenen morphologischen Interpretationen eines Wortes werden als Lesarten bezeichnet. Wenn sich lokale Ambiguitäten über mehrere adjazente Wörter ausbreiten, spricht man von *Attachment-Ambiguitäten*. Attachment-Ambiguitäten beziehen sich im Normalfall auf Adjunktanbindungen. Das klassische Beispiel für eine Attachment-Ambiguität ist die PP-Anbindung in dem Satz *I saw the man on the hill with the telescope* ([29], Kapitel 2.1).

Die strukturellen Ambiguitäten sollen im Folgenden durch spezielle Disambiguierungsregeln eliminiert werden, um letztlich eine flache Dependenzstruktur zu extrahieren.

2.1.1 Dependenzstruktur

Es existieren zwei grundlegende Arten, Sätzen Struktur zuzuweisen. Ein Satz kann in *Konstituenten* aufgeteilt werden und diese können wiederum in kleinere Konstituenten unterteilt werden. So entsteht eine *Konstituentenstruktur*.

Die zweite Möglichkeit, Sätzen Struktur zuzuweisen, besteht darin, syntaktische Relationen zwischen einzelnen Wörtern in einem Satz zu erstellen. Diese Relationen werden als *Dependenzrelationen* bezeichnet. In einer solchen Dependenzrelation bildet ein Wort den *Kopf* und ein anderes den *Dependenten*, wobei der Kopf den Dependenten regiert. Generell ist der Dependent entweder Modifizierer, Objekt oder Komplement des Kopfes und hängt von diesem ab. Wenn ein Kopf ein Dependent eines anderen Kopfes ist, entsteht eine komplexe hierarchische *Dependenzordnung*. Die entstehende syntaktische Struktur wird als *Dependenzstruktur* bezeichnet und kann als *Dependenzbaum* dargestellt werden. Die Wortstellung spielt dabei keine Rolle, jedoch kann die Struktur um eine Wortstellungsbeschreibung erweitert werden. Im Verhalten zweier Wörter einer Dependenzrelation spielt der Kopf die größere Rolle. Ein Dependent, der seinem Kopf vorausgeht, wird *Prädependent* genannt; ein Dependent, der seinem Kopf folgt, wird dementsprechend als *Postdependent* bezeichnet. Ein Wort, das kein Dependent eines anderen Wortes ist, und daher keinen Kopf besitzt, wird als *Independent* bezeichnet. Ein Wort im Satz, für gewöhnlich das finite Hauptverb, ist der Kopf des gesamten Satzes. Jedes andere Wort steht in einer Dependenzrelation zu irgendeinem Kopf und kann selber Kopf einer bestimmten Anzahl von Dependenten sein ([12], S.1-4). Für wohlgeformte Dependenzstrukturen existieren vier Richtlinien, die in vielen Arbeiten im Bereich der maschinellen Sprachverarbeitung angenommen werden ([44], S.519):

(3)
i. one and only one word is independent, that is, not linked to some other word;
ii. all others depend directly on some word;
iii. no word depends on more than one other; and

iv. if a word A depends directly on word B, and some word C intervenes between them (in linear order), then C depends directly on A or on B, or on some other intervening word.

Diese Richtlinien sind insbesondere für die Verbindung von Chunking und syntaktischem Tagging relevant.

2.1.2 Flache Dependenzstruktur

Syntaktische Dependenzrelationen werden in einem Satz mit Hilfe von Tags gekennzeichnet. Die Anzahl der verschiedenen Tags ist dabei begrenzt. Jedes Tag stellt eine spezielle Dependenzrelation dar. Wörter können als Subjekt (`@SUBJ`), direktes Objekt (`@OBJ`), indirektes Objekt (`@I-OBJ`) oder Adverbial (`@ADVL`) usw. getaggt werden, sie alle sind Dependenten eines finiten Verbs. Aus diesem Grund bezeichnet Karlsson in [29] seine Syntax als funktional. Das unabhängige finite Verb eines Hauptsatzes oder eines Nebensatzes wird mit einem speziellen Tag für finit (`@+FMAINV`) und infinit (`@-FMAINV`) gekennzeichnet. Das finite Verb des Hauptsatzes ist dabei nach (3-i) das einzige Wort, das als unabhängig zu interpretieren ist. Alle anderen Wörter hängen nach (3-ii) von einem anderem Wort ab und müssen syntaktisch getaggt sein. Falls ein Verbkompositum vorliegt, werden unterschiedliche Tags für jeden einzelnen Bestandteil aus dem Kompositum verwendet (`@+FAUXV`, `@-FAUXV`). Diese bis hier aufgeführten Tags werden *Kopf-Funktionen* genannt. Dependenten können auch als Adjunkt oder Komplement mit einem speziellen Tag gekennzeichnet werden. Diese Tags können die Kategorie des Dependenten und müssen die Kategorie des Kopfes angeben und geben darüber Auskunft, ob es sich um einen Prädependenten (>) oder Postdependenten (<) handelt. Wörter können als prä-

nominales Adjektiv (@AN>), pränominaler Determinator (@DN>), postnominales Adjunkt (@<NOM) oder als Komplement einer Präposition (@<P) usw. getaggt werden. Diese Tags werden *Modifizierer-Funktionen* genannt. Die Analyse ist flach, da keine Bäume oder hierarchische Strukturen generiert werden und beinhaltet eine Wortstellungsbeschreibung ([29], Kapitel 5).

In der Regel gibt es in einem einfachen Satz nur ein einziges Hauptverb und ebenso gibt es nur ein Subjekt und nur eine bestimmte Anzahl von Objekten. Daher soll der Begriff des *Uniqueness Principle* eingeführt werden ([28], S.4):

(4) *Uniqueness Principle:*
Certain verb chain and head labels may occur maximally once in a simplex clause.

Für das Uniqueness Principle müssen jegliche Haupt- und Nebensatzgrenzen markiert sein. Für diese Markierungen dient das Tag <**CLB> ([28], S.4).

2.1.3 Darstellen von morphologischen Ambiguitäten

Die morphologischen Eigenschaften eines Wortes werden durch Tags dargestellt. In den folgenden Beispielen wird hierfür das ENGCG Tagset verwendet ([21], S.309f.). Wenn ein Wort morphologisch ambig ist, werden seine Lesarten *Kohorte* genannt. Jede Lesart enthält zusätzlich Informationen über die potentiellen syntaktischen Funktionen des betreffenden Wortes im Satz ([28], S.2):

(5)	move	N NOM SG	@SUBJ @OBJ @I-OBJ
		V SUBJUNCTIVE	@+FMAINV
		V IMP	@+FMAINV
		V INF	@-FMAINV

Das Beispiel (5) zeigt das Wort *move* mit der dazugehörigen *Kohorte*. Tags in Großbuchstaben markieren morphologische Merkmale, ein vorangestelltes @ kennzeichnet ein syntaktisches Tag. Die syntaktischen Tags werden hier wie morphologische Eigenschaften eines Wortes behandelt. *move* im Beispiel (5) hat vier verschiedene Lesarten und zu diesen jeweils eine oder mehrere potentielle syntaktische Funktionen. Die Kohorte hat demzufolge die Größe vier.

2.1.4 Darstellen von Attachment-Ambiguitäten

Bei der syntaktischen Analyse werden morphologische Informationen, Satzgrenzeninformationen und Wortabfolgeinformationen auf syntaktische Tags abgebildet. Attachment-Ambiguitäten werden dadurch dargestellt, dass ein Wort mehrere alternative syntaktische Tags erhält. Ein Wort kann nach (3-iii) nicht von mehreren Wörtern abhängen, daher ist die Anbindung ambig. Ein Beispiel für eine Attachment-Ambiguität ist der folgende Satz ([28], S.5):

(6)

Bill	N NOM SG	@SUBJ
saw	V PAST	@+FMAINV
the	DET	@DN>
little	A	@AN>
dog	N NOM SG	@OBJ
in	PREP	@<NOM @ADVL
the	DET	@DN>
park	N NOM SG	@<P

In (6) ist die Präposition *in* mit einer Modifizierer-Funktion und einer Kopf-Funktion getaggt und kann entweder als Adverbial vom Hauptverb abhängen (@ADVL) oder als Postdependent vom nächsten Nomen *dog* (@<NOM). Attachment-Ambiguitäten sind in dieser Repräsentation sehr kompakt dargestellt. Beim syntaktischen Tagging muss die Richtlinie (3-iii) nicht unbedingt gelten, es können theoretisch auch überkreuzte Abhängigkeiten dargestellt werden.

2.1.5 Disambiguierung durch Constraints

Für die Disambiguierung werden Kontextinformationen herangezogen:

(7)

a	DET INDEF	@DN>
move	N NOM SG	@SUBJ @OBJ @I-OBJ

In (7) kann intuitiv durch den unbestimmten Artikel *a* die Größe der Kohorte von *move* (vgl. (5)) auf eins reduziert werden, da auf einen Artikel in der Regel ein Nomen folgt. *move* kann hier durch Kontextinformation zwar morphologisch, jedoch nicht syntaktisch disambiguiert werden, da der Artikel nichts über die potentielle Funktion von *move* im Satz aussagt. Für eine weitere Disambi-

guierung muss ein weiterer Kontext herangezogen werden. Durch die Disambiguierung werden letztlich nicht wohlgeformte Abfolgen von Wörtern entfernt.

Satzgrenzen von Nebensätzen werden zunächst überall angenommen. Diese können dann durch Kontextinformationen disambiguiert werden. Die syntaktische und die morphologische Disambiguierung verlaufen dabei parallel. Um diese Disambiguierungen zu realisieren, verwendet Karlsson in [28] Constraints. Ein Constraint verwendet Kontextinformationen, um bestimmte morphologische und syntaktische Lesarten auszuschließen oder nur eine Lesart zuzulassen. Der Fokus des Constraints liegt hierbei auf einem Wort und dessen Eigenschaften. Die Constraints können auf die Wortform und auf die morphologischen und syntaktischen Tags Bezug nehmen. Es könnten folgende Constraints gelten:

(8)
a. Eine ambige Wortform, die `N` als eine von mehreren Lesarten hat, ist ein Nomen, wenn dieser ein Artikel vorangeht.
b. Eine Nebensatzgrenze muss vor einer bestimmten Wortform erscheinen, wenn diese eindeutig ein finites Verb ist und ein eindeutiges finites Verb exakt vor diesem Wort steht.
c. Die syntaktische Funktion eines Nomens ist `@SUBJ`, wenn ein finites Verb folgt und kein anderes Nomen dazwischenliegt.

Durch die Abstraktion über die morphologischen Tags konnte in (7) durch den Constraint in (8-a) das Wort *move* in (5) disambiguiert und somit eine Lesart als einzige angenommen werden.

2.1.6 Analyse durch eine Constraint-Grammatik

Constraints können als Regeln aufgefasst werden, die sich auf lineare Abfolgen beziehen. Eine Menge von Constraints können zu einer *Constraint-Grammatik* zusammengefasst werden.[2] Constraints einer Grammatik werden in drei Typen gegliedert ([29], S.12):

(9)
- Constraints für kontextabhängige morphologische Disambiguierung
- Constraints für die Festsetzung von Nebensatzgrenzen
- Constraints für die Disambiguierung von syntaktischen Funktionen

Constraints werden verwendet, um alternative syntaktische und morphologische Lesarten zu eliminieren und Satzgrenzen festzusetzen. Das Verfahren ist also *reduktionistisch*. Die Anfangssituation einer Analyse ist ein morphologisch analysierter Satz, in dem alle Lesarten eines Wortes mit potentiellen syntaktischen Funktionen getaggt sind. Die Eingabe ist also strukturell höchst ambig. Eine erfolgreiche Analyse eliminiert alle strukturellen Ambiguitäten, macht einen Satz morphologisch und syntaktisch unambig und enthält eine wohlgeformte Abfolge. Eine nicht erfolgreiche Analyse belässt die Eingabe. Eine falsche Analyse eliminiert alle wohlgeformten Lesarten und belässt nicht wohlgeformte. Dieser Fall ist ersichtlich nicht wünschenswert und das Resultat einer mangelhaften Constraint-Grammatik. Der Übergang von einer erfolgreichen zu einer nicht erfolgreichen oder sogar falschen Analyse ist hierbei fließend ([29], S.13f.).

Das Verfahren des syntaktischen Taggings hat nicht den Anspruch, zu entscheiden, ob ein Satz in einer Sprache korrekt ist. Es wird soweit wie möglich

[2] Karlsson gibt in [29] einen Formalismus für Constraint-Grammatiken an. Auf diesen Formalismus wird hier nicht weiter eingegangen.

disambiguiert, die Analyse scheitert dabei nicht, wenn Satzteile nicht disambiguiert werden können. Demzufolge können Analysen unspezifiziert bleiben. Es ist folglich keine Constraint-Grammatik notwendig, die alle Sätze disambiguieren bzw. vollständig abdecken kann.

Constraints können auf der Basis von Korpusstudien erstellt werden und gleichen regelähnlichen Fakten. Die Theorie ist sprachunabhängig, da Constraint-Grammatiken theoretisch für jede Sprache anhand von Korpora erstellt werden können ([29], S.17).

2.1.7 Probleme

Wie schon zu Anfang dieses Kapitels erwähnt, können einige syntaktische Ambiguitäten nicht aufgelöst werden (vgl. (6)). In diesem Fall erhalten Wörter zwei verschiedene syntaktische Tags. Es können jedoch auch Ambiguitäten unaufgelöst bleiben, wenn jedes Wort eines Satzes die Kohortengröße eins hat und mit nur einem syntaktischem Tag versehen ist ([49], S.398f.):

(10) Fat butcher's wife

Das Adjektiv *fat* in (10) ist folgendermaßen getaggt:

(11) fat A @AN>

in (10) kann sich das Adjektiv *fat* entweder auf das Nomen *butcher's* oder auf das Nomen *wife* beziehen, da das Tag @AN> in (11) anzeigt, das *fat* Prädependent des nächsten nominalen Kopfes ist. *butcher's* ist aber nicht unbedingt der nächste nominale Kopf. Dabei ist zu beachten, dass es nach (3-iv) nicht erlaubt ist, dass ein Dependent von mehreren Wörtern abhängt. Es ist also nicht

klar, welches Nomen der Kopf des Adjektivs ist. Das gleiche Problem tritt bei Koordination auf ([36], S.158):

(12) John's brother and aunts

Es wird angenommen, dass das Nomen *John's* in (12) folgendermaßen getaggt ist:

(13) John's N GEN @GN>

Das syntaktische Tag @GN> in (13) zeigt an, dass das Nomen *John's* Prädependent des nächsten nominalen Kopfes ist. Das Nomen *John's* kann also vom Nomen *brother* oder von den koordinierten Nomen *brother and aunts* abhängen. Auch diese Ambiguität bleibt unaufgelöst.

2.2 Chunking

Bei der Definition von Chunks stützt sich Abney [3] auf die Begriffe *funktionales Element, thematisches Element* und *F-Selektion*. Zunächst sollen diese von Abney [1] eingeführten grundlegenden Begriffe erläutert werden. Funktionale Elemente sind durch bestimmte Eigenschaften charakterisiert, die allerdings keine Kriterien für eine Klassifikation darstellen ([1], S.43f.):

(14)
i. Functional elements constitute closed lexical classes.
ii. Functional elements are generally phonologically and morphologically dependent. They are generally stressless, often clitics or affixes, and sometimes even phonologically null.
iii. Functional elements permit only one complement, which is in ge-

neral not an argument. The arguments are CP, PP and [...] DP. Functional elements select IP, VP, NP.

iv. Functional elements are usually unseparable from their complement.

v. Functional elements lack [...] „descriptive content“. Their semantic contribution is second-order, regulating or contributing to the interpretation of their complement. They mark grammatical or relational features, rather than picking out a class of objects.

Elemente, die durch die Eigenschaften in (14) kein funktionales Element darstellen und einen beschreibenden Inhalt haben, werden als thematisches Element bezeichnet. Funktionale und thematische Elemente werden durch das syntaktische Merkmal [$\pm$F] unterschieden. [+F] bezeichnet funktionale Elemente und [-F] thematische Elemente. *F-Selektion* ist die Bezeichnung für die syntaktische Relation zwischen einem funktionalen Element und seinem Komplement ([1], S.39). Im Zusammenhang mit dem Chunking stellen nach (14-iii) Komplementierer, Artikel und Präpositionen funktionale Elemente dar.

Die Kernidee des Chunking ist die Gruppierung von funktionalen Elementen mit ihrem Komplement. Ein Chunk kann wiederum als thematisches Element von einem funktionalen Element selektiert werden und ist dann kein Chunk mehr, sondern Teil eines größeren Chunks. Die Zuordnung eines Wortes zu einem Chunk ist eindeutig, daher sind Überschneidungen und rekursive Einbettungen von Chunks nicht möglich.

2.2.1 Chunk Connectedness

Um den Sachverhalt noch weiter zu vereinfachen und auf die Betrachtung von funktionalen Elementen zu beschränken, ist das Prinzip *Chunk Connectedness* definiert ([3], S.6):

(15) *Chunk Connectedness:*
A functional element defining a chunk must be included in the chunk it defines.

Dieses Prinzip ergibt sich aus der Eigenschaft (14-iv) von funktionalen Elementen. Chunks sind auf diese Weise ausschließlich mit Hilfe der funktionalen Elemente definiert. Thematische Elemente, die nicht durch ein funktionales Element selektiert sind, können grundsätzlich keinen Chunk bilden. Sie müssen Teil eines anderen Chunks sein:

(16) [the big green fish]

Jedoch gibt es thematische Elemente, die von keinem funktionalen Element selektiert und zu keinem anderen Chunk gruppiert werden. In diesem Fall müssen leere funktionale Elemente angenommen werden:

(17) a. [$\emptyset_{Deg}$ less green]
b. [$\emptyset_{Det-pl}$ proud men]

In (17-a) wie auch in (17-b) muss ein funktionales Element angenommen werden, welches *green* bzw. *men* selektiert, um einen Chunk zu bilden. Daher müssen leere funktionale Elemente wie $\emptyset_{Deg}$ oder $\emptyset_{Det}$ eingefügt werden ([3], S.8f.). Dies wird durch die Eigenschaft (14-ii) von funktionalen Elementen

möglich.

Das Prinzip Chunk Connectedness beinhaltet, dass kein funktionales Element zwischen einem anderen funktionalen Element und dessen Komplement auftreten darf, sonst würde ein funktionales Element nicht nur genau zu seinem Komplement gruppiert werden, sondern auch zu dem Komplement eines anderen funktionalen Elements. Aber nach (14-iii) erlaubt ein funktionales Element nur ein einziges Komplement. Vereinfacht gesagt, es darf kein Chunk in einem anderen eingebettet sein. Leere funktionale Elemente sollen daher nur dann angenommen werden, wenn es zu keinen Verletzungen des Prinzips Chunk Connectedness kommt ([3], S.10):

(18) a. *[the[$\emptyset_{Deg}$ big green] fish]
b. *[$\emptyset_{Det}$[a dozen] men]
c. [a dozen][$\emptyset_{Det}$ men]

In (18-a) darf kein leeres funktionales Element angenommen werden, welches *green* selektiert (vgl. (16)). In (18-b) ist durch das leere funktionale Element $\emptyset_{Det}$ das Prinzip *Chunk Connectedness* verletzt, in (18-c) verursacht das leere funktionale Element $\emptyset_{Det}$ dagegen keine Verletzung dieses Prinzips.

2.2.2 Chunk Inclusiveness

Im Folgenden wird gezeigt, dass im Zusammenhang mit dem Prinzip Chunk Connectedness Probleme auftreten können. In manchen Fällen lässt sich das funktionale Element nicht mit seinem Komplement gruppieren.

(19) a. *[that[the man] sang]
b. *[in[the man's] house]

In (19-a) liegt das funktionale Element *the*, das mit seinem Komplement *man* einen Chunk bildet, zwischen dem funktionalen Element *that* und seinem Komplement *sang*. Das Prinzip Chunk Connectedness verbietet jedoch, dass ein Chunk in einem anderen Chunk eingebettet ist. Die gleiche Problematik gilt für (19-b). Das funktionale Element *the*, das mit seinem Komplement *man's* einen Chunk bildet, steht zwischen dem funktionalen Element *in* und dessen Komplement *house*. Die funktionalen Elemente, die nicht zu einem Chunk gruppiert werden können, werden *verwaiste Wörter* (orphanable words) genannt. Um verwaiste Wörter zuzulassen, ist die Bedingung *Chunk Inclusiveness* definiert ([3], S.8):

(20) *Chunk Inclusiveness:*
Whith the exception of a distinguished subset of function words (the 'orphanable' words), every word must belong to some chunk.

Da kein Chunk in einen anderen eingebettet werden darf, ist es also möglich, dass ein funktionales Element als verwaistes Wort zurückbleibt. Mit Hilfe von Chunk Inclusiveness sind nun bestimmte verwaiste funktionale Elemente zulässig:

(21) a. that[the man] [sang]
b. in[the man's] [house]

In (21) wird angenommen, dass *that* und *in* zu der Teilmenge von funktionalen Elementen gehört, die als verwaiste Wörter zurückbleiben dürfen. Exhaustivität ist bei einer Analyse durch Chunks folglich nicht gefordert. Im Zusammenhang mit Chunk Inclusiveness sollen leere funktionale Elemente nur dann angenom-

men werden, wenn sie zu keinem verwaisten funktionalen Element führen.

(22) *[a big[and red] bird]

In (22) stellt die Konjunktion *and* ein Problem dar, wenn sie als funktionales Element zugelassen wird, da in diesem Fall das Prinzip der Chunk Connectedness verletzt wäre. Die Konjunktion kann aber auch kein thematisches Element sein, da sie keinen beschreibenden Inhalt hat. Es muss also angenommen werden, dass Konjunktionen weder funktionale noch thematische Elemente sind. Das bedeutet, dass das Merkmal $[^{+}_{-}\mathrm{F}]$ die syntaktischen Kategorien nicht partitioniert ([3], S.10f.).

(23) [a big and red bird]

In (23) wird also angenommen, dass das Wort *and* weder ein funtionales noch ein thematisches Element ist und daher die Gruppierung des funktionalen Elementes *a* mit seinem Komplement nicht blockiert.

2.2.3 Konstituentenstruktur

Abney nimmt in [3] weiter an, dass Chunks eine interne Struktur haben. Diese Struktur ist eine Konstituentenstruktur. Ein Konstituentenknoten wird durch ein funktionales Element und dessen Komplement gebildet; er dominiert diese Elemente. Das thematische Element, das von einem funktionalen Element selektiert wird, ist der semantische Kopf einer Konstituente. Dieser Kopf projiziert seine semantischen Eigenschaften in den Konstituentenknoten. Ein Konstituentenknoten bildet als *s-Projektion* des semantischen Kopfes eine kategorielle Ebene und kann wiederum durch ein funktionales Element selektiert werden

([3], S.4):

(24) α is an s-projection of β iff

i. $\alpha = \beta$, or

ii. the immediate head of α is an s-projection of β, or

iii. α f-selects an s-projection of β.

Die s-Projektion folgt als Konsequenz aus der Eigenschaft (14-v) von funktionalen Elementen. Funktionale Elemente sind Teil der s-Projektionen der semantischen Köpfe und haben keinen eigenen semantischen Inhalt. Sie dienen lediglich dazu, dass die Komplemente ihren semantischen Inhalt weiterreichen können. Im Sinne der klassischen syntaktischen Projektionen ist der Komplementierer der syntaktische Kopf eines Nebensatzes (CP), die Präposition der syntaktische Kopf einer Präpositionalphrase (PP) und das Nomen Kopf einer Nominalphrase (NP) ([17], S.46, S.143).[3] Die s-Projektion lässt sich aus der klassischen syntaktischen Projektion, die Abney c-Projektion nennt, ableiten ([1], S.39):

(25) α is an s-projection of β iff

i. $\beta = \alpha$, or

ii. β is a c-projection of an s-projection of α, or

iii. β f-selects an s-projection of α

Demnach ist das Verb der semantische Kopf eines Satzes, das Nomen semantischer Kopf einer Nominalphrase oder Präpositionalphrase und das Adjektiv se-

[3] Abney nimmt in [1] eine DP-Analyse an. Der Determinator ist dann Kopf einer DP und nimmt eine NP als Komplement. Hier wird keine DP-Analyse angenommen, um die Verbindung mit dem syntaktischen Tagging zu vereinfachen.

mantischer Kopf der Adjektivphrase. Der semantische Kopf ist demnach nicht unbedingt gleich dem syntaktischen Kopf. Nach (25) können die Konstituentenknoten eines Chunks aber ohne Probleme nach den Projektionen der syntaktischen Köpfe benannt werden. Die maximale s-Projektion, d. h. die Projektion des prominentesten Wortes im Chunk, wird *Chunk-Ceiling* genannt. Mit diesem Begriff lässt sich auch aus diesem Blickwinkel ein Chunk definieren ([3], S.4):

(26) a chunk is the maximal subgraph of a chunk-ceiling C which:

i. includes the thematic element defining C
ii. does not contain any other chunk-ceiling, and
iii. has a connected frontier

Die Kategorie eines Chunks ist letztlich durch den Chunk-Ceiling definiert. Der Satz *that the man loves the tent in the big garden* würde folgendermaßen analysiert werden, wenn die interne Struktur mit *labeled bracketing*[4] dargestellt wird:

(27) that $[_{NP}\ [_{DET}$ the$]\ [_{N}$ man$]]\ [_{VP}\ [_{V}$ loves$]]\ [_{NP}\ [_{DET}$ the$]\ [_{N}$ tent$]]\ [_{PP}\ [_{P}$ in$]\ [_{NP}\ [_{DET}$ the$][_{ADJ}$ big$][_{NP}$ garden$]]]$

Im folgenden Abschnitt wird erläutert, wie sich diese lokalen Parse-Bäume zu einem globalen Parse-Baum zusammenfügen lassen.

[4] Das labeled bracketing ist eine alternative Darstellungsform zu Baumdiagrammen. Einzelne Kategorien werden jeweils in eckige Klammern eingeschlossen, die öffnende Klammer erhält die Kategorienbezeichnung als Subskript. Mit dieser Darstellungsform ist es möglich, nur bestimmte Teile einer Struktur zu kennzeichnen ([17], S.29f.).

2.2.4 Chunks als Teil eines Parse-Baums

Chunks sind keine Konstituenten im Sinne klassischer Konstituentenstrukturen, wie das Beispiel *this student of physics* zeigt (vgl. [17], S.50):

(28) a. $[_{NP}\ [_{DET}$ this$][_{N}$ student$]][_{PP}$ of physics$]$

b. $[_{NP}\ [_{DET}$ this$][_{N'}\ [_{N}$ student$][_{PP}$ of physics$]]]$

In (28-a) ist die Chunk-Struktur des Ausdrucks *this student of physics* dargestellt. Hierbei ist es nicht relevant, ob die c-Projektionen oder s-projektionen betrachtet werden, da sie identisch sind. Der NP-Chunk *this student* bildet keine Konstituente im Sinne der klassischen Konstituentenstruktur, die in (28-b) dargestellt ist. Chunks können jedoch zu einer klassischen Konstituente wie in (28-b) zusammengefügt werden. Dabei werden zwei Chunks miteinander verbunden, indem fehlende Verbindungen zwischen Konstituentenknoten innerhalb der Chunks eingefügt werden. Dieser Mechanismus wird *Attacher* genannt ([3], S.5). In (28-a) muss eine Verbindung zwischen dem Chunk-Ceiling des PP-Chunks und dem N-Knoten von *student* hergestellt werden, um die Struktur in (28-b) zu erhalten. Dabei muss eine weitere Projektionsebene hinzugefügt werden. Durch dieses Verfahren kann aus einem Satz, der durch Chunks analysiert ist, ein globaler Phrasenstrukturbaum erstellt werden. Allgemein ist ein Chunk ein potentieller lokaler Baum eines globalen Parse-Baums eines Satzes.

2.2.5 Probleme

Das Prinzip Chunk Connectedness wurde von Abney in [3] für das Englische entwickelt. Es ist fraglich, ob dieses Prinzip für alle Sprachen gilt. Durch das Prinzip Chunk Inclusiveness, durch die flexible Definition von funktionalen und

thematischen Elementen und durch die Möglichkeit, leere funktionale Elemente anzunehmen, ist das Prinzip Chunk Connectedness jedoch sehr flexibel.

Im Folgenden soll Chunk Connectedness auch auf das Deutsche angewendet werden. Im Deutschen kann zwischen einem Determinator und seinem Komplement ein komplexes pränominales Adjunkt stehen. Ein Beipiel für solche Konstruktionen ist der Ausdruck *die auf der Rückseite beschriebene Aufgabe* ([3], S.11f.). Solche Konstruktionen verletzen jedoch unter bestimmten Annahmen nicht Chunk Connectedness:

(29) die [auf der Rückseite] [Ø_{Comp} beschriebene] [Aufgabe]

In (29) bleibt der Determinator *die* durch das Prinzip Chunk Inclusiveness ein verwaistes Wort. Für das Partizip II *beschriebene* muss das leere funktionale Element Ø_{Comp} angenommen werden. Das Beipiel zeigt, daß im Deutschen zwar das Prinzip Chunk Connectedness gilt, jedoch werden durch viele verwaiste Wörter viele Chunks gebildet, die ihr funktionales Element nicht beinhalten. Die verwaisten Wörter können zudem weit von ihrem Chunk, den sie definieren, entfernt stehen. Dabei können wie in (29) zwischen dem funktionalen Element und seinem Komplement mehrere Chunks liegen. Ein ähnliches Beispiel stellt der Nebensatz *dass Hans Eva liebt* dar:

(30) dass [Ø_{Det} Hans] [Ø_{Det} Eva] [liebt]

In (30) stehen zwischen dem Komplementierer und dem selektierten Verb zwei Chunks. Das resultiert daraus, dass das Deutsche in Nebensätzen eine Sprache vom Typ SOV ist ([9], S.27).

Atachment-Ambiguitäten treten nur zwischen und nicht innerhalb von

Chunks auf. Ein Chunk ist daher ein nicht ambiger lokaler Baum eines globalen Parse-Baums eines Satzes. Die Behandlung von morphologischen Ambiguitäten fehlt jedoch bei der Definition von Chunks.

2.3 Verbindung von Chunking und syntaktischem Tagging

Es soll diskutiert werden, inwieweit sich das Verfahren des syntaktischen Taggings und das Verfahren des Chunkings zusammenbringen lassen.

2.3.1 Chunks als lokale Dependenzbäume

Es existiert ein hoher Grad an Isomorphie zwischen der Dependenzstruktur des syntaktischen Taggings und einer syntaktischen Konstituentenstruktur. Daher kann eine Konstituentenstruktur aus syntaktischen Tags inferiert werden ([29], S.33, S.37). Eine Konstituente besteht dann aus einem Wort und all seinen Dependenten und deren Dependenten etc. ([11], S.2). Eine Eigenschaft von Konstituenten ist die Kontinuität. Demzufolge muss (3-i) gelten, d. h. es dürfen keine Kreuzabhängigkeiten in der Dependenzstruktur enthalten sein. Hier soll die Umformung einer Konstituentenstruktur in eine Dependenzstruktur betrachtet werden. Die Konstituentenstruktur von Chunks spiegelt sowohl die c-Projektionen als auch die s-Projektionen wieder. Es interessieren in diesem Zusammenhang die c-Projektionen. Um eine Dependenzstruktur aus der Konstituentenstruktur von Chunks zu inferieren, soll folgende Umformung angewendet werden:

(31) Ein Kopf α regiert seinen Dependenten β, wenn

i. α syntaktischer Kopf einer Konstituente ist und β sein Komplement.

Da Konstituenten kontinuierlich sind, genügt die entstehende Struktur der Richtlinie (3-iv). In den entstandenen lokalen Dependenzbäumen können also keine Kreuzabhängigkeiten auftreten (vgl. [9], S.37f.). Der syntaktische Kopf eines Chunks, der Chunk-Ceiling, wird zur Wurzel der lokalen Dependenzbäume. Er ist independent und genügt der Richtlinie (3-i). Alle weiteren Wörter in einem Chunk hängen von einem anderen Wort ab, da sie Komplement eines syntaktischen Kopfes sind, daher ist die Richtlinie (3-ii) erfüllt. Da eine Konstituente nur einen Kopf hat, lässt sich aus (31) erschließen, dass die Richtlinie (3-iii) erfüllt ist. Ein Chunk beinhaltet folglich nicht nur eine Konstituentenstruktur, sondern auch eine Dependenzstruktur. Ein Chunk kann nicht nur als lokaler Konstituentenbaum eines globalen Konstituenten-Parse-Baums angesehen werden, sondern auch als ein lokaler Dependenzbaum eines globalen Dependenz-Parse-Baums. Um diese lokalen Dependenzbäume in die flache dependenz-orientierte funktionale Syntax des syntaktischen Taggings im Sinne eines Attachers einzugliedern, kann einem Chunk-Ceiling und somit einem Chunk ein syntaktisches Tag zugewiesen werden. Auf diese Weise können bei der disambiguierenden Analyse des syntaktischen Taggings für einen Chunk-Typ bestimmte syntaktische Funktionen angenommen werden. Diese werden dann durch syntaktische Constraints gefiltert. Eine Analyse des Satzes *Bill saw the little dog in the park* würde dann folgende Form haben:

(32)	[$Ø_{Det}$ Bill]	N	@SUBJ
	[$Ø_{Comp}$ saw]	V PAST	@+FMAINV
	[the little dog]	N NOM SG	@OBJ
	[in the park]	PREP	@<NOM @ADVL

In (32) betreffen die aufgeführten morphologischen und syntaktischen Eigenschaften den Chunk-Ceiling eines Chunks. Durch dieses Verfahren können auch verwaiste Wörter durch syntaktische Tags an die Chunks gebunden werden, die sie definieren.

Es soll ein Beispiel für das Deutsche gegeben werden. Es wird hierfür das STTS (Stuttgart-Tübinger Tagset) verwendet, wobei für das Partizip II das Tag VVPP2 angenommen wird.[5] Die Analyse des Satzes *die auf der Rückseite beschriebene Aufgabe* hat hierbei folgende Form:

(33)	die	ARTDEF	@DN>
	<**CLB>		
	[auf der Rückseite]	APPR	@ADVL
	[$Ø_{Comp}$ beschriebene]	VVPP2	@-FMAINV
	<**CLB>		
	[Aufgabe]	NN	@SUBJ @OBJ @I-OBJ

In (33) ist der verwaiste Artikel *die* durch das syntaktische Tag @DN> an seinen Kopf *Aufgabe* gebunden. Im Abschnitt (2.2.5) wurde gezeigt, dass es im Deutschen zu vielen verwaisten Wörter mit großem Abstand zum Komplement kommen kann. Durch das syntaktische Tagging kann diese entstehende Lücke geschlossen werden.

[5] www.sfs.nphil.uni-tuebingen.de/Elwis/stts/stts-guide.ps.gz

2.3.2 Probleme

Morphologische Ambiguitäten werden bei einer Analyse durch Chunks nicht behandelt, da sie für Chunks nicht definiert sind. Es muss daher angenommen werden, dass ein Satz keine morphologischen Ambiguitäten enthält. An diesem Punkt scheitert die Symbiose des Cunkings und des syntaktischen Taggings. Es wäre hierbei wünschenswert, beim Chunking morphologische Ambiguitäten behandeln und bestenfalls auflösen zu können.

3 Reguläre Sprachen und Formal Power Series

Die in Kapitel (2) eingeführten Linguistischen Strukturen sollen formal gefasst werden. In diesem Kapitel wird ein kurzer Überblick über grundsätzliche Konzepte gegeben, die zur Darstellung und Generierung dieser Strukturen in Späteren Kapiteln von Bedeutung sind.

Als erstes soll der Begriff des Semirings und der Zeichenketten genauer erläutert und auf reguläre Sprachen und Relationen eingegangen werden. Dann wird auf den Begriff der *Formal Power Series* und mit diesen verbundenen gewichtete reguläre Sprachen und Relationen eingegangen.

3.1 Monoide und Semiringe

Es soll hier die algebraische Struktur des Monoiden und die algebraische Struktur des Semirings kurz erläutert werden. Ein Monoid beschreibt eine Operation auf einer Menge mit bestimmten Eigenschaften. Ein Semiring dagegen besteht aus zwei Monoiden und beinhaltet daher zwei Operationen auf einer Menge, die wiederum bestimmte Eigenschaften erfüllen müssen. Die algebraische Struktur eines Monoiden ist folgend definiert:

(1) Ein Monoid ist ein Tripel $(M, \oplus, \bar{0})$, das (i)-(iii) erfüllt.

i. M ist eine Menge.

ii. $\oplus$ ist eine zweistellige Funktion auf M.
$\oplus$ ist assoziativ: $a \oplus (b \oplus c) = (a \oplus b) \oplus c$ für alle $a, b \in M$

iii. $\bar{0} \in M$
$\bar{0}$ ist neutrales Element bezüglich $\oplus$: $a \oplus \bar{0} = a = \bar{0} \oplus a$ für alle $a, b \in M$

Ein Monoid $(M, \oplus, \bar{0})$ heißt kommutativ, wenn $a \oplus b = b \oplus a$ für alle $a, b \in M$ gilt ([37], S.5). Ein Element $a \in$ M heißt idempotent, wenn $a \oplus a = a$ gilt. Wenn alle Elemente $a \in M$ diese Eigenschaft haben, ist $(M, \oplus, \bar{0})$ ein idempotenter Monoid. Wenn $(M, \oplus, \bar{0})$ ein kommutativer idempotenter Monoid ist, ist eine partielle Ordnung $\leq_M$, eine natürliche Ordnung über M, durch $a \leq_M b \Leftrightarrow a \oplus b = a$ definiert (vgl. [24], S.58f.).

Ein Semiring besteht aus zwei Monoiden $(S, \oplus, \bar{0})$ und $(S, \otimes, \bar{1})$. Der Monoid $(S, \oplus, \bar{0})$ muss hierbei kommutativ sein, es muss das Distributivgesetz gelten und es müssen bestimmte Eigenschaften bezüglich der neutralen Elemente der Monoide gelten:

(2) Ein Semiring ist ein Quintupel $(S, \oplus, \otimes, \bar{0}, \bar{1})$, das (i)-(iii) erfüllt.

i. $(S, \oplus, \bar{0})$ und $(S, \otimes, \bar{1})$ sind Monoide.
$\bar{0}$ ist Annulator bezüglich $\otimes$: $a \otimes \bar{0} = \bar{0} = \bar{0} \otimes a$ für alle $a \in S$

ii. $\oplus$ ist kommutativ: $a \oplus b = b \oplus a$ für alle $a, b \in S$

iii. $\otimes$ distributiert über $\oplus$: $a \otimes (b \oplus c) = a \otimes b \oplus a \otimes c$ und $(b \oplus c) \otimes a = b \otimes a \oplus c \otimes a$ für alle $a, b, c \in S$

Ein Semiring $(S, \oplus, \otimes, \bar{0}, \bar{1})$ ist kommutativ, wenn $(M, \otimes, \bar{0})$ kommutativ ist, additiv idempotent wenn $(S, \oplus, \bar{0})$ idempotent ist und multiplikativ idempotent, wenn $(S, \otimes, \bar{1})$ idempotent ist ([24], S.10, S.13). In den folgenden Kapiteln werden gerade die additiv idempotenten Semiringe interessant sein, um partielle und insbesondere lineare Ordnungen definieren zu können.

3.2 Zeichenketten

Eine Menge von Zeichenketten kann aus einem Alphabet Σ gebildet werden. Die Elemente, die dieses Σ enthält, sind Zeichen. Eine endliche Sequenz von diesen Zeichen bildet eine Zeichenkette. Die Größe von Σ kann hierbei begrenzt oder abzählbar unendlich sein. Die Menge aller Zeichenketten, die aus dem Alphabet $\Sigma \neq \emptyset$ gebildet werden kann, ist die Menge Σ^+:

$$\Sigma^+ = \bigcup_{n \in \mathbf{N}} \{(x_1, ..., x_n) | x_i \in \Sigma\} \tag{3}$$

Es soll der freie Monoid $(\Sigma^+ \cup \{\epsilon\}, \cdot, \epsilon)$ = $(\Sigma^*, \cdot, \epsilon)$ betrachtet werden. Das neutrale Element ϵ wird als leeres Wort interpretiert. Für alle $w \in \Sigma^*$ gilt $w \cdot \epsilon = \epsilon \cdot w = w$. Die Multiplikation ist als Konkatenation auf der Menge Σ^* definiert:

$$(x_1, ..., x_n) \cdot (y_1, ..., y_n) = (x_1, ..., x_n, y_1, ..., y_n) \tag{4}$$

Diese Operation ist assoziativ. Durch $(x_v) = x_v$ für alle $x_v \in \Sigma$ und Weglassen von Multiplikationspunkten kann folgend vereinfacht werden:

$$(x_1, x_2..., x_n) = (x_1) \cdot (x_2) \cdot ... \cdot (x_n) = x_1 x_2 ... x_n \tag{5}$$

Die Multiplikation hat daher die Form:

(6) $(x_1...x_n) \cdot (y_1...y_n) = (x_1...x_n y_1...y_n)$

Dieser freie Monoid ist nur für $|\Sigma|$ = 1 kommutativ, da sonst $x_1 x_2 \neq x_2 x_1$ für alle $x_1 \neq x_2$ aus Σ gilt. Dieser Monoid wird abkürzend auch als freier Monoid Σ^* bezeichnet ([24], S.244f.).

Im Falle eines begrenzten Σ werden Teilmengen von Σ^* formale Sprachen über Σ genannt. Die Zeichenketten einer formalen Sprache werden als Wörter über Σ bezeichnet. Die Länge eines Wortes *w*, die mit $|w|$ bezeichnet wird, ist die Anzahl von Zeichen aus Σ, die in *w* auftreten. Dabei gilt $|\epsilon|$ = 0. Eine beliebige Anzahl von führenden Zeichen eines Wortes wird als Präfix bezeichnet, eine Folge von Zeichen am Ende eines Wortes als Suffix ([25], S.1). Mit $rev(w)$ soll das umgekehrte Wort bezeichnet werden.

3.3 Reguläre Sprachen

Eine formale Sprache über Σ ist eine beliebige Menge von Wörtern über Σ aus Σ^*. Eine formale Sprache ist also ein Element der Potenzmenge $\mathcal{P}(\Sigma^*)$. Im Folgenden soll die Menge aller formalen Sprachen über Σ, also $\mathcal{P}(\Sigma^*)$, als $\mathcal{L}_\Sigma$ bezeichnet werden ([24], S.250).

Über die Menge $\mathcal{L}_\Sigma$ sind die Operationen *Konkatenation* und *Vereinigung* definiert. Wenn $L_1 \subseteq \mathcal{L}_\Sigma$ und $L_2 \subseteq \mathcal{L}_\Sigma$ zwei Mengen von Wörtern sind, dann ist die Konkatenation von L_1 und L_2 die Menge

(7) $L_1 \cdot L_2 = \{u \cdot v | u \in L_1 \text{ und } v \in L_2\}$

und die Vereinigung von L_1 und L_2 die Menge

(8) $L_1 \cup L_2 = \{u | u \in L_1 \text{ oder } u \in L_2\}$.

Die Operationen Konkatenation und Vereinigung über der Menge $\mathcal{L}_\Sigma$ bilden den Semiring $(\mathcal{P}(\Sigma^*), \cup, \cdot, \emptyset, \{\epsilon\})$ mit der leeren Sprache $\emptyset$ als absorbierendes Nullelement und dem leeren Wort $\{\epsilon\}$ als Einselement ([24], S.250).

Der Semiring $(\mathcal{P}(\Sigma^*), \cup, \cdot, \emptyset, \{\epsilon\})$ ist bezüglich der *Sternhülle* und damit auch der *Plushülle* vollständig. Wenn $L \subseteq \Sigma^*$ eine Menge von Wörtern ist, und $L^0 = \{\epsilon\}$ und $L^i = LL^{i-1}$ für $i \geq 1$ definiert ist, ist die Sternhülle L^* die Menge

$$L^* = \bigcup_{i \geq 0} L^i \tag{9}$$

und die Plushülle L^+ die Menge

$$L^+ = \bigcup_{i \geq 1} L^i \tag{10}$$

Hierbei existiert t^* für alle $t \in \mathcal{L}_\Sigma$ mit $t^* \in \mathcal{L}_\Sigma$ ([24], S.238; [46], S.2f.; [25], S.29).

Es interessieren im Folgenden die regulären Sprachen. Die Menge aller endlichen Sprachen soll mit $\mathcal{E}_\Sigma$, bezeichnet werden, wobei $\mathcal{E}_\Sigma \subseteq \mathcal{L}_\Sigma$ gilt. Eine Sprache über dem Alphabet Σ ist regulär, wenn sie aus endlich vielen Schritten der Form $L_1 \cup L_2$, $L_1 \cdot L_2$, L^+ und L^* mit $L_1, L_2, L \subseteq \mathcal{E}_\Sigma$ gebildet werden kann ([24], S.251). Die Menge der regulären Sprachen soll mit $\mathcal{R}_\Sigma$ bezeichnet werden.

Die Menge $\mathcal{R}_\Sigma$ ist bezüglich der Sternhülle und somit auch bezüglich der Plushülle abgeschlossen, da für $t \in \mathcal{L}_\Sigma \cap \mathcal{R}_\Sigma$ stets t^+ und t^* existiert mit $t^+ \in \mathcal{R}_\Sigma$ und $t^* \in \mathcal{R}_\Sigma$ ([24], S.242).

$\mathcal{R}_\Sigma$ ist außerdem unter der *Schnittbildung*, *Subtraktion* und *Komplementbildung* abgeschlossen ([25], S.63). Wenn $L_1 \subseteq \mathcal{R}_\Sigma$ und $L_2 \subseteq \mathcal{R}_\Sigma$ zwei reguläre

Sprachen sind, dann ist die Schnittbildung von L_1 und L_2 die Menge

(11) $L_1 \cap L_2 = \{u | u \in L_1 \text{ und } u \in L_2\}$

und die Subtraktion von L_1 und L_2 die Menge

(12) $L_1 - L_2 = \{u | u \in L_1 \text{ und } u \notin L_2\}$

und die Komplementbildung von $L = L_1$ die Menge

(13) $\overline{L} = \Sigma^* - L$

([46], S.2f.; [25], S.5, S.63).

Reguläre Sprachen sind unter der *Umkehrung* abgeschlossen. Die Umkehrung einer regulären Sprache bezeichnet die Umkehrungen aller Wörter einer regulären Sprache. Wenn $L \subseteq \mathcal{R}_\Sigma$ eine reguläre Sprache ist, ergibt die Umkehrung die Menge

(14) $Rev(L) = \{rev(u) | u \in L\}$

(vgl. [27], S.340f.).

3.4 Reguläre Relationen

Eine reguläre Relation ist eine Abbildung von einer regulären Sprache auf eine reguläre Sprache. Eine reguläre Relation besteht demnach aus geordneten Paaren von Wörtern aus $\mathcal{R}_\Sigma \times \mathcal{R}_\Sigma$.[6]

[6] Wenn $L_1 \subseteq \mathcal{R}_\Sigma$ und $L_2 \subseteq \mathcal{R}_\Sigma$ zwei reguläre Sprachen sind und $L_1 \neq \emptyset \neq L_2$ gilt und wenn eine Bijektion $\sigma : L_1 \rightarrow L_2$ existiert, also wenn L_1 und L_2 die gleiche Kardinalität haben, dann ist eine reguläre Relation isomorph ([25], S.6).

Reguläre Relationen können aus regulären Sprachen mit Hilfe des *Kartesischen Produkts* gebildet werden. Wenn $L_1 \subseteq \mathcal{R}_\Sigma$ und $L_2 \subseteq \mathcal{R}_\Sigma$ zwei reguläre Sprachen sind, ist deren Kartesisches Produkt die Menge

(15) $L_1 \times L_2 = \{\langle u, v\rangle | u \in L_1 \text{ und } v \in L_2\}$

(vgl. [25], S.6).

Es ist auch möglich, durch die *Identitätsrelation* eine reguläre Sprache in eine reguläre Relation zu überführen. Wenn $L \subseteq \mathcal{R}_\Sigma$ eine reguläre Sprache ist, ist die Identitätsrelation zu L die Menge

(16) $Id(L) = \{\langle u, u\rangle | u \in L\}$

(vgl. [27], S.341).

Die erste reguläre Sprache einer regulären Relation bezeichnet den *Definitionsbereich*, die zweite den *Wertebereich* ([25], S.7). Mit Dom und $Range$ kann auf den Definitions- und den Wertebereich Bezug genommen werden. Wenn $R \subseteq \mathcal{R}_\Sigma \times \mathcal{R}_\Sigma$ eine reguläre Relation ist, ist der Wertebereich die Menge

(17) $Dom(R) = \{u | \langle u, v\rangle \in R\}$

und der Definitionsbereich die Menge

(18) $Range(R) = \{v | \langle u, v\rangle \in R\}$

(vgl. [27], S.340f.).

Durch die *Invertierung* einer regulären Relation können der Definitions- und Wertebereich vertauscht werden. Wenn $R \subseteq \mathcal{R}_\Sigma \times \mathcal{R}_\Sigma$ eine reguläre Relation ist, ist die Invertierung von R die Menge

(19) $R^{-1} = \{<v, u>|<u, v> \in R\}$

([27], S.340f.).

Reguläre Relationen sind wie reguläre Sprachen unter Konkatenation, Vereinigung, Plushülle, Sternhülle und der Umkehrung abgeschlossen. Wenn $R_1 \subseteq \mathcal{R}_\Sigma \times \mathcal{R}_\Sigma$ und $R_2 \subseteq \mathcal{R}_\Sigma \times \mathcal{R}_\Sigma$ zwei reguläre Relationen sind, dann ist die Konkatenation von R_1 und R_2 die Menge

(20) $R_1 \cdot R_2 = \{uv|u \in R_1 \text{ und } v \in R_2\}$

und die Vereinigung von R_1 und R_2 die Menge

(21) $R_1 \cup R_2 = \{u|u \in R_1 \text{ oder } u \in R_2\}$.

Wenn $R \subseteq \mathcal{R}_\Sigma \times \mathcal{R}_\Sigma$ eine reguläre Relation ist, und $R^0 = \{<\epsilon, \epsilon>\}$ und $R^i = RR^{i-1}$ für $i \geq 1$ definiert ist, dann ist die Sternhülle R^* die Menge

(22) $R^* = \bigcup_{i \geq 0} R^i$

und die Plushülle R^+ die Menge

(23) $R^+ = \bigcup_{i \geq 1} R^i$

und die Umkehrung die Menge

(24) $Rev(R) = \{<rev(u), rev(v)>|<u, v> \in R\}$.[7]

Reguläre Relationen sind nicht unter der Schnittbildung und der Komplement-

[7] vgl. [27], S.338, S.340: dort werden n-stellige Relationen behandelt.

bildung abgeschlossen[8], aber bezüglich der Komposition. Wenn $R_1 \subseteq \mathcal{R}_\Sigma \times \mathcal{R}_\Sigma$ und $R_2 \subseteq \mathcal{R}_\Sigma \times \mathcal{R}_\Sigma$ zwei reguläre Relationen sind, ist die Komposition von R_1 und R_2 die Menge

(25) $$R_1 \circ R_2 = \{\langle u, v\rangle | \langle u, k\rangle \in R_1 \text{ und } \langle k, v\rangle \in R_2\}$$

([27], S.342).

3.5 Formal Power Series

Im Folgenden sollen die *Formal Power Series* eingeführt werden. Grundlage hierfür ist der Begriff des Semimoduls.

Wenn $(S, \oplus, \otimes, \bar{0}, \bar{1})$ ein Semiring ist, dann wird ein kommutativer Monoid $(A, \oplus, \bar{\bar{0}})$ S-Semimodul oder Semimodul über S genannt oder genauer $({}_SA, \oplus, \bar{\bar{0}})$, wenn bestimmte Eigenschaften erfüllt sind. Es muss eine *Operatoranwendung* (eine Operation) von S auf A, d. h. eine Abbildung von $S \times A$ in A existieren. Ein Element aus A, das dem Paar $(\sigma, a) \in S \times A$ zugeordnet ist, wird mit σa bezeichnet. Hierbei müssen die folgenden Operatorgesetze für alle σ,$\tau \in S$ und alle $a, b \in A$ gelten ([24], S.273f.; [37], S.6):

(26)
i. $\sigma(a \oplus b) = \sigma a \oplus \sigma b$
ii. $(\sigma \oplus \tau)a = \sigma a \oplus \tau a$
iii. $(\sigma \otimes \tau)a = \sigma(\tau a)$
iv. $\bar{1}a = a$
v. $\bar{0}\sigma = \bar{0}$

[8] Eine Teilmenge der regulären Relationen ist bezüglich der Schnittoperation abgeschlossen. Diese Teilmenge wird aus den regulären Relationen gebildet die nur Wortpaare (x,y) enthalten, für die gilt $|x| = |y|$ ([27], S.342ff.).

i. $a\bar{0} = \bar{\bar{0}}$

Mit dem Begriff des Semimoduls lässt sich nun der Begriff der Formal Power Series erläutern. Es soll der Semiring $(S, \oplus, \otimes, \bar{0}, \bar{1})$, der freie Monoid Σ^* und die Menge $S\langle\langle\Sigma^*\rangle\rangle$ aller Abbildungen $r : \Sigma^* \rightarrow S$ betrachtet werden. Die Bilder von r werden durch (r, w), mit $w \in \Sigma^*$, denotiert und werden als *Koeffizienten* von w in r bezeichnet. Der Monoid $({}_S S\langle\langle\Sigma^*\rangle\rangle, \oplus, \bar{0})$ ist ein S-Semimodul, wenn für alle $a, b \in S\langle\langle\Sigma^*\rangle\rangle$ und $\sigma \in S$ die Summe und die Operatorenanwendung folgend definiert sind ([24], S.278):

(27) $(a \oplus b, w) = (a, w) \oplus (b, w)$ und $(\sigma a, w) = \sigma(a, w)$ für alle $w \in \Sigma^*$

Die Elemente r des S-Semimoduls $({}_S S\langle\langle\Sigma^*\rangle\rangle, \oplus, \bar{0})$ bilden eine Formal Power Series. r wird als formale Summe geschrieben ([24], S.278; [37], S.7):

(28) $r = \sum_{w \in \Sigma^*} (r, w)w$

Die Menge $S\langle\langle\Sigma^*\rangle\rangle$ ist also die Menge aller Formal Power Series. Wenn $r \in S\langle\langle\Sigma^*\rangle\rangle$ gegeben ist, wird eine Teilmenge von Σ^* durch $supp(r)$ denotiert und ist folgend definiert ([24], S.279; [37], S.7):

(29) $supp(r) = \{w \in \Sigma^* \mid (r, w) \neq \bar{0}\}$

Diese Teilmenge von Σ^* wird Support auf r genannt.

Die Teilmenge von $S\langle\langle\Sigma^*\rangle\rangle$, die aus allen Formal Power Series mit einem endlichen Support besteht, wird durch $S\langle\Sigma^*\rangle$ denotiert. Formal Power Series aus $S\langle\Sigma^*\rangle$ werden als Polynome bezeichnet ([37], S.7).

Wenn $r_1, r_2 \in S\langle\langle\Sigma^*\rangle\rangle$ zwei Formal Power Series sind, gilt für die abstrak-

te Addition des S-Semimoduls $({}_S S\langle\langle\Sigma^*\rangle\rangle, \oplus, \bar{0})$ folgendes, wobei $r_1 \oplus r_2 \in \mathrm{S}\langle\langle\Sigma^*\rangle\rangle$ ([37], S.7):

(30) $\quad (r_1 \oplus r_2, w) = (r_1, w) \oplus (r_2, w)$ für alle $w \in \Sigma^*$

Die abstrakte Addition ist dann folgend definiert:

(31) $\quad r_1 \oplus r_2 = \sum\limits_{w\in\Sigma^*} ((r_1, w) \oplus (r_2, w))w$

Es soll nun eine Semiringstruktur für die Formal Power Series eingeführt werden. Es soll also der Semiring $({}_S S\langle\langle\Sigma^*\rangle\rangle, \oplus, \otimes, \bar{0}, \epsilon)$ betrachtet werden. Für die abstrakte Multiplikation gilt folgendes, wenn r_1, $r_2 \in \mathrm{S}\langle\langle\Sigma^*\rangle\rangle$ zwei Formal Power Series sind und $r_1 \otimes r_2 \in \mathrm{S}\langle\langle\Sigma^*\rangle\rangle$ ([37], S.8):

(32) $\quad (r_1 \otimes r_2, w) = \sum\limits_{w_1 w_2 = w} (r_1, w_1) \otimes (r_2, w_2)$ für alle $w \in \Sigma^*$

Die abstrakte Multiplikation ist dann folgend definiert:

(33) $\quad r_1 \otimes r_2 = \sum\limits_{w\in\Sigma^*} \left(\sum\limits_{w_1 w_2 = w} (r_1, w_1) \otimes (r_2, w_2) \right) w$

Der Semiring $(S\langle\langle\Sigma^*\rangle\rangle, \oplus, \otimes, \bar{0}, \epsilon)$ ist hierbei kommutativ, wenn der Semiring $(S, \oplus, \otimes, \bar{0}, \bar{1})$ kommutativ ist.

Eine Formal Power Series $r \in S\langle\langle\Sigma^*\rangle\rangle$ ist *quasiregulär*, wenn $(r, \epsilon) = \bar{0}$ ([37], S.21). Wenn $r \in S\langle\langle\Sigma^*\rangle\rangle$ quasiregulär ist, $r^0 = \epsilon$ und $r^i = r \otimes r^{i-1}$ für $i \geq 1$, dann ist die Plushülle r^+ die Menge

(34) $\quad r^+ = \sum\limits_{w\in\Sigma^*} \left(\sum\limits_{i=1}^{|w|} r^i, w \right) w$

und die Sternhülle r^* die Menge

(35) $r^* = \sum_{w \in \Sigma^*} \left(\sum_{i=0}^{|w|} r^i, w \right) w$ ([37], S.22).

Das Hadamard-Produkt zweier Formal Power Series $r_1 \in S\langle\langle \Sigma_1^* \rangle\rangle$ und $r_2 \in S\langle\langle \Sigma_2^* \rangle\rangle$ ist folgend definiert ([37], S.22):

(36) $r_1 \odot r_2 = \sum_{w \in (\Sigma_1 \cap \Sigma_2)^*} (r_1, w) \otimes (r_2, w) w$

Eine Formal Power Series $r \in \mathrm{S}\langle\langle \Sigma^* \rangle\rangle$, wird rational oder genauer S-rational über Σ genannt, wenn r aus Elementen der Menge $S\langle \Sigma^* \rangle$ gebildet werden kann, indem begrenzt viele Operationen von Addition, Multiplikation, Plus und Stern auf Polynome angewendet werden. Die Familie von S-rationalen Formal Power Series über Σ ist durch $S^{rat}\langle\langle \Sigma^* \rangle\rangle$ denotiert.[9]

3.6 Gewichtete reguläre Sprachen

Im Folgenden wird die Theorie der Formal Power Series mit der Theorie der formalen Sprachen zusammengebracht.

Wenn σ ein surjektiver Homomorphismus von $(S\langle\langle \Sigma^* \rangle\rangle, \oplus, \otimes, \bar{0}, \epsilon)$ auf $(\mathcal{L}_\Sigma, \cup, \cdot, \emptyset, \{\epsilon\})$ ist, dann gilt $\sigma(r^*) = (\sigma(r))^*$ für alle $r \in S\langle\langle \Sigma^* \rangle\rangle$, zu denen r^* existiert.[10] Es wird der Subsemiring $(S\langle \Sigma^* \rangle, +, \cdot, \bar{0}, \epsilon)$ der Polynome aus $\langle\langle \Sigma^* \rangle\rangle$ auf den Subsemiring $(\mathcal{E}_\Sigma, \cup, \cdot, \emptyset, \{\epsilon\})$ der endlichen Sprachen und der Subsemiring $(S^{rat}\langle\langle \Sigma^* \rangle\rangle, \oplus, \otimes, \bar{0}, \epsilon)$ der rationalen Power Series auf den Subsemiring $(\mathcal{R}_\Sigma, \cup, \cdot, \emptyset, \{\epsilon\})$ der regulären Sprachen abgebildet. Daher korrespondiert r^+ zu L^+ und r^* zu L^* mit $r \in S^{rat}\langle\langle \Sigma^* \rangle\rangle$ und $L \in \mathcal{R}_\Sigma$ ([24], S.310f.). Die

[9] $S^{rat}\langle\langle \Sigma^* \rangle\rangle$ bildet den kleinsten rational abgeschlossenen Untersemiring von $\mathrm{S}\langle\langle \Sigma^* \rangle\rangle$, der $\mathrm{S}\langle \Sigma^* \rangle$ enthält ([37], S.116).

[10] Für den boolschen Halbkörper $(B, +, \cdot, 0, \epsilon)$ ist die Abbildung σ sogar isomorph ([37], S.8).

Schnittoperation der regulären Sprachen korrespondiert hierbei zum Hadamard-Produkt ([37], S.20).

Die Betrachtung des Supports der rationalen Power Series bildet hierbei eine Verbindung zwischen den rationalen Power Series und den regulären Sprachen. Rationale Power Series sollen im Folgenden als gewichtete reguläre Sprachen betrachtet werden. Der Support bezeichnet hierbei eine reguläre Sprache; die Koeffizienten sind die Gewichte dieser regulären Sprache. Den Wörtern einer regulären Sprache sind hierbei Koeffizienten zugeordnet. Im Zusammenhang mit gewichteten regulären Sprachen soll daher auch von gewichteten Wörtern einer gewichteten regulären Sprache gesprochen werden.

3.7 Gewichtete reguläre Relationen

Analog zu den regulären Relationen können auch gewichtete reguläre Relationen gebildet werden. Es soll eine Abbildung h von Σ_1^* nach Σ_2^* betrachten werden, dann ist für eine Formal Power Series $r \in S^{rat}\langle\langle\Sigma_1^*\rangle\rangle$ folgende Abbildung definiert ([37], S.89):

$$h(r) = \sum_{w \in \Sigma_1^*} (r, w) h(w) \tag{37}$$

In (37) ist h eine Abbildung von $A^{rat}\langle\langle\Sigma_1^*\rangle\rangle$ nach $A^{rat}\langle\langle\Sigma_2^*\rangle\rangle$ ([37], S.89, S.141). Eine gewichtete reguläre Relation ist also eine Abbildung einer gewichteten regulären Sprache in eine gewichtete reguläre Sprache.

Gewichtete reguläre Relationen sind wie reguläre Relationen abgeschlossen unter Sternhülle, Plushülle, Konkatenation und Vereinigung.

Eine wichtige Operation von gewichteten regulären Relationen ist die Komposition, die sich aus dem Hadamard-Produkt ableiten lässt. Wenn $R_1 \subseteq A \times B$

und $R_2 \subseteq C \times D$ zwei gewichtete reguläre Relationen sind, dann ist die Komposition folgend definiert (vgl. [37], S.165):

(38) $$R_1 \otimes R_2 = \{(a,d) \in A \times D \| \exists x \in B \odot C, (a,x) \in R_1 \text{ und } (x,d) \in R_2\}$$

4 Finite-State Maschinen

Finite-State Maschinen sind Maschinen auf der Ebene der Zeichenkettenverarbeitung, die in der Sprachverarbeitung für viele Aufgaben, wie Tokenisierung, morphologische Analyse und Parsing eingesetzt werden. Aus linguistischer Sicht sind sie praktisch, da sie es auf einfache Weise erlauben, die meisten lokalen Phänomene zu beschreiben, auf die man bei empirischen Studien von Sprache stößt. Zudem wird eine Integration einer Vielzahl von Prozessebenen, von der Phonologie bis hin zur syntaktischen Analyse, erleichtert. Ihre praktische Anwendung ist hauptsächlich durch Zeit- und Platzeffizienz motiviert. Zeiteffizienz wird durch deterministische Maschinen erreicht, Platzeffizienz durch Minimierungsalgorithmen auf deterministischen Maschinen. Finite-State Maschinen sind eine Generalisierung der endlichen Automaten, Transduktoren, gewichteten endlichen Automaten und gewichteten Transduktoren.

Es soll erst auf endliche Automaten eingegangen werden, dann auf Transduktoren und letztlich auf die gewichteten Varianten, die eine echte Erweiterung der endlichen Automaten und Transduktoren darstellen. Im Zusammenhang mit regulären Ausdrücken wird es dann möglich sein, diese Maschinen zu modellieren.

4.1 Endliche Automaten

Ein endlicher Automat beschreibt eine Sprache und kann entscheiden, ob ein Eingabewort zu seiner Sprache gehört. Er hat eine endliche Anzahl von Zuständen und befindet sich immer in einem dieser Zustände. Er besitzt einen Startzustand und eine Menge von Endzuständen. Er beginnt im Startzustand und liest Eingabewörter über einem festen Alphabet Σ Zeichen für Zeichen in einzelnen Schritten. Ein Folgezustand hängt vom momentanen Zustand und dem aktuell gelesenen Zeichen des Eingabewortes ab, welches einen Zustandsübergang verursacht. Ein Eingabewort wird akzeptiert, wenn es eine Sequenz von Zustandsübergängen gibt, die zu einem Endzustand führt. Ein endlicher Automat ist folgendermaßen formal definiert (vgl. [45], S.4):

(1) Ein endlicher Automat wird durch ein 5-Tupel der Form $(\Sigma, Q, q_0, F, \delta)$ beschrieben, das (i)-(v) erfüllt

- i. Σ ist eine endliche Menge, das Eingabealphabet.
- ii. Q ist eine endliche Menge, die Menge der Zustände.
- iii. $q_0 \in Q$ ist der Startzustand.
- iv. F$\subseteq$ Q ist die Menge der Endzustände.
- v. $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$ ist eine endliche Menge, die Übergangsrelation.

Wenn sich ein endlicher Automat in einem beliebigen Zustand befindet, und ein Symbol zu null oder zu genau einem Zustandsübergang führt, ist dieser ein deterministischer endlicher Automat (vgl. [45], S.7):

(2) Ein endlicher Automat $(\Sigma, Q, q_0, F, \delta)$ ist ein deterministischer endli-

cher Automat, falls (i) gilt.

i. Die Übergangsrelation δ ist eine Funktion die $Q \times \Sigma$ auf Q abbildet.

Ein deterministischer endlicher Automat kann in linearer Zeit proportional zur Länge des Eingabewortes entscheiden, ob ein Eingabewort zu der Sprache, die er beschreibt, gehört. Jeder nichtdeterministische endliche Automat lässt sich in einen deterministischen endlichen Automaten überführen ([25], S.22-24). Die Zeit- und Platzkomplexität des Determinisierens ist exponentiell ([39], S.20).

Zu jedem deterministischen endlichen Automaten lässt sich über Äquivalenzrelationen ein minimaler endlicher Automat konstruieren, der die gleiche Sprache beschreibt und eine minimale Anzahl an Zuständen hat ([25], S.72-76). Die Zeitkomplexität der Minimierung ist logarithmisch; falls der endliche Automat eine endliche Sprache beschreibt, ist sie linear ([39], S.29).

4.2 Transduktoren

Ein Transduktor ist ein endlicher Automat mit Ausgabe. Dabei ist mit jedem Zustandsübergang eine Ausgabe assoziiert. Ein Transduktor ist folgendermaßen formal definiert:[11]

(3) Ein Transduktor wird durch ein 6-Tupel der Form $(\Sigma, \triangle, Q, q_0, F, \delta)$ beschrieben, das (i)-(v) erfüllt.

i. Σ und $\triangle$ sind zwei endliche Mengen, das Eingabe- und Ausgabealphabet.

ii. Q ist eine endliche Menge, die Menge der Zustände.

iii. $q_0 \in Q$ ist der Startzustand.

[11] vgl. [45], S.14, S.17f.: hier soll ein Zeichen-Transduktor angenommen werden.

iv. F$\subseteq$ Q ist die Menge der Endzustände.

v. $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\triangle \cup \{\epsilon\}) \times Q$ ist die Übergangsrelation.

Wenn es für ein beliebiges Eingabezeichen in jedem Zustand eines Transduktors null oder genau einen Zustandsübergang gibt, hat dieser Transduktor eine deterministische Eingabe und wird sequenzieller Transduktor genannt. Ein sequenzieller Transduktor kann in linearer Zeit proportional zur Länge des Eingabewortes entscheiden, ob ein Eingabewort zu der Sprache gehört, die seine Eingabe beschreibt ([39], S.3(Kapitel 2.1)).

Nicht jeder Transduktor lässt sich in einen sequenziellen Transduktor überführen. Ob ein Transduktor in einen sequenziellen Transduktor überführt werden kann, ist aber entscheidbar. Bei der Determinisierung gilt die gleiche Komplexität wie bei endlichen Automaten ([39], Kapitel 3.3-3.5).

Jeder sequenzielle Transduktor lässt sich in einen minimalen sequenziellen Transduktor überführen, der die gleiche Funktion darstellt und eine minimale Anzahl an Zuständen hat. Bei der Minimierung gilt die gleiche Komplexität wie bei endlichen Automaten ([39], Kapitel 3.7).

Ein Transduktor, der nicht in einen sequenziellen Transduktor überführt werden kann, kann in einen endlichen Automaten überführt werden, indem jedes Paar in $(\Sigma \cup \{\epsilon\}) \times (\triangle \cup \{\epsilon\})$ als ein Zeichen im Eingabealphabet eines endlichen Automaten angesehen wird. Der so gebildete endliche Automat kann dann determinisiert und minimiert und wieder in einen Transduktor überführt werden, indem die Relationen wiederhergestellt werden. Der resultierende Transduktor ist dann optimiert, jedoch nicht unbedingt deterministisch und minimal.

4.3 Gewichtete endliche Automaten

Ein gewichteter endlicher Automat arbeitet ähnlich wie ein endlicher Automat. Beim Lesen eines Zeichens wird jedoch eine rationale Power Series generiert, die vom Zustand und dem gelesenen Zeichen abhängt. Eine Sequenz von Übergängen verursacht eine Multiplikation der rationalen Power Series, die von den einzelnen Übergängen erzeugt werden. Mit dem Startzustand und den Endzuständen sind rationale Power Series assoziiert.[12] Wenn ein Wort durch einen gewichteten endlichen Automaten gelesen wurde und eine rationale Power Series generiert wurde, wird diese rationale Power Series mit Hilfe der Multiplikation mit dem Startgewicht und Endgewicht verbunden. Ein gewichteter endlicher Automat ist folgend formal definiert ([41], Kapitel 2):

(4) $(\Sigma, Q, q_0, F, \delta, \lambda, \rho)$ ist ein gewichteter endlicher Automat (über S), wobei $S = (S, \oplus, \otimes, \bar{0}, \bar{1})$ ein Semiring ist, falls (i)-(vii) erfüllt sind.

i. Σ ist eine endliche Menge, das Eingabealphabet.
ii. Q ist eine endliche Menge, die Menge der Zustände.
iii. $q_0 \in Q$ ist der Startzustand.
iv. $F \subseteq Q$ ist die Menge der Endzustände.
v. $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times S \times Q$ ist eine endliche Menge von Übergängen.
vi. λ ist eine Funktion von q_0 nach S, die Initialgewichtsfunktion.
vii. ρ ist eine Funktion von F nach S, die Endgewichtsfunktion.

Es ist nicht immer möglich, gewichtete endliche Automaten zu determinisieren. Ob eine Determinisierung möglich ist, ist aber wie bei Transduktoren entscheidbar. Hierbei ändert sich die ursprüngliche Komplexität der Determinisierung

[12] Es wird hier angenommen, dass es nur einen Startzustand gibt.

nicht ([39], Kapitel 3.3-3.5). Wenn ein gewichteter endlicher Automat nicht determinisiert werden kann, kann er in einen endlichen Automaten überführt werden.[13]

Jeder deterministische gewichtete endliche Automat kann minimiert werden. Hierbei gilt die Komplexität der Minimierung von endlichen Automaten.

4.4 Gewichtete Transduktoren

Ein gewichteter Transduktor ist ein gewichteter endlicher Automat mit Ausgabe, dabei ist mit jedem Zustandsübergang eine Ausgabe assoziiert. Ein gewichteter Transduktor ist wie folgt formal definiert ([41], Kapitel 2):

(5) $(\Sigma, \triangle, Q, q_0, F, \sigma, \lambda, \rho)$ ist ein gewichteter Transduktor (über S), wobei $S = (S, \oplus, \otimes, \bar{0}, \bar{1})$ ein Semiring ist, falls (i)-(vii) erfüllt sind.

i. Σ und $\triangle$ sind zwei endliche Mengen, das Eingabealphabet und das Ausgabealphabet.

ii. Q ist eine endliche Menge, die Menge der Zustände.

iii. $q_0 \in Q$ ist der Startzustand.

iv. $F \subseteq Q$ ist die Menge der Endzustände.

v. $\sigma \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\triangle \cup \{\epsilon\}) \times S \times Q$ ist eine endliche Menge von Übergängen.

vi. λ ist eine Funktion von q_0 nach S, die Initialgewichtsfunktion.

vii. ρ ist eine Funktion von F nach S, die Endgewichtsfunktion.

[13] Hierbei wird jedes Paar in $\Sigma \cup \{\epsilon\} \times S$ als ein Zeichen im Alphabet eines endlichen Automaten angesehen.

Auch gewichtete Transduktoren können auf der Eingabeseite determinisiert werden. Hierbei gilt die Komplexität der Determinisierung von endlichen Automaten ([39], Kapitel 3.3-3.5). Falls die Determinisierung nicht möglich ist, was auch hier entscheidbar ist, kann ein Transduktor in einen endlichen Automaten überführt werden.[14]

Es ist auch möglich, sequenzielle gewichtete Transduktoren zu minimieren. Hierbei bleibt die ursprüngliche Komplexität der Minimierung unverändert ([39], Kapitel 3.7).

5 Reguläre Ausdrücke

Die Sprache der regulären Ausdrücke ist eine formale Sprache. Sie hat eine einfache Syntax, die jedoch sehr komplex werden kann. Die Sprache der regulären Ausdrücke denotiert Mengen und kann gewichtete und ungewichtete reguläre Sprachen und Relationen beschreiben.

Für eine reguläre Sprache oder gewichtete reguläre Sprache, die durch einen regulärereren Ausdruck r bezeichnet wird, wird L(r) geschrieben, für eine reguläre Relation oder gewichtete reguläre Relation hingegen R(r). Im Zusammenhang mit regulären Ausdrücken wird es dann möglich sein, Finite-State Maschinen zu modellieren.

Es soll als erstes auf die Symbole von regulären Ausdrücken eingegangen werden, dann auf die Operatoren, mit denen diese Symbole verknüpft werden können.

[14] Hierbei wird jedes Element in $(\Sigma \cup \{\epsilon\}) \times (\triangle \cup \{\epsilon\}) \times S$ als ein Zeichen im Alphabet eines endlichen Automaten angesehen.

5.1 Symbole

Reguläre Ausdrücke denotieren im Zusammenhang mit regulären Sprachen eine Menge von Wörtern und im Zusammenhang mit regulären Relationen eine Menge von Paaren von Wörtern. Ein Symbol (a, b, etc.) denotiert hierbei eine Menge aus einem Wort über Σ ($\{a\}$, $\{b\}$, etc.) und bildet einen regulären Ausdruck über reguläre Sprachen. Symbole, die mit einem Doppelpunkt verbunden werden, kennzeichnen ein Symbolpaar. Ein Symbolpaar (a:b, a:0, 0:b, etc.) denotiert eine Menge aus $\Sigma \cup \{\epsilon\} \times \Sigma \cup \{\epsilon\}$ ($\{(a, b)\}$, $\{(a, \epsilon)\}$, $\{(\epsilon, b)\}$, etc.) und bildet einen regulären Ausdruck über reguläre Relationen. Einstellige Symbole können eine Identitätsrelation darstellen. Daher ist es möglich, für das Symbolpaar a:a vereinfacht a zu schreiben. Das Symbol 0 denotiert die Menge $\{\epsilon\}$ bzw. $\{(\epsilon, \epsilon)\}$ und bildet einen regulären Ausdruck. Das Symbol [ ] ist ein regulärer Ausdruck und bezeichnet die leere Menge. Im Folgenden soll der Ausdruck ?* die Universalsprache Σ^* denotieren ([32], S.2f.).

Die gewichteten Varianten können ebenso durch reguläre Ausdrücke beschrieben werden. Im Falle von gewichteten regulären Sprachen beschreibt ein regulärer Ausdruck eine Teilmenge aus $S^{rat}\langle\langle\Sigma^*\rangle\rangle$, bzw. eine Menge von gewichteten Wörtern. Es genügt, die Polynome $a\epsilon$ mit $a \in S$ und $\bar{1}x = x$ mit $x \in \Sigma$ zu betrachten. Hierbei werden die Gewichte mit spitzen Klammern angegeben. <a> und x sind reguläre Ausdrücke und denotieren die rationalen Power Series $a\epsilon$ und x ([37], S.116f.). Bei gewichteten regulären Relationen genügt es, die Polynompaare $(a\epsilon, a\epsilon)$ mit $a \in S$ und $(\bar{1}x, \bar{1}y) = (x, y)$ mit x,$y \in \Sigma \cup \{\epsilon\}$ zu betrachten. <a> und x:y sind reguläre Ausdrücke und denotieren die Polynompaare $(a\epsilon, a\epsilon)$ und (x, y) (vgl. [37], S.116f.). Das Symbol 0 denotiert hierbei die gewichtete Sprache $\{\bar{1}\epsilon\}$ bzw. die gewichtete Relation $\{(\bar{1}\epsilon, \bar{1}\epsilon)\}$ und bil-

det einen regulären Ausdruck. Der Ausdruck ?* denotiert in diesem Fall die Universalsprache mit dem Koeffizienten $\bar{1}$.

5.2 Operatoren

Komplexe reguläre Ausdrücke können mit Hilfe von Operatoren aus einfacheren Ausdrücken aufgebaut werden. Diese Operatoren denotieren Operationen auf gewichtete und ungewichtete reguläre Sprachen oder Relationen. Die gebräuchliche Definition von regulären Ausdrücken beinhaltet die Operationen Konkatenation, Vereinigung und Sternhülle ([25], Kapitel 2.5; [32], S.3).

Die Ausdrucksstärke, die von der grundsätzlichen Definition von regulären Ausdrücken benutzt wird, ist eingeschränkt. Die Einführung von Operatoren auf einer höheren Ebene der Abstraktion ist im Zusammenhang mit gewichteten und ungewichteten regulären Sprachen und Relationen unumgänglich. Es sollen Operatoren einführt werden, die für die folgenden Kapitel relevant sind; dabei sollen die Notationen von Karttunen et al. verwendet werden ([32], S.4).

Es sollen erst die Operatoren für reguläre Sprachen und Relationen angegeben werden. Diese lassen sich dann auf die gewichteten Varianten übertragen. Da reguläre Sprachen und reguläre Relationen unter Konkatenation, Vereinigung, Plushülle, Sternhülle und Umkehrung abgeschlossen sind, können diese Operationen mit Operatoren von regulären Ausdrücken über reguläre Sprachen und Relationen verbunden werden:

(1) *Vereinigung:* A|B

(2) *Konkatenation:* AB

(3) *Plushülle:* A^+

(4) *Sternhülle:* A^*

(5) *Umkehrung:* Rev(A)

Reguläre Sprachen sind unter der Komplementbildung und unter der Schnittbildung und daher auch unter der Subtraktion abgeschlossen. Reguläre Relationen haben nicht diese Eigenschaft (vgl. [27], S.342). Daher können diese Operationen nur mit Operatoren regulärer Ausdrücke über reguläre Sprachen verbunden werden:

(6) *Komplementbildung:* $\sim$A

(7) *Schnittbildung:* A&B

(8) *Subtraktion:* A$-$B

Reguläre Relationen können durch das Kartesische Produkt gebildet werden. Dabei ist zu beachten, dass das Kartesische Produkt auf zwei reguläre Sprachen angewendet wird. Es existieren also Operatoren, die reguläre Ausdrücke über reguläre Sprachen mit regulären Ausdrücken über reguläre Relationen verbinden. Das Kartesische Produkt zweier regulärer Relationen ist durch folgenden Operator denotiert:

(9) *Kartesisches Produkt:* A.x.B

Die Operatoren *Range* und *Dom* denotieren die gleichnamigen Operationen auf reguläre Relationen:

(10) *Definitionsbereich:* Dom(A)

(11) *Wertebereich:* Range(A)

Die Operation *Id* denotiert die gleichnamige Operation auf reguläre Sprachen:

(12) *Identitätsrelation:* Id(A)

Die Komposition wird durch einen Operator denotiert, der ersichtlich ausschließlich auf reguläre Ausdrücke über reguläre Relationen angewendet werden kann:

(13) *Komposition:* A.o.B

Da gewichtete reguläre Sprachen und Relationen unter Sternhülle, Plushülle, Konkatenation, Hadamard-Produkt (Schnitt) und Vereinigung abgeschlossen sind, können diese Operationen mit den entsprechenden Operatoren für reguläre Sprachen und Relationen verbunden werden. Gleiches gilt für gewichtete reguläre Relationen und die Operationen Kartesisches Produkt und Komposition und der Ermittlung des Werte- und Definitionsbereichs.

Eckige Klammern werden benutzt, um Ausdrücke zu gruppieren. Wenn eine reguläre Relation R(B) auf eine reguläre Sprache L(A) angewendet wird, bedeutet dies im Folgenden Range(Id(A).o.B).

Die Syntax von regulären Ausdrücken kann erweitert werden, indem neue Operatoren eingeführt werden. Ein Beispiel dafür sind die Operatoren $Containment$, $Ignore$ und $Option$, die sich auf gewichtete bzw. ungewichtete reguläre Sprachen beziehen ([32], S.4f.; [27], S.345):

(14) *Containment:*
$\$A =_{def}$?*A?*

(15) *Ignore:*

$A/B =_{def}$ Range(Id(A).o.[Id(?)|[0.x.B]]*)

(16) *Option:*
(A) $=_{def}$ A|0

In den folgenden Kapiteln werden einige komplexere Operatoren benötigt, die hier aufgeführt werden sollen. Diese Operatoren stellen Ersetzungsregeln dar.

Der *Unconditional-Replacement-Operator* ersetzt einen Ausdruck durch einen anderen Ausdruck ([30], Abschnitt 1):

(17) *Unconditional-Replacement-Operator:*
$A \rightarrow B =_{def} [\sim \$[A-0][A.x.B]]^* \sim \$[A-0]$

Der reguläre Ausdruck in (17) denotiert eine reguläre Relation, die alle Vorkommen von Wörtern der regulären Sprache L(A) in der Universalsprache durch die Wörter der regulären Sprache L(B) ersetzt. A und B müssen hierbei reguläre Sprachen denotieren.

Der Conditional-Replacement-Operator ersetzt einen Ausdruck in einem bestimmten Kontext durch einen anderen Ausdruck ([30], Abschnitt 2):

(18) *Conditional-Replacement-Operator*:
$A \rightarrow B||C\_D$

Der reguläre Ausdruck in (18) denotiert eine reguläre Relation, die alle Vorkommen von Wörtern der regulären Sprache L(A), die in der Universalsprache von Wörtern aus L(C) geführt und von Wörtern aus L(D) gefolgt werden, durch Wörter der regulären Sprache L(B) ersetzt. A und B müssen auch hier reguläre Sprachen denotieren. Es kann auch nur der linke oder der rechte Kon-

text angegeben werden. Die Definition des Operators ist sehr komplex und für die folgenden Kapitel nicht relevant. Es soll daher auf Karttunen in [30] oder Kaplan & Kay in [27] verwiesen werden. Es ist nur wichtig zu erwähnen, dass der Operator Komplementbildungen von C und D verlangt.

5.3 Modellieren von Finite-State Maschinen

Von endlichen Automaten werden genau die Sprachen akzeptiert, die von regulären Ausdrücken, welche reguläre Sprachen denotieren, spezifiziert werden können. Dahingegen sind die von Transduktoren bezeichneten Relationen genau die Relationen, die mittels regulärer Ausdrücke, welche reguläre Relationen denotieren, spezifiziert werden können.

Reguläre Ausdrücke, die gewichtete reguläre Sprachen beschreiben, können durch gewichtete endliche Automaten dargestellt werden (vgl. [37], S.117f.), reguläre Ausdrücke, die gewichtete reguläre Relationen beschreiben, durch gewichtete Transduktoren (vgl. [37], S.140).

Die regulären Ausdrücke für reguläre Sprachen und Relationen können mit den regulären Ausdrücken für gewichtete reguläre Sprachen und Relationen vermischt werden. Hierbei werden reguläre Sprachen, die durch einen regulären Ausdruck beschrieben werden, als Support einer Formal Power Series angesehen, dessen Koeffizienten den Wert $\bar{1}$ haben. Allgemein werden reguläre Ausdrücke benutzt, um Finite-State Maschinen zu modellieren. Dabei ist die Ausdrucksform der Finite-State Maschinen lediglich eine Darstellungsform, mit der ein Computer umgehen kann ([25], Kapitel 2.5; [27], S.339f.).

Ein Entwickler, der mit komplexen regulären Ausdrücken arbeitet, muss jedoch immer berücksichtigen, welche Zeit- und Platzkomplexität einzelne Operationen auf der Seite der Finite-State Maschinen haben. Die Zeit- und Platz-

komplexität der Schnittoperation von gewichteten und ungewichteten endlichen Automaten und somit auch der Kompositon von gewichteten und ungewichteten Transduktoren ist quadratisch. Die Komplementbildung setzt sogar einen deterministischen endlichen Automaten voraus. Hierbei können sich die endlichen Automaten sehr stark aufblähen ([40], Abschnitt 2). Zwischen solchen aufwendigen Operationsschritten sollte dann minimiert bzw. optimiert werden, damit Finite-State Maschinen handhabbar bleiben. Zudem empfiehlt sich eine abschließende Determinisierung und Minimierung bzw. Optimierung, um Finite-State Maschinen effizienter zu machen.

6 Arbeiten zum robusten Parsing

In diesem Kapitel sollen Arbeiten vorgestellt werden, die das syntaktische Tagging mit regulären Sprachen und das Chunking mit regulären Relationen realisieren. Das syntaktische Tagging und das Chunking werden dann mit Hilfe von regulären Relationen zusammengebracht.

6.1 Syntaktisches Tagging

Karlssons Ansatz zur syntaktischen Dependenzanalyse und Disambiguierung in [28] wird von Koskenniemi, Tapanainen und Voutilainen in mehreren Arbeiten auf reguläre Sprachen abgebildet ([35], [36], [49], [50]).

Es soll erläutert werden, wie die Eingabe, wie die Constraints und wie die Constraint-Grammatik in diesen Ansätzen dargestellt werden. Dann sollen Probleme, die bei der Realisierung des syntaktischen Taggings mit regulären Sprachen auftreten, diskutiert werden.

6.1.1 Die Eingabe

Die Eingabe kann als eine reguläre Sprache angesehen werden, die eine Abfolge von Wörtern eines Satzes mit allen möglichen morphologischen und syntaktischen Lesarten beschreibt. Das Alphabet dieser regulären Sprache besteht aus den Buchstaben eines Alphabets einer natürlichen Sprache und zusätzlich aus Zeichen, die morphologische, syntaktische sowie Wort- und Satzgrenzen-Tags darstellen. In den Arbeiten zur Realisierung des syntaktischen Taggings mit endlichen Automaten werden nicht unbedingt die syntaktischen Tags verwendet, die Karlsson in [28] definiert. Es sollen in den folgenden Beispielen die von Karlsson ([28]) eingeführten syntaktischen Tags und die morphologischen Tags aus dem ENGCG Tagset verwendet werden ([21], S.309f.). Im Folgenden ist mit dem Begriff Tag ein Zeichen aus dem Alphabet der Eingabe gemeint.

Die Eingabe wird als eine reguläre Sprache dargestellt, die Abfolgen von natürlichsprachlichen Wörtern mit ihren morphologischen Analysen und potentiellen syntaktischen Funktionen beschreibt. Zwischen den natürlichsprachlichen Wörtern mit ihren Analysen stehen potentielle Nebensatz- oder Wortgrenzen. Alle Wörter der Eingabe besitzen als Präfix und als Suffix das Hauptsatz-Tag @@. Es wird hier im Gegensatz zu Karlsson [29] zwischen mehreren Arten von Nebensatzgrenzen unterschieden. Das Tag @< markiert den Beginn eines eingebetteten Nebensatzes, das Tag @> das Ende eines eingebetteten Nebensatzes und das Tag @/ alle anderen Nebensatzgrenzen. Das Tag @ kennzeichnet Wortgrenzen, wenn sie nicht schon durch Nebensatzgrenzen markiert sind ([35], S.230).

Der folgende reguläre Ausdruck denotiert eine reguläre Sprache für den Ausdruck *the program run* (vgl. [36], S.156):

(1) @@
the DET
[@|@/|@<|@>]
[[program N NOM SG[@SUBJ|@OBJ]] |
[program V PRES -SG3 @+MAINV]]
[@|@/|@<|@>]
[[run V PRES SG3 @+MAINV]|
[run N NOM PL [@SUBJ|@OBJ]]]
@@

Die Eingabe in (1) ist nicht disambiguiert, da sie mehr als ein Wort enthält. Alle möglichen Lesarten sind hierbei kompakt repräsentiert.[15] Für die Erstellung der Eingabe wird eine morphologische Analyse jedes natürlichsprachlichen Wortes vorausgesetzt. Koskenniemi verwendet in [35] für die morphologische Analyse eine *two-level Morphologie*, die mit regulären Relationen realisiert ist. Bezüglich der Realisierung der two-level Morphologie soll auf Koskenniemi in [34] verwiesen werden.

6.1.2 Constraints

Das Ziel von Constraints ist es, nicht wohlgeformte Lesarten aus einer ambigen Eingabe herauszufiltern. Die Constraints lassen sich als reguläre Sprachen darstellen und können mit Hilfe des *Restriction-Operators* ausgedrückt werden. Der Restriction-Operator beschränkt das Auftreten einer regulären Sprache in

[15] Die Eingabe ist ein azyklischer endlicher Automat, bei dem die einzelnen Wörter mit ihren morphologischen Analysen, potentiellen syntaktischen Funktionen und potentiellen Wort- und Satzgrenzen wie Perlen auf einer Perlenschnur aneinandergereiht sind. Dieser Automat akzeptiert alle möglichen Lesarten.

der Universalsprache auf einen bestimmten Kontext ([32], S.4f.):

(2) *Restriction-Operator:*

- $A \Rightarrow B\_C =_{def} \sim [[\sim [?^* B]A\ ?^*]|[?^* A \sim [C\ ?^*]]]$
- $A \Rightarrow B\_ =_{def} \sim [\sim [?^* B]A\ ?^*]$
- $A \Rightarrow \ \_C =_{def} \sim [?^* A \sim [C\ ?^*]]$

Der Restriction Operator in (2) denotiert eine reguläre Sprache, in der jedes Auftreten eines Teilwortes in L(A) von Wörtern in L(B) geführt und von Wörtern in L(C) gefolgt wird, je nachdem, ob B oder C oder beide regulären Ausdrücke angegeben sind. Der Restriction-Operator ist mit Hilfe der Komplementbildung definiert, daher müssen A, B und C reguläre Sprachen denotieren.

Beim Modellieren der Constraints kann auf die natürlichsprachlichen Wörter, auf die morphologischen Merkmale, auf die syntaktischen Funktionen und auf die Wort- und Satzgrenzen Bezug genommen werden. Es ist in diesem Zusammenhang praktisch, *Makros* zu definieren (vgl. [36], S.157):

(3) $. =_{def} \sim \$[@@|@|@/|@_<|@_>]$

(4) $.. =_{def} [.|@|@_<[.|@|@/]^*@_>]^*$

Das Makro „ . “ bezeichnet eine beliebige Sequenz von Zeichen innerhalb eines Wortes, das Makro „ .. “ eine beliebige Sequenz von Zeichen innerhalb eines Satzes. Mit diesen Makros lassen sich aus den Beschreibungen in (8) aus Abschnitt (2.1.5) Constraints erstellen. Diese Constraints sollen mit dem Restriction-Operator realisiert werden. Die Beschreibung in (8-a) wird folgendermaßen formuliert:

(5) N ⇒ DET _

und die Beschreibung in (8-b) folgendermaßen:

(6) [@/|@<|@>]⇒[V[PRES|PAST].]_[. V[PRES|PAST]]

und die in (8-c) folgendermaßen:

(7) [N . @SUBJ] ⇒ _[∼ [.. N ..]V[PRES|PAST]]

Das Uniqueness Principle für die syntaktische Kopf-Funktion @+FMAINV wird folgendermaßen ausgedrückt:

(8) @+FMAINV⇒ [@@∼ [.. @+FMAINV ..]]_
[∼ [.. @+FMAINV ..]@@]

6.1.3 Constraint-Grammatik

Mehrere Constraints können durch die Schnittbildung zu einer Constraint-Grammatik zusammengefasst werden ([35], S.231; [36], S.157). Eine Grammatik G, die aus den Constraints C_1, C_2, ...,C_n besteht, ist folgend definiert:

(9) G =$_{def}$ C_1&C_2& ... &C_n

Die Reihenfolge der Constraints ist beliebig, da die Schnittoperation kommutativ ist. Eine Constraint-Grammatik kann demnach durch eine reguläre Sprache dargestellt werden, die eine Menge von Wörtern enthält, die wohlgeformte Abfolgen repräsentieren.

Die Anwendung einer Grammatik G auf eine Eingabe S ist folgend definiert:

(10) $S' =_{def} S\&G$

Bei der Anwendung der Grammatik G auf die Eingabe S bleiben die wohlgeformten Lesarten übrig, die auch in der Constraint-Grammatik aufgeführt sind. Die Eingabe in (1), wobei hier Nebensatzgrenzen nicht berücksichtigt werden, kann mit einer Constraint-Grammatik disambiguiert werden, die die Constraints aus (5), (6), (7) und (8) enthält:

(11)
```
@@
the DET
[@|@/|@<|@>]
[program N NOM SG[@SUBJ|@OBJ]]
[@|@/|@<|@>]
[run V PRES SG3 @+FMAINV]
@@
```

Es soll nun diskutiert werden, welche Probleme im Zusammenhang mit dem syntaktischen Tagging mit endlichen Automaten auftreten können.

6.1.4 Berechnung

Die endlichen Automaten, die aus dem Restriction-Operator resultieren, können relativ groß werden, da Komplementierungen der Kontextbeschreibungen vorgenommen werden müssen. Eine große Anzahl von Constraints kann dazu führen, dass ein endlicher Automat, der eine Constraint-Grammatik beschreibt, nicht mehr praktikabel ist, da auch die Schnittbildung sehr aufwendig ist. In der Regel sind aber viele Constraints notwendig, um gute Ergebnisse zu erzielen. Es können mehrere Strategien angewendet werden, um dieses Platzproblem

zu behandeln.

Koskenniemi in [35] berechnet den Schnitt der Constraints einer Constraint-Grammatik nicht vor der Anwendung. Erst bei der Anwendung auf eine Eingabe werden alle enthaltenen Constraints parallel mit der Eingabe geschnitten. Die Resultate der Schnittbildung werden dann untereinander geschnitten. Bestimmte Constraints können hierbei auch vor der Anwendung der Constraint-Grammatik durch die Schnittbildung zusammengefasst werden ([35], S.232).

Auch Voutilainen & Tapanainen in [49] berechnen den Schnitt der Constraints einer Constraint-Grammatik nicht vor der Anwendung. Bei der Anwendung auf eine Eingabe wird im ersten Schritt ein Constraint aus der Constraint-Grammatik mit der Eingabe geschnitten. Das Ergebnis wird dann mit einem nächsten Constraint aus der Constraint-Grammatik geschnitten, bis alle Constraints abgearbeitet sind. Die Constraints werden hier in einer Sequenz angewendet, sie können dabei nach ihrer Relevanz geordnet sein. Constraints, die einen potentiell hohen Disambiguierungsgrad haben, werden hierbei zuerst angewendet. Bestimmte Constraints können auch hier vor der Anwendung der Constraint-Grammatik durch die Schnittbildung zusammengefasst werden ([49], S.402).

Auf die Strategien, die im Zusammenhang mit der Anwendung einer Constraint-Grammatik auf eine Eingabe verfolgt werden, soll nicht weiter eingegangen werden; es wird auf Tapanainen verwiesen ([48], Kapitel 10). Die Zeitkomplexität der Anwendung einer Constraint-Grammatik auf eine Eingabe ist quadratisch, da die Zeitkomplexität der Schnittbildung quadratisch ist.

6.1.5 Robustheit

Im Folgenden soll diskutiert werden, ob das syntaktische Tagging mit endlichen Automaten robust ist.

Wenn eine Constraint-Grammatik Ambiguitäten auflöst, indem sie nur wohlgeformte Abfolgen zulässt, wird diese als *effektiv* bezeichnet. Das Ergebnis der Anwendung einer Constraint-Grammatik auf eine Eingabe kann unter bestimmten Umständen leer sein. Dieser Fall kann auftreten, wenn in der Eingabe keine wohlgeformte Lesart enthalten ist und die Constraint-Grammatik effektiv ist. Wenn die Eingabe einen längeren Satz mit mehreren Nebensätzen darstellt und eine nicht wohlgeformte Abfolge in einem der Nebensätze auftritt, kann die gesamte Analyse der Eingabe durch eine Constraint-Grammatik scheitern ([50], S.298).

Eine Constraint-Grammatik ist *true*, wenn sie jede wohlgeformte Abfolge erlaubt. Wenn eine Constraint-Grammatik nicht true ist, kann es zu einer leeren Analyse einer Eingabe, die eine wohlgeformte Lesart enthält, kommen. Dieser Fall kann bei längeren Sätzen auftreten, die nicht mehr von einer Constraint-Grammatik abgedeckt werden. In diesem Fall ist die wohlgeformte Lesart der Eingabe zu komplex, um von der Constraint-Grammatik beschrieben zu werden. Voutilainen in [50] schlägt vor, in einem Präprozess erst einfache morphologische und syntaktische Ambiguitäten durch eine Constraint-Grammatik aufzulösen und dann durch eine andere Constraint-Grammatik weiter zu disambiguieren. Auf diese Weise kann ein längerer Satz wenigstens teilweise analysiert werden ([50], S.298, S.304).

Wegen den hier aufgeführten Problematiken ist das syntaktische Tagging mit regulären Sprachen nicht robust.

6.2 Chunking

Joshi & Hopely in [26], Grefenstette in [20] und Megyesi und Rydin in [38] bilden eine Analyse auf der Basis von Wortgruppierungen auf reguläre Relationen ab. Die Wortgruppierungen bilden hier nicht unbedingt Chunks im Sinne von Abney [3]. Es werden hier die Begriffe *Gruppierungen*, *first-order strings* oder *Phrasen* benutzt.[16]

Es soll nicht weiter darauf eingegangen werden, inwieweit Gruppierungen in den Ansätzen Chunks bilden. Alle Ansätze können jedoch für eine Analyse durch Chunks verwendet werden. Die Verfahren unterscheiden sich nicht in der Darstellungsform der Eingabe, aber in der Verfahrensweise, wie Gruppierungen gebildet werden. Im Folgenden sollen die Verfahren aus der Sichtweise des Chunking vorgestellt werden. In den aufgeführten Beispielen wird das *Brown Corpus Tagset* verwendet ([21], S.305ff.).

6.2.1 Die Eingabe

Die Eingabe kann als eine reguläre Sprache dargestellt werden. Es wird hierbei über die Kategorien der natürlichsprachlichen Wörter eines Satzes abstrahiert. Chunks sind nicht für morphologische Ambiguitäten definiert, daher darf die reguläre Sprache der Eingabe nur ein Wort enthalten. Die natürlichsprachlichen Wörter eines Satzes müssen also eindeutig getaggt sein. Die Eingabe enthält folglich ein Wort, das eine Abfolge von Tags beschreibt (vgl. [26], S.1f.; [20], S.2; [38], Abschnitt 1). Die reguläre Sprache für den Satz *Bill saw the little dog in the park* enthält das Wort:

[16] Die first-order strings bilden hierbei Chunks ([26], S.5).

(12) NP VBD DTI JJ NN IN DTI NN <$.>

6.2.2 Chunking mit regulären Relationen

Chunks können nach ihrem syntaktischen Kopf, dem Chunk-Ceiling, in Chunk-Typen unterteilt werden. Diese Chunk-Typen können wiederum als reguläre Sprachen betrachtet werden, die Tagabfolgen beschreiben. Diese Tagabfolgen stellen Abfolgen von funktionalen Elementen mit ihren Komplementen und den dazwischenliegenden Elementen dar. Leere funktionale Elemente werden hierbei nicht dargestellt. Eine Chunk-Struktur wird durch Konstituenten gebildet. Diese Konstituenten können wiederum aus anderen Konstituenten aufgebaut sein. Die Konstituenten können genau wie Chunks als reguläre Sprachen betrachtet werden. Jeder Konstituente kann anhand ihres syntaktischen Kopfes ein Typ zugeordnet werden. Ein Chunk besteht also aus den Sprachen verschiedener Typen von Konstituenten.

Chunks mit ihrer Konstituentenstruktur werden hier durch Klammerungen angezeigt. Das Klammern wird durch reguläre Relationen realisiert. Diese regulären Relationen bilden reguläre Sprachen, die Chunk- bzw. Konstituenten-Typen beschreiben, auf reguläre Sprachen ab, in denen Chunk- bzw. Konstituenten-Typen geklammert sind. Die so gebildete Struktur gleicht dem labeled bracketing ([17], S.29f.). Jedem Konstituenten- und damit auch jedem Chunk-Typ werden hierbei spezielle Klammersymbole zugeteilt (vgl. [38], Abschnitt 2).

Der Aufbau der Chunks verläuft bottom-up. Dabei kann die Analyse aus mehreren Ebenen bestehen. Jede Ebene stellt eine reguläre Relation dar, die einen bestimmten Chunk-Typ klammert. Es werden erst kleinere Chunk-Typen erkannt. Diese können in einer höheren Ebene als Komplement eines funktiona-

len Elementes mit anderen Tags zu einem größeren Chunk-Typ zusammengefasst werden. Chunk-Typen bilden hierbei Konstituenten-Typen eines anderen Chunk-Typs. Wenn Tags in einer Ebene zu einem Chunk-Typ zusammengefasst wurden, werden sie in einer höheren Ebene durch die Typ-Klammerungen als eine Kategorie behandelt. Tags, die in einer Ebene nicht zu einem Chunk-Typ gruppiert werden können, werden zur nächsten Ebene weitergereicht (vgl. [38], Abschnitt 2.1).

Die Einbettungstiefe der Chunkstruktur ist auf die Anzahl der angewendeten Ebenen begrenzt. Die Hintereinanderschaltung solcher Ebenen wird als *Kaskade* bezeichnet ([6], S.1). Eine Kaskade K, die aus den Ebenen E_1, E_2, ..., E_n besteht, ist folgendermaßen definiert:

(13) $K =_{def} E_1.o.E_2.o. \ldots .o.E_n$

Die in (13) definierte Kaskade kann nun auf eine Eingabe S angewendet werden, um Chunks mit ihrer Konstituentenstruktur in S zu klammern:

(14) $S' =_{def}$ Range(Id(S).o.K)

NP-Chunks sollen durch die Klammern $\{_{np}$ und $\}_{np}$, PP-Chunks durch die Klammern $\{_{pp}$ und $\}_{pp}$ und AP-Chunks durch die Klammern $\{_{ap}$ und $\}_{ap}$ angezeigt werden. Im folgenden Beispiel werden zwei Ebenen angenommen, die NP- und PP-Chunks klammern und zu einer Kaskade komponiert werden. Hierbei können PP-Chunks NP-Chunks als Konstituenten enthalten. Der Satz *Bill saw the little dog in the park*, der wie in (12) getaggt ist, wird nun schrittweise analysiert:

(15) 1. *Eingabe:*

Bill$_{NP}$ saw$_{VBD}$ the$_{DTI}$ little$_{JJ}$ dog$_{NN}$ in$_{IN}$ the$_{DTI}$ park$_{NN}$.$_{<\$.>}$

2. *NP-Chunks:*
   $\{_{np}$ Bill$_{NP}$ $\}_{np}$ saw$_{VBD}$ $\{_{np}$ the$_{DTI}$ little$_{JJ}$ dog$_{NN}$ $\}_{np}$ in$_{IN}$ $\{_{np}$ the$_{DTI}$ park$_{NN}$ $\}_{np}$.$_{<\$.>}$
3. *PP-Chunks:*
   $\{_{np}$ Bill$_{NP}$ $\}_{np}$ saw$_{VBD}$ $\{_{np}$ the$_{DTI}$ little$_{JJ}$ dog$_{NN}$ $\}_{np}$ $\{_{pp}$ in$_{IN}$ $\{_{np}$ the$_{DTI}$ park$_{NN}$ $\}_{np}$ $\}_{pp}$.$_{<\$.>}$

Bei diesem Verfahren ist es manchmal notwendig, mehrere Chunk-Typen in einer Ebene zu behandeln. AP-Chunks können keine Konstituenten innerhalb eines NP-Chunks bilden, da hier auf Grund des Prinzips Chunk Connectedness kein leeres funktionales Element angenommen werden darf. Daher müssen NP-Chunks und AP-Chunks in einer Ebene behandelt werden. Es wird entweder ein AP-Chunk oder ein NP-Chunk gebildet. Eine Ebene E kann also mehrere Ebenen E_1, E_2, ...,E_n enthalten:

(16) $E =_{def} E_1|E_2| \ldots |E_n$

Wenn sich zwei Chunk-Typen nicht enthalten können, können diese wahlweise durch eine Kaskade oder durch die Bildung einer neuen Ebene verbunden werden.

6.2.3 Attachment-Ambiguitäten

Im Folgenden sollen die einzelnen Ebenen einer Kaskade betrachtet werden. Die regulären Sprachen der Chunk-Typen sollen hierbei geklammert werden, um Gruppierungen anzuzeigen. Um Vorkommen einer regulären Sprache in der Universalsprache zu klammern, kann der *Optional-Insertion-Operator* verwen-

det werden:[17]

(17) *Optional-Insertion-Operator:*
$A(\rightarrow)P...S =_{def} [?^*[0.x.P]A[0.x.S]]^*?^*$

In (17) müssen P (Präfix) und S (Suffix) reguläre Sprachen denotieren und bezeichnen die linke und rechte Klammer, mit denen der Ausdruck A geklammert werden soll. A kann eine reguläre Relation denotieren; in unserem Fall denotiert A die reguläre Sprache eines Chunk-Typs. Nach diesem Operator werden folgende Klammerungen vorgenommen, wenn die reguläre Relation R([ab|b|ba|aba] (→) { ... }) auf die Eingabe L(aba) angewendet wird:

(18)

aba	aba	aba	aba	aba
a{b}a	a{ba}	{ab}a	{aba}	aba

Die Klammerungen sind höchst ambig. Zudem wird nicht jedes Vorkommen der zu klammernden Sprache geklammert. Um wirklich jedes Vorkommen zu klammern und beim Chunking das Prinzip der Chunk Connectedness nicht zu verletzen, kann ein *Obligatory-Insertion-Operator* definiert werden:[18]

(19) *Obligatory-Insertion-Operator:*
$A\rightarrow P...S =_{def}$
$[\sim \$[A\text{-}0][0.x.P]A[0.x.S]]^* \sim \$[A\text{-}0]$

In (19) müssen P, S und A reguläre Sprachen denotieren. Nach diesem Operator werden folgende Klammerungen vorgenommen, wenn die reguläre Relation R([ab|b|ba|aba]→{...}) auf die Eingabe L(aba) angewendet wird.

[17] vgl. [30], Abschnitt 1.4: dort ist ein Optional-Replacement-Operator definiert.
[18] vgl. [30], Abschnitt 1: dort ist ein Unconditional-Replacement-Operator definiert.

(20) aba aba aba aba
a{b}a a{ba} {ab}a {aba}

Auch nach der Anwendung des Obligatory-Insertion-Operator bleiben die Klammerungen höchst ambig. Jedoch wird jetzt jedes Vorkommen der zu klammernden Sprache geklammert.

Beim Chunking treten hier ambige Klammerungen auf, wenn der Anfangs- und Endpunkt eines Chunk-Typs nicht eindeutig zu bestimmen ist. Dieser Fall kann eintreten, wenn zwischen einem funktionalen Element und dem thematischen Element, das selektiert wird, ein Inhaltswort steht, welches potentiell selektiert werden kann. Eine mit dem Obligatory-Insertion-Operator gebildete reguläre Relation, die Chunks in der Universalsprache klammert, enthält jedoch keine Information darüber, welches thematische Element selektiert ist, wenn mehrere thematische Elemente zur Auswahl stehen:

(21) a. $\{_{np}$ the$_{DTI}$ country$_{NN}$ departments$_{NNS}$ $\}_{np}$
b. $\{_{np}$ the$_{DTI}$ country$_{NN}$ $\}_{np}$ $\{_{np}$ departments$_{NNS}$ $\}_{np}$

Demnach kann in (21) entweder *country* oder *departments* Komplement des funktionalen Elements *the* sein. Ähnliches gilt für Ebenen, die zu einer neuen Ebene vereinigt wurden:

(22) a. $\{_{np}$ the$_{DTI}$ big$_{JJ}$ fish$_{NN}$ $\}_{np}$
b. *the$_{DTI}$ $\{_{AP}$ big$_{JJ}$ $\}_{AP}$ $\{_{np}$ fish$_{NN}$ $\}_{np}$

In (22-b) wird ein leeres funktionales Element angenommen, das *big* selektiert. Daher kann das funktionale Element *the* nicht zu seinem Komplement gruppiert werden. Leere funktionale Elemente sollen aber nur angenommen wer-

den, wenn es zu keinen verwaisten funktionalen Elementen kommt. In (22-b) dagegen wird kein leeres funktionales Element angenommen, das *big* selektiert. Eine mit dem Obligatory-Insertion-Operator gebildete reguläre Relation, die Chunks in der Universalsprache klammert, enthält jedoch keine Information darüber, ob ein leeres funktionales Element angenommen wird. Die ambigen Klammerungen in (22) stellen somit eine Attachment-Ambiguität dar, in der die Zugehörigkeit einer Tag-Abfolge nicht eindeutig ist.

Durch diese Attachment-Ambiguitäten kann es zu globalen strukturellen Ambiguitäten kommen:

(23) a. $\{_{np}$ John$_{NP}$ $\}_{np}$ sold$_{VBD}$ $\{_{np}$ old$_{JJ}$ folks$_{NNS}$ homes$_{NNS}$ $\}_{np}$
b. $\{_{np}$ John$_{NP}$ $\}_{np}$ sold$_{VBD}$ $\{_{np}$ old$_{JJ}$ folks$_{NNS}$ $\}_{np}$
$\{_{np}$ homes$_{NNS}$ $\}_{np}$

In (23) gibt es zwei Lesarten, entweder ist *John* Verkäufer von *old folks homes* oder John verkauft *homes* an *old folks* ([5], S.6). Zudem tritt ein Problem bei Mehrwortausdrücken auf:

(24) a. $\{_{np}$ Atlanta$_{NP}$ Country$_{NN}$ $\}_{np}$
b. *$\{_{np}$ Atlanta$_{NP}$ $\}_{np}$ $\{_{np}$ Country$_{NN}$ $\}_{np}$

In (24-b) ist das Prinzip der Chunk Connectedness verletzt, da *Atlanta Country* ein Mehrwortausdruck ist und daher als ein einzelnes thematisches Element angesehen werden kann, das von einem leeren funktionalen Element selektiert wird. Das Problem der Mehrwortausdrücke und das Problem von ambigen Ausdehnungen können durch Disambiguierungsstrategien behandelt werden. Ziel ist es, eindeutige wohlgeformte Chunks zu erhalten. Es gilt also, die ambigen

Klammerungen durch Einschränkungen auf eine Klammerung zu reduzieren.

6.2.4 Constraints

Durch die Chunk-Analyse mit dem Obligatory-Insertion-Operator können Attachment-Ambiguitäten innerhalb des Chunkings auftreten. Einige gebildete Klammerungen können hierbei das Prinzip Chunk Connectedness verletzen. Im Folgenden sollen Constraints vorgestellt werden, die nach einem bestimmten Prinzip Klammerungen ausschließen. Die Definitionen der Constraints stützen sich auf Präfix- und Suffixeigenschaften von Wörtern einer regulären Sprache, in unserem Fall die reguläre Sprache eines Chunk-Typs. Letztlich soll es möglich sein, durch einen bestimmten Constraint wohlgeformte Chunk-Typen eindeutig zu klammern.

Es sollen erst einfachere Constraints vorgestellt werden, mit deren Hilfe komplexere Constraints definiert werden können. Durch die Constraints kann es jedoch zu falschen Analysen kommen. Zudem ist die Umsetzung einiger Constraints in einen Transduktor nicht unproblematisch.

Left-to-Right-Constraint

Joshi & Hopely stellen in [26] einen *Left-to-Right-Constraint* (LR-Constraint)[19] vor, der bestimmte Klammerungen bzw. Gruppierungen unterdrückt:

(25) *LR-Constraint:*
Es soll eine reguläre Sprache A hinsichtlich (i)-(ii) geklammert werden.

[19] Joshi & Hopely nennen den LR-Constraint *left to right FST* ([26], S.1).

i. Ein Wort in A, dessen Präfix in der Universalsprache Suffix eines Wortes in A ist, wird nicht geklammert.
ii. Alle anderen Wörter in A werden in der Universalsprache geklammert.

In (25) können ambige Klammerungen nur dann auftreten, wenn ihre Ausdehnungen nach rechts nicht eindeutig sind.

Wenn innerhalb von Chunk-Typen das funktionale Element immer rechts vom Komplement steht, treten hier keine Attachment-Ambiguitäten auf, da die Ausdehnung einer Klammerung nach rechts durch das funktionale Element eindeutig ist. Hierbei wird die größte Ausdehnung geklammert. Letztlich ist mit diesem Constraint jedoch keine eindeutige Klammerung von Chunk-Typen gewährleistet.

Joshi & Hopely haben in [26] keinen regulären Ausdruck für den LR-Constraint angegeben. Die folgenden Definitionen orientieren sich an Karttunen [31], damit später dessen Constraints überleitend eingeführt werden können. Es soll der Left-to-Right-Insertion-Operator (LRI-Operator) eingeführt werden, der den LR-Constraint realisiert. Die Definition des LRI-Operators kann in die Schritte Initial-Match, Left-to-Right und Insertion unterteilt werden:

(26) *LRI-Operator:*
A(@→)P...S $=_{def}$
Initial-Match(A).o.Left-to-Right(A).o.Insertion(P,S)

Der Ausdruck in (26) klammert nach dem LR-Constraint Wörter aus L(A) in der Universalsprache mit den Wörtern aus L(P) (Präfix) und L(S) (Suffix).

Es soll nun auf die Bestandteile des LRI-Operator eingegangen werden. Initial-Match fügt das Zeichen ^ vor jedem Vorkommen der zu klammernden regulären Sprache ein. Die Einfügung des Zeichen ^ kann mit einem *Conditional-Replacement-Operator* realisiert werden ([31], Abschnitt 2):

(27) Initial-Match(A) $=_{def}$ 0→^ || _A

Durch Initial-Match werden auch Vorkommen von Wörtern in L(A) markiert, die in einem anderen Vorkommen von Wörtern in L(A) enthalten sind. Es wird angenommen, dass das Zeichen ^ nicht in der Eingabe vorkommt. Wenn die reguläre Relation R(Initial-Match(ab|b|ba|aba)) auf die Eingabe L(aba) angewendet wird, werden folgende Markierungen vorgenommen:

(28) aba
^a^ba

Left-to-Right klammert Wörter der zu klammernden Sprache nach dem LR-Constraint mit spitzen Klammern und orientiert sich dabei an den eingefügten Zeichen ^ ([31], Abschnitt 2):

(29) Left-to-Right(A) $=_{def}$
$[[\sim \$[\hat{}][\hat{}:<[[A/\hat{}] - [?^*\hat{}]]0:>]]^* \sim \$[\hat{}]].o.[\hat{}\rightarrow 0]$

In (29) bedeutet der Teilausdruck $[[A/\hat{}]-[?^*\hat{}]]$, dass das Zeichen ^ nur innerhalb der zu klammernden regulären Sprache ignoriert wird, daher darf das letzte Zeichen in dem Ausdruck kein Zeichen ^ sein. Es werden nur Wörter aus L(A) geklammert, deren Anfänge in der Universalsprache mit einem Zeichen ^ markiert sind. Markierungen, die sich in der Universalsprache innerhalb eines Wortes aus

L(A) befinden, werden ignoriert und können zu keiner Klammerung führen. Ein Zeichen ˆ, das ein Wort in L(A) in der Universalsprache markiert, wird durch eine öffnende spitze Klammer ersetzt; eine schließende spitze Klammer wird am Ende eines markierten Wortes eingefügt. Die ignorierten Zeichen ˆ werden dann mit Hilfe des Obligatory-Replacement-Operators entfernt. Es wird angenommen, dass die spitzen Klammern nicht in der Eingabe vorkommen. Wenn die reguläre Relation R(Initial-Match(ab|b|ba|aba)) mit der regulären Relation R(Left-to-Right-Constraint(ab|b|ba|aba)) komponiert und auf die Eingabe L(aba) angewendet wird, werden folgende Klammerungen vorgenommen:

(30) aba aba
<ab>a <aba>

Im letzten Schritt werden durch Insertion die eingefügten spitzen Klammern durch die gewünschten Klammerungsausdrücke ersetzt, in unserem Fall durch Konstituenten-Typ- bzw. Chunk-Typ-Klammern. Es wird hierfür ein Obligatory-Replacement-Operator verwendet (vgl. [31], Kapitel 2):

(31) insertion(P,S) $=_{def}$ [<→P].o.[>→S]

Durch Insertion werden öffnende spitze Klammern durch Wörter aus L(P) und schließende spitze Klammern durch Wörter aus L(S) ersetzt. Wenn die reguläre Relation R([ab|b|ba|aba](@→){...}) auf die Eingabe L(aba) angewendet wird, werden folgende Klammerungen vorgenommen:

(32) aba aba
{ab}a {aba}

Mit dem LRI-Operator werden nicht unbedingt sequenzielle Transduktoren konstruiert ([26], S.3). Für ein Wort der zu klammernden regulären Sprache kann es in einem Transduktor mehrere Endzustände geben, die eine Ausgabe produzieren.

Right-to-Left-Constraint

Joshi & Hopely stellen in [26] den Right-to-Left-Constraint (RL-Constraint)[20] vor. Dieser ähnelt dem LR-Constraint, es ist lediglich die Richtung der zu ignorierenden Klammerungen vertauscht:

(33) *RL-Constraint:*
Es soll eine reguläre Sprache A hinsichtlich (i)-(ii) geklammert werden.

i. Ein Wort in A, dessen Suffix in der Universalsprache Präfix eines Wortes in A ist, wird nicht geklammert.
ii. Alle anderen Wörter in A werden in der Universalsprache geklammert.

In (33) können ambige Klammerungen nur auftreten, wenn sie nach links mehrere Ausdehnungen haben. Die Klammerung von Chunk-Typen ist also eindeutig, wenn das funktionale Element immer links vom Komplement steht. Wie bei dem LR-Constraint ist auch hier keine eindeutige Bildung von Chunk-Konstituenten gewährleistet.

Für den RL-Constraint haben Joshi & Hopely in [26] keinen regulären Ausdruck angegeben. Es soll daher der *Right-to-Left-Insertion-Operator*

[20] Joshi & Hopely nennen den RL-Constraint *right to left FST* ([26], S.1).

(RLI-Operator) eingeführt werden, der sich mit Hilfe des LRI-Operators definieren lässt:

(34) *RLI-Operator*:
A(→@)P...S $=_{def}$ Rev(Rev(A)(@→)S...P)

Wenn die reguläre Relation R(⌊ab|b|ba|aba⌋(→@){...}) auf die Eingabe L(aba) angewendet wird, werden folgende Klammerungen vorgenommen:

(35) aba aba
a{ba} {aba}

Der RLI-Operator bildet wie der LRI-Operator nicht unbedingt einen sequenziellen Transduktor.

Bidirectional-Constraint

Joshi & Hopely benutzen in [26] einen *Bidirectional-Constraint*[21], um Chunk-Konstituenten eindeutig zu klammern. Der Bidirectional-Constraint besteht aus zwei wesentlichen Teilen, dem LR- und dem RL-Constraint:[22]

(36) *Bidirectional-Constraint:*
Es soll eine reguläre Sprache A hinsichtlich (i)-(iii) geklammert werden.

i. Ein Wort in A, dessen Suffix in der Universalsprache Präfix eines Wortes in A ist, wird nicht geklammert.

[21] Joshi & Hopely nennen den Bidirectional-Constraint *longest path strategy* ([26], S.2).

[22] Der Bidirectional-Constraint ähnelt einem Verfahren das Kaplan & Kay in [27] im Zusammenhang mit Ersetzungsregeln entwickelt haben. Kaplan & Kay sind dort an der längsten Ersetzung innerhalb eines bestimmten Kontextes interessiert ([27], S.358).

ii. Ein Wort in A, dessen Präfix in der Universalsprache Suffix eines Wortes in A ist, wird nicht geklammert.

iii. Alle anderen Wörter in A werden in der Universalsprache geklammert.

In (36) können keine ambigen Klammerungen auftreten, da der Beginn und das Ende jeder Klammerung eindeutig bestimmt ist. Zudem wird die größte Ausdehnung angenommen, da die am weitesten links stehende und am weitesten rechts stehende Klammer eine Klammerung bilden.

Joshi & Hopely haben in [26] keine formale Definition des Bidirectional-Constraint angegeben. Daher soll der *Bidirectional-Insertion-Operator* definiert werden, der sich aus dem LRI-Operator und dem RLI-Operator ableiten lässt:

(37) *Bidirectional-Insertion-Operator:*
$A@\rightarrow@P...S =_{def}$
$[A(\rightarrow@)P...S].o.[Id(Range(A(@\rightarrow)P...S))]$

Der Ausdruck in (37) klammert nach dem Bidirectional-Constraint Wörter aus L(A) in der Universalsprache mit den Wörtern aus L(P) (Präfix) und L(S) (Suffix). Durch die Komposition des RLI-Operators mit dem Wertebereich des LRI-Operators werden nur Wörter aus L(A) geklammert, die nach dem LR-Constraint und dem RL-Constraint geklammert werden. Wenn die reguläre Relation R([ab|b|ba|aba]@→@{...}) auf die Eingabe L(aba) angewendet wird, wird folgende Klammerung vorgenommen:

(38) aba
{aba}

Es existiert nur eine Klammerungsvariante, die sowohl bei der Anwendung des LR-Constraint und des RL-Constraint auftritt. Jedoch gibt es Fälle, die durch dieses Verfahren nicht abgedeckt werden. Wenn die reguläre Relation R([aba]@→@{...}) auf die Eingabe L(ababa) angewendet wird, scheitert die Analyse. Es soll die Problematik an den Zwischenschritten des Bidirectional-Insertion-Operator gezeigt werden. Die Anwendung der regulären Relation R([aba](@→){...}) auf die Eingabe L(ababa) liefert folgendes Ergebnis:

(39) ababa
{aba}ba

Die Anwendung der regulären Relation R([aba](→@){...}) auf die Eingabe L(ababa) liefert dagegen folgendes Ergebnis:

(40) ababa
ba{aba}

Die Anwendungen des LR- und des RL-Constraints besitzen keine übereinstimmenden Klammerungen, aus diesem Grund wird die Eingabe nicht akzeptiert und die Analyse scheitert.

Der Transduktor, der aus dem Bidirectional-Insertion-Operator resultiert, ist nicht in jedem Fall sequenziell. Dies ist der Fall, wenn die zu klammernde reguläre Sprache eine unendliche Sprache ist. Ein Transduktor, der aus dem regulären Ausdruck [a^+ b]@→@{...} resultiert, kann nicht sequenziell sein. Vor einem Teilwort, das aus den Zeichen *a* besteht, eine Klammer einzufügen und jedes Zeichen *a* zur Ausgabe zu kopieren, hängt letztlich davon ab, ob das Wort mit dem Zeichen *b* terminiert (vgl. [31], Abschnitt 2).

Left-to-Right-Longest-Match-Constraint

Grefenstette in [20] und Megyesi und Rydin in [38] verwenden einen *Left-to-Right-Longest-Match-Constraint* (LRLM-Constraint), um Chunks eindeutig zu klammern.[23]

Dieser Left-to-Right-Longest-Match-Constraint ist von Karttunen in [31] definiert. Der LRLM-Constraint stützt sich auf den LR-Constraint, er determiniert jedoch die unterschiedlichen Ausdehnungen von Klammerungen nach rechts:

(41) *LRLM-Constraint:*
Es soll eine reguläre Sprache A hinsichtlich (i)-(iii) geklammert werden.

i. Ein Wort in A, dessen Präfix in der Universalsprache Suffix eines Wortes in A ist, wird nicht geklammert.
ii. Ein Wort in A, das in der Universalsprache ein echtes Präfix eines Wortes in A ist, wird nicht geklammert.
iii. Alle anderen Wörter in A werden in der Universalsprache geklammert.

In (41) können keine ambigen Klammerungen auftreten, da der Beginn jeder Klammerung eindeutig bestimmt ist und die größte Ausdehnung angenommen

[23] Auch Abney wendet in [6] innerhalb seines Parser CASS (Cascaded Analysys of Syntactic Structure) die Left-to-Right und Longest-Match Strategie an. Abney realisiert das Chunking mit Hilfe von deterministischen endlichen Automaten. Jedoch basiert sein Ansatz auch auf Prozeduren, die nicht der Automatentheorie entspringen. Nach Elworthy ist die Zeitkomplexität seines Parsers quadratisch ([14], S.2).

wird.

Es soll ein Operator für den LRLM-Constraint eingeführt werden. Der *Left-to-Right-Longest-Match-Insertion-Operator* (LRLMI-Operator) lässt sich in die Teiloperationen Initial-Match, Left-to-Right, Longest-Match und Insertion gliedern und ist folgend definiert (vgl. [31], Kapitel 2):

(42) *LRLMI-Operator:*
A@→P...S $=_{def}$ Initial-Match(A).o.Left-to-Right(A).o.
Longest-Match(A).o.insertion(P,S)

Der Ausdruck in (42) klammert nach dem LRLM-Constraint Wörter aus L(A) in der Universalsprache mit den Wörtern aus L(P) (Präfix) und L(S) (Suffix).

Initial-Match, Left-to-Right und Insertion sind bereits in Abschnitt (6.2.4) definiert. Longest-Match erlaubt Klammerungen von Wörtern der zu klammernden regulären Sprache mit der Einschränkung (41-ii):

(43) Longest-Match(A) $=_{def}$
$\sim \$[<[[[A/[< \mid >] - [?^*[< \mid >]]]\&\$[>]]]$

In (43) bedeutet der Teilausdruck $[[A/[< \mid >] - [?^*[< \mid >]]]$, dass ausschließlich in der regulären Sprache L(A) alle spitzen Klammern ignoriert werden, und daher das letzte Zeichen keine spitze Klammer ignorieren darf. Wörter in L(A) die in der Universalsprache geklammert sind, dürfen in keiner anderen Klammerung enthalten sein. Auf diese Weise wird gewährleistet, dass die Wörter in L(A) in der Universalsprache geklammert werden, die die weiteste Ausdehnung besitzen. Dabei bezieht sich Longest-Match auf die durch Left-to-Right eingefügten spitzen Klammern.

Wenn die reguläre Relation R([ab|aba]@→{...}) auf die Eingabe L(ababa) angewendet wird, werden folgende Klammerungen vorgenommen:

(44) ababa
{aba}ba

Im Gegensatz zum Bidirectional-Constraint wird eine Eingabe in jedem Fall akzeptiert. Der Transduktor, der aus dem LRLMI-Operator resultiert, ist nicht in jedem Fall sequenziell. Dieser Fall tritt auf, wenn die zu klammernde Sprache nicht endlich ist.

Right-to-Left-Longest-Match-Constraint

Entsprechend dem LRLM-Constraints lässt sich auch ein Right-to-Left-Longest-Match-Constraint (RLLM-Constraint) definieren (vgl. [31], Abschnitt 1):

(45) *RLLM-Constraint:*
Es soll eine reguläre Sprache A hinsichtlich (i)-(iii) geklammert werden.

i. Ein Wort in A, dessen Suffix in der Universalsprache Präfix eines Wortes in A ist, wird nicht geklammert.
ii. Ein Wort in A, das in der Universalsprache ein echtes Suffix eines Wortes in A ist, wird nicht geklammert.
iii. Alle anderen Wörter in A werden in der Universalsprache geklammert.

Für den RLLM-Constraint soll der *Right-to-Left-Longest-Match-Insertion-Operator* (RLLMI-Operator) definiert werden, der sich aus dem LRLMI-Operator

ableiten lässt:

(46) *RLLMI-Operator*:
$A \rightarrow @P...S =_{def} Rev(Rev(A)@\rightarrow S...P)$

Wenn die reguläre Relation R([ba|aba]→@{...}) auf die Eingabe L(ababa) angewendet wird, werden folgende Klammerungen vorgenommen:

(47) ababa
ab{aba}

Auch der RLLMI-Operator erzeugt wie der LRLMI-Operator nicht in jedem Fall einen sequenziellen Transduktor.

6.2.5 Berechnung und Robustheit

Die Kompilation des LRLMI- und somit auch des RLLMI-Operators in einen Transduktor hat seine Tücken. Die Berechnung von Longest-Match beinhaltet eine Komplementbildung, die von dem zu klammernden Ausdruck abhängt. Ein Transduktor kann sich in diesem Fall stark aufblähen. Bei komplexeren Ausdrücken kann diese Operation praktisch nicht mehr durchführbar sein. Das Komponieren der einzelnen Teilschritte kann zudem verheerend sein, wenn der Transduktor, der aus Longest-Match resultiert, zu groß wird ([31], Abschnitt 2).

Die Operation Initial-Match ist dagegen unproblematisch. Das Einfügen der Zeichen ˆ vor den zu klammernden regulären Sprachen kann auf Seiten der Transduktoren mit einem Verfahren von Mohri & Sproat in [40] effizient realisiert werden. Diese Realisierung fordert lediglich eine Determinisierung von Σ^* in Konkatenation mit der zu klammernden Sprache ([40], Abschnitt 3.2).

Die Zeitkomplexität der Anwendung einer Kaskade auf eine Eingabe ist quadratisch, da die Komposition quadratisch ist.

Mit dem RLMI-Operator und dem LRLMI-Operator ist es möglich, eine reguläre Relation zu erstellen, die Chunk-Typen eindeutig klammert. Das Verfahren liefert zu einer nicht leeren Eingabe nie eine leere Analyse. Die Eingabe wird lediglich durch Informationen angereichert, hierbei ist die Analyse konstruktiv. Letztlich ist die Analyse wegen der grundlegenden Definition des Chunking robust.

6.2.6 Probleme mit morphologischen Ambiguitäten

Es soll eine ambige Eingabe betrachtet werden, deren Wörter die gleiche Länge haben. Der LRLMI-Operator und der RLLMI-Operator liefern bei der Anwendung der regulären Relation R([ab|b|ba|aba]@→{...}) bzw. R([ab|b|ba|aba] →@{...}) auf die ambige Eingabe L([a|b]b[a|b]) folgendes Ergebnis:

(48)

aba	bba	bbb	abb
{aba}	{b}{ba}	{b}{b}{b}	{ab}{b}

In (48) sind die Klammerungen für jedes Wort eindeutig. Der LRLM- und RLLM-Constraint beziehen sich hierbei nur lokal auf die einzelnen Wörter der ambigen Eingabe. Die Ambiguitäten in der Eingabe werden hierbei nicht aufgelöst. Gerade aus diesem Grund muss beim Chunking mit regulären Relationen eine eindeutige Eingabe vorausgesetzt werden. Später wird gezeigt, dass eine Eingabe disambiguiert werden kann, wenn sich Constraints global auf alle Wörter einer Eingabe beziehen.

6.2.7 Garden-Path-Effekt

Durch den LRLM- und RLLM-Constraint werden Chunks wie in (21-b), (23-b) und (24-b) ausgeschlossen. Durch die beiden Constraints kann es jedoch zu einem *Garden-Path-Effekt* kommen. Es kann vorkommen, dass eine Analyse angenommen wird, die später zu Verarbeitungsschwierigkeiten führen kann. Grund hierfür sind lokale Ambiguitäten, die theoretisch durch weiteren Kontext aufgelöst werden können ([18], S.3-8). Ein Beispiel dafür ist der Satz *The emergency crews really hate is family violence* (Abney [5], S.6):

(49) a. $\{_{NP}$ The$_{DTI}$ emergency$_{NN}$ $\}_{NP}$ $\{_{NP}$ crews$_{NNS}$ $\}_{NP}$ really$_{RB}$ hate$_{VB}$ is$_{BEZ}$ $\{_{NP}$ family$_{NN}$ violence$_{NN}$ $\}_{NP}$ $\cdot_{<\$.>}$

b. $\{_{NP}$ The$_{DTI}$ emergency$_{N}$ crews$_{N}$ $\}_{NP}$ really$_{RB}$ hate$_{VB}$ is$_{BEZ}$ $\{_{NP}$ family$_{NN}$ violence$_{NN}$ $\}_{NP}$ $\cdot_{<\$.>}$

In (49-b) werden durch den RLLM- und den LRLM-Constraint Chunks gebildet, die einen Garden-Path-Effekt hervorrufen, da durch die beiden Strategien *the emergency crews* als ein Chunk angesehen wird und so im Satz das Subjekt fehlt. In (49-a) dagegen kommt es zu keinem Garden-Path-Effekt.

Durch den LRLM- und RLLM-Constraint wird das Parsing-Prinzip der *Late Closure* verfolgt ([18], S.9):

(50) If grammatically permissible, attach new items into the clause or phrase currently being processed (i. e. , the clause or phrase postulated most recently).

Solch eine Strategie wird auch als gierig bezeichnet. Abney zeigt aber in [5] für das Englische, dass der LRLM-Constraint gute Ergebnisse liefert, auch wenn

es zu falschen Analysen kommen kann. Wir werden später sehen, dass sich das Problem des Garden-Path-Effekts lösen lässt, wenn Constraints auf eine andere Weise definiert werden.

6.3 Verbindung von Chunking und syntaktischem Tagging

Im Folgenden sollen die Vorteile des Chunkings mit regulären Relationen und des syntaktischen Taggings mit regulären Sprachen zusammengebracht werden. Das Chunking mit regulären Relationen hat den Vorteil, dass die Analyse robust ist (mit dem LRLM- und RLLM-Constraint). Das syntaktische Tagging mit regulären Sprachen ist hingegen nicht robust. Es beinhaltet jedoch eine syntaktische Analyse, die informativer ist, als das Chunking. Im Folgenden soll eine nicht ambige Eingabe durch das Chunking mit regulären Relationen analysiert werden. Es wird daher eine eindeutig getaggte Eingabe vorausgesetzt. Die gebildeten Chunks sollen dann durch syntaktische Tags in eine Dependenzstruktur eingegliedert werden. Durch dieses Verfahren wird eine durch Chunks gebildete Struktur mit zusätzlicher syntaktischer Information angereichert.

Es soll auf Techniken eingegangen werden, die von Megyesi & Rydin ([38]), Grefenstette ([20]) und Aït-Mokhtar & Chanod ([7]) angewendet werden, um Chunks in eine Dependenzstruktur einzugliedern, wobei die Gruppierungen, die in diesen Ansätzen behandelt werden, nicht unbedingt Chunks darstellen. Daher werden hier auch keine verwaisten Wörter behandelt. Nebensatzgrenzen werden in den Ansätzen nicht markiert.

6.3.1 Taggen von syntaktischen Köpfen

Wie in Abschnitt 2.3.1 diskutiert wurde, lässt sich aus einer Chunkstruktur eine Dependenzstruktur ableiten, die innerhalb des syntaktischen Taggings als lokale Dependenzstruktur angesehen werden kann. Um eine Dependenzstruktur aus einem Chunk ableiten zu können, müssen die syntaktischen Köpfe der Konstituenten bekannt sein.

Mit regulären Relationen ist es möglich, Köpfe innerhalb von Chunk-Typen zu markieren. Megyesi & Rydin in [38] und Grefenstette in [20] verwenden hierfür einen Conditional-Replacement-Operator. Für das Markieren von Köpfen soll das Tag @HEAD dienen.

Es soll das letzte Nomen in einem NP-Chunk als Kopf markiert werden:

(51) $0 \rightarrow$@HEAD$||$
[[NN|'NN$'|NNS|'NNS$'|NP|'NP$'|NPS|'NPS$']]_$\}_{np}$

Die Elemente in einem NP-Chunk, die nicht durch das Tag @HEAD markiert sind, hängen nun von dem durch das Tag @HEAD markierte Nomen ab. Solche Constraints können durch die Komposition zu einer Kopf-Markierung-Grammatik zusammengefasst werden.

6.3.2 Taggen von potentiellen syntaktischen Funktionen

Um Chunks in die Struktur des syntaktischen Taggings im Sinne eines Attachers einzugliedern, sollen Chunks mit syntaktischen Tags ausgezeichnet werden. Durch den Conditional-Replacement-Operator können Chunk-Typen mit einem syntaktischen Tag versehen werden ([38], Kapitel 2.3; [7], S.77).

Ein NP-Chunk kann potentielles Subjekt eines Satzes sein, wenn sein syntaktischer Kopf im Nominativ steht:

(52) $0 \rightarrow$@SUBJ$|||$[[NN|NNS|NP|NPS]@HEAD$\}_{np}]_$

In diesem Fall hat der Kopf eines NP-Chunks, die Funktion @SUBJ und hängt vom Hauptverb ab. Es können auch einzelnen Tags der Eingabe syntaktische Tags zugeordnet werden. Solche Constraints können durch die Komposition zu einer Tag-Einfügung-Grammatik verbunden werden.

6.3.3 Entfernen von potentiellen syntaktischen Funktionen

Nun gilt aber für Subjekte das Uniqueness Principle. Um dieses Prinzip zu realisieren, können Tags, die Subjekte markieren, rückgängig gemacht werden. Auch diese Constraints können mit dem Conditional-Replacement-Operator realisiert werden. Dabei werden spezielle Tags in einem bestimmten Kontext entfernt ([7], S.77). In dem folgenden Beispiel-Constraint wird angenommen, dass ein NP-Chunk nach einem finiten Verb nicht das Subjekt eines Satzes sein kann, wenn vor dem finiten Verb ein durch das Tag @SUBJ markierter NP-Chunk steht.

(53) @SUBJ$\rightarrow$0$|||$[$\{_{np}\sim$ \$[$\{_{np}|\}_{np}$] $\}_{np}$@SUBJ .. V[PRES|PAST].. $\{_{np}\sim$ \$[$\{_{np}|\}_{np}$]]$\}_{np}]\_$

Solche Constraints können nun durch die Komposition zu einer Tag-Entfernung-Grammatik zusammengefasst werden.

6.3.4 Attacher

Die Kopf-Markierung-Grammatik, die Tag-Einfügung-Grammatik und die Tag-Enfernung-Grammatik können nun zu einem Attacher zusammengefasst werden:

(54) Attacher $=_{def}$ Kopf_Markierung_Grammatik.o.
Tag_Einfügung_Grammatik.o.Tag_Enfernung_Grammatik

Durch den Attacher sollen nun Chunks in eine Dependenzstruktur eingegliedert werden. Daher soll der Attacher mit der Kaskade komponiert werden, die Chunks klammert:

(55) Parser $=_{def}$ Kaskade.o.Attacher

Die reguläre Relation R(Parser) kann nun auf eine Eingabe angewendet werden. Im folgenden Beispiel wird angenommen, dass nur NP-Chunks geklammert werden. Die Analyse für den Satz *the man loves the woman* würde folgendermaßen schrittweise analysiert werden wenn die Eigabe als DTI NN VB DTI NN <$.> getaggt ist:

(56)
1. *Eingabe:*
   The$_{DTI}$ man$_{NN}$ loves$_{VB}$ the$_{DTI}$ woman$_{NN}$.<$.>
2. *Kaskade:*
   $\{_{np}$ The$_{DTI}$ man$_{NN}$ $\}_{np}$ loves$_{VB}$ $\{_{np}$ the$_{DTI}$ woman$_{NN}$ $\}_{np}$.<$.>
3. *Kopf_Markierung_Grammatik:*
   $\{_{np}$ The$_{DTI}$ man$_{NN}$ @HEAD $\}_{np}$ loves$_{VB}$ $\{_{np}$ the$_{DTI}$ woman$_{NN}$ @HEAD $\}_{np}$.<$.>

4. *Tag_Einfügung_Grammatik:*
   $\{_{np}$ The$_{DTI}$ man$_{NN}$ @HEAD $\}_{np}$ @SUBJ loves$_{VB}$ $\{_{np}$ the$_{DTI}$ woman$_{NN}$ @HEAD $\}_{np}$ @SUBJ $._{<\$.>}$
5. *Tag_Enfernung_Grammatik:*
   $\{_{np}$ The$_{DTI}$ man$_{NN}$ @HEAD $\}_{np}$ @SUBJ loves$_{VB}$ $\{_{np}$ the$_{DTI}$ woman$_{NN}$ @HEAD $\}_{np}$ $._{<\$.>}$

Bei diesem Verfahren ist die Analyse sehr modular gehalten. Als erstes werden Chunks gebildet, dann werden diese syntaktisch getaggt. Das bedeutet, dass reguläre Ausdrücke für Chunks und für das syntaktische Tagging unabhängig voneinander entwickelt werden können. Hierbei müssen nicht zwingend nur Chunks getaggt werden. Es können auch einzelnen Tags syntaktische Funktionen zugewiesen werden.

6.3.5 Berechnung und Robustheit

Der Conditional-Replacement-Operator verlangt Komplementierungen der Kontexte. Daher können die resultierenden Transduktoren relativ viele Zustände haben. Der Transduktor, der einen Attacher realisiert, kann folglich sehr groß werden. Jedoch müssen bei einer nicht Ambigen Eingabe, die dieses Verfahren voraussetzt, nur syntaktische Tags behandelt werden.

Auch ein Transduktor, der eine Kaskade realisiert, kann sehr groß werden. Das Komponieren einer Kaskade mit einem Attacher kann daher schnell zu nicht mehr praktikablen Transduktoren führen. Bei der praktischen Anwendung sollten daher das Chunking und das Eingliedern von Chunks in eine Dependenzstruktur durch einen Attacher voneinander getrennt werden.

Die Zeitkomplexität der Anwendung einer Kaskade ist wie die Anwendung

eines Attachers auf eine Eingabe quadratisch. Die Zeitkomplexität der Analyse ist daher insgesamt quadratisch.

Das Verfahren ist aufgrund des Chunking mit regulären Relationen robust. Das Markieren von Köpfen und syntaktischen Funktionen reichert die Chunk-Analyse lediglich durch Informationen an. Das Entfernen von syntaktischen Tags bezieht sich dann nur genau auf diese eingefügten Informationen.

6.3.6 Garden-Path-Effekt

Durch dieses Verfahren ist es nicht möglich, die Garden-Path-Effekte, die durch das Chunking mit regulären Relationen auftreten, aufzulösen. Das syntaktische Tagging hat keinen Einfluss auf das Bilden von Chunks. Es kann Also kein weiterer Kontext verwendet werden, um einen Garden-Path-Effekt aufzulösen. Wünschenswert wäre hier eine Interaktion von Chunking und syntaktischem Tagging, die entweder eine Reanalyse oder eine parallele Verarbeitung möglich macht.

7 Robustes Parsing mit Gewichten

In diesem Kapitel wird erläutert, wie das syntaktische Tagging und das Chunking mit gewichteten regulären Sprachen bzw. gewichteten regulären Relationen realisiert werden kann. Danach wird das Chunking mit gewichteten regulären Relationen und das syntaktische Tagging mit gewichteten Regulären Sprachen zusammengeführt. Grundlage für die hier entwickelten Verfahren ist ein von Hanneforth [22] formulierter Ansatz zur Realisierung des Longest-Match-Constraints in Kartunnen [31] mit gewichteten Transduktoren. Es wird gezeigt, dass sich Probleme auf Gewichte umlagern lassen. Dadurch kann die

Zustandsanzahl von Transduktoren oder endlichen Automaten als gewichtete Varianten kleiner gehalten werden.

7.1 Bewertung anhand von Gewichten

Um gewichtete Wörter einer gewichteten regulären Sprache anhand ihres Gewichtes zu bewerten, können bestimmte Semiringe verwendet werden. Gleiches gilt für reguläre Relationen. Im Folgenden sollen jedoch nur gewichtete Wörter einer gewichteten regulären Sprache betrachtet werden. Verschiedene Semiringe können hierbei verschiedene Kriterien ausdrücken. Die gewichteten Wörter einer gewichteten regulären Sprache lassen sich nach diesen Kriterien ordnen. Somit können Teilmengen einer gewichteten Sprache gebildet werden. Im Folgenden soll die Teilmenge einer gewichteten regulären Sprache extrahiert werden, die ein Kriterium am besten erfüllt.

7.1.1 Bewertungssemiringe

Zur Vereinfachung wird hier ein additiv idempotenter Semiring $(S, \oplus, \otimes, \bar{0}, \bar{1})$ mit $\bar{0} \neq \bar{1}$, bei dem die Operation $\oplus$ eine lineare Ordnung über S definiert, als *Bewertungssemiring* bezeichnet. Ein Beispiel für einen Berwertungssemiring ist der *Tropische Semiring* $(\mathbf{R} \cup \{\infty\}, min, +, \infty, 0)$ mit ∞ als Nullelement und der Zahl 0 als Einselement ([42], S.339). Die Operation min ist die klassische Minimumbildung und die Operation $+$ die klassische Addition. Für die klassische Minimumbildung und die klassische Addition gilt folgendes:

(1) $min\{\infty, a\} = min\{a, \infty\} = a$ für alle $a, b \in \mathbf{R} \cup \{\infty\}$

(2) $\infty + a = a + \infty = \infty$ für alle $a, b \in \mathbf{R} \cup \{\infty\}$

Der Bewertungssemiring ist kommutativ, da die klassische Minimumbildung und die klassische Addition kommutativ sind und er ist additiv idempotent, da $min\{a, a\} = a$ für alle $a \in \mathbf{R}$ gilt. Gewichtete Wörter können über die Operation min linear geordnet werden. Die natürliche Ordnung ist hierbei die Ordnung der reellen Zahlen:

(3) $(min\{a, b\} = a) \Leftrightarrow (a \leq b)$ für alle $a, b \in \mathbf{R} \cup \{\infty\}$

Hierbei gilt $x < \infty$ für alle $x \in \mathbf{R}$. Mit diesem Bewertungssemiring kann das gewichtete Wort in einer gewichteten regulären Sprache mit dem kleinsten Gewicht ermittelt werden, wobei die Gewichte durch die rationalen Zahlen dargestellt werden ([42], S.339; [24], S.260).

Wenn im Zusammenhang mit rationalen Zahlen das gewichtete Wort mit dem größten Gewicht ermittelt werden soll, kann der Bewertungssemiring $(\mathbf{R} \cup \{-\infty\}, max, +, -\infty, 0)$ mit $-\infty$ als Nullelement und der Zahl 0 als Einselelement verwendet werden. Hierbei ist die Operation max die klassische Maximumbildung und die Operation $+$ die klassische Addition. Für die klassische Maximumbildung und die klassische Addition gilt folgendes:

(4) $max\{-\infty, a\} = max\{a, -\infty\} = a$ für alle $a, b \in \mathbf{R} \cup \{-\infty\}$,

(5) $\infty + a = a + -\infty = -\infty$ für alle $a, b \in \mathbf{R} \cup \{-\infty\}$.

Da die klassische Maximumbildung und die klassische Addition kommutativ sind ist auch dieser Bewertungssemiring kommutativ und er ist additiv idempotent, da $max\{a, a\} = a$ für alle $a \in \mathbf{R}$ gilt. Über die Operation max können Gewichtete Wörter linear geordnet werden, dabei ist hier die natürliche Ordnung die umgekehrte Ordnung der reellen Zahlen:

(6) $max\{a, b\} = a \Leftrightarrow (b \leq a)$ für alle $a, b \in \mathbf{R} \cup \{\infty\}$

Hierbei gilt $-\infty < x$ für alle $x \in \mathbf{R}$ ([24], S.260).

Im Folgenden soll von der Bestimmung des Wortes mit dem besten Gewicht gesprochen werden. Durch Bewertungsemiringe können spezielle Kriterien beschrieben werden, wie zum Beispiel der minimale Benzinverbrauch oder die minimale Dauer für eine Autobahnfahrt von einem Punkt A zu einem Punkt B. Hierbei geht es darum, ein Kriterium am besten zu erfüllen. Dabei ist eine lineare Ordnung über den Gewichten einer gewichteten regulären Sprache notwendig, um alle gewichteten Wörter anhand eines Kriteriums miteinander zu vergleichen. Kriterien sollen im Folgenden mit $\mathcal{K}$ bezeichnet werden.

Für die regulären Ausdrücken, die gewichtete reguläre Sprachen denotieren, soll nun ein Operator definiert werden, der die Ermittlung der Wörter einer gewichteten regulären Sprache mit dem besten Gewicht denotiert:

(7) *Bestes Gewicht:* ΦA

Es soll ein kleines Beispiel gegeben werden. Es wird das Kriterium $\mathcal{K}_{a\_Pref}$ angenommen:

(8) $\mathcal{K}_{a\_Pref}$:
Das Wort wird präferiert, dass die meisten Zeichen *a* enthält.

Dieses Kriterium kann durch den Bewertungssemiring $(\mathbf{N} \cup \{-\infty\}, max, +, -\infty, 0)$ beschrieben werden, der ein Subsemiring des Bewertungssemirings $(\mathbf{R} \cup \{-\infty\}, max, +, -\infty, 0)$ darstellt. Jedem Zeichen a soll das Gewicht 1 zugeordnet werden, da die Zeichen a präferiert werden sollen. Es soll daher ein regulärer Ausdruck erstellt werden, der die Zeichen a gewichtet:

(9) a_Pref $=_{def}$ $[\sim \$[a]a\ <\mathcal{K}_{a\_Pref},1>]^* \sim \$[a]$

Zudem soll der reguläre Ausdruck A angenommen werden:

(10) A $=_{def}$ [aca] | [bbb] | [aab] | [bab]

Die reguläre Sprache L(A) kann nun durch die gewichtete reguläre Sprache L(a_Pref) gewichtet werden, indem A und a_Pref miteinander geschnitten werden (Hadamard-Produkt):

(11) L(A & a_Pref) = $\{2aca, 0bbb, 2aab, 1bab\}$

Aus (10) resultiert, dass L(Φ[A & a_Pref]) = $\{2aca, 2aab\}$. Hierbei teilen sich die gewichteten Wörter $2aca$ und $2aab$ das beste Gewicht.

Um ein Kriterium auszudrücken, sind demnach zwei Dinge notwendig, einerseits ein entsprechender Bewertungssemiring und anderseits eine gewichtete reguläre Sprache, die Gewichte nach einem bestimmten Prinzip einstreut. Das ausschließen von Pfaden soll hier als Disambiguierung angesehen werden. Gerade das Disambiguieren anhand von Kriterien ist dann im Zusammenhang mit dem Chunking und dem syntaktischen Tagging interessant.

7.1.2 Berücksichtigung mehrerer Kriterien

Im Zusammenhang mit den Attachment-Ambiguitäten innerhalb des Chunking mit regulären Relationen wird es notwendig sein, anhand von mehrere Kriterien zu disambiguieren. Es soll also möglich sein, mehrere Kriterien, die durch Bewertungssemiringe beschrieben werden, zu berücksichtigen. Durch die Ordnung von Kriterien nach ihrer Relevanz wird letztlich eine Berücksichtigung mehrerer Kriterien möglich sein.

7.1.3 Kartesisches Produkt von Bewertungssemiringen

Um zwei Kriterien gleichzeitig zu betrachten, kann das *Kartesische Produkt* von zwei Bewertungssemiringen gebildet werden ([8], S.175ff.). Das Kartesische Produkt zweier Bewertungssemiringe ist die Vektorisierung der Domänen und Operationen der verbundenen Bewertungssemiringe. Aus diesem Kartesischen Produkt resultiert ein neuer Semiring (vgl. [8], S.49f.).

Gegeben sind zwei Bewertungssemiringe $S_1 = (A_1, \oplus_1, \otimes_1, \bar{0}_1, \bar{1}_1)$ und $S_2 = (A_2, \oplus_2, \otimes_2, \bar{0}_2, \bar{1}_2)$, das Kartesische Produkt der Domänen ist dann folgendermaßen definiert ([8], S.176, S.49):

(12) $$S = S_1 \times S_2 = (A_1 \times A_2, \oplus', \otimes', (\bar{0}_1, \bar{0}_2), (\bar{1}_1, \bar{1}_2)).$$

Die Operationen des Semirings S sind folgendermaßen definiert, wenn $(a_1, a_2) \in S_1 \times S_2$ und $(b_1, b_2) \in S_1 \times S_2$ gegeben sind[24]:

(13) $$(a_1, a_2) \oplus' (b_1, b_2) = (a_1 \oplus_1 b_1, a_2 \oplus_2 b_2)$$

(14) $$(a_1, a_2) \otimes' (b_1, b_2) = (a_1 \otimes_1 b_1, a_2 \otimes_2 b_2)$$

S_1 und S_2 sollen als Instanzen von S bezeichnet werden. Gerade weil die Instanzen völlig unabhängig voneinander sind, ist der Semiring S kommutativ, wenn seine Instanzen S_1 und S_2 kommutativ sind, und additiv bzw. multiplikativ idempotent, wenn seine Instanzen additiv bzw. multiplikativ idempotent sind.

[24] vgl. [8], S.49: dort ist das Kartesische Produkt als Komposition von Semiringen beschrieben.

7.1.4 Problem der partiellen Ordnung

Es wird angenommen, dass die Instanzen S_1 und S_2 eines Semirings $S = S_1 \times S_2$ Bewertungssemiringe und somit linear geordnet sind. Es soll gezeigt werden, dass der Semiring S selbst nur partiell geordnet ist und somit keinen Bewertungssemiring darstellt.

Eine natürliche Ordnung für S ist folgend definiert, wenn $(a_1, a_2) \in S_1 \times S_2$ und $(b_1, b_2) \in S_1 \times S_2$ gegeben sind (vgl. [8], S.50):

(15) $(a_1, a_2) \leq_S (b_1, b_2)$ wenn und nur wenn $(a_1 \oplus_1 b_1, a_2 \oplus_2 b_2)$ = (b_1, b_2) mit $a_1, b_1 \in S_1$ und $a_2, b_2 \in S_2$

Es soll ein kleines Beispiel gegeben werden. Wenn $S_1 = (\mathbf{N} \cup \{\infty\}, min, +, \infty, 0)$ und $S_2 = (\mathbf{N} \cup \{\infty\}, min, +, \infty, 0)$ zwei Bewertungssemiringe sind, dann bildet deren Kartesisches Produkt $S_1 \times S_2$ den Semiring $S = (\mathbf{N} \cup \{\infty\} \times \mathbf{N} \cup \{\infty\}, min', +', (\infty, \infty), (0, 0))$. min' und $+'$ sind die klassischen Operationen $+$ und min, nur entsprechend auf Paare ausgedehnt. Durch den Semiring S können zwei Kriterien, die durch S_1 und S_2 beschrieben werden, gleichzeitig berücksichtigt werden. Jedoch sind nicht alle Elemente der Menge $\mathbf{N} \cup \{\infty\} \times \mathbf{N} \cup \{\infty\}$ durch die Relation $\leq_S$ vergleichbar. Das Minimum der Werte (8,2) und (3,7) ist (3,2). Die Werte (8,2) und (3,7) können daher nicht geordnet werden.

Eine abstrakte Lösung eines Problems über solch einen Semiring enthält generell eine nicht vergleichbare Menge von Paaren. Mit einem Semiring, der aus dem Kartesischen Produkt zweier Bewertungssemiringe gebildet wird, können daher gewichtete Wörter einer gewichteten regulären Sprache nicht linear geordnet werden. Aber gerade eine lineare Ordnung ist für die Ermittlung eines Wortes mit dem besten Gewicht notwendig.

7.1.5 Ordnung von Bewertungssemiringen

Um zwei Kriterien gleichzeitig zu betrachten, können sie nach ihrer Relevanz geordnet werden. Gewichtete Wörter einer gewichteten regulären Sprache werden hierbei in erster Linie nach dem Kriterium mit der höchsten Relevanz geordnet. Wenn das Kriterium mit der höchsten Relevanz von gewichteten Wörtern gleichermaßen erfüllt wird, werden diese gewichteten Wörter nach dem Kriterium mit der zweit höchsten Relevanz geordnet.

Um zwei Kriterien unter einer bestimmten Ordnung zu betrachten, kann ein spezielles Kartesisches Produkt von zwei Bewertungssemiringen, die diese Kriterien ausdrücken, gebildet werden. Dieses wird im Folgenden als *Komposition* bezeichnet. Diese Operation unterscheidet sich vom Kartesischen Produkt von Bewertungssemiringen nur in der Behandlung der abstrakten Addition.

Es soll die Komposition der Domänen zweier Bewertungssemiringe gebildet werden. Gegeben sind zwei Bewertungssemiringe $S_1 = (A_1, \oplus_1, \otimes_1, \bar{0}_1, \bar{1}_1)$ und $S_2 = (A_2, \oplus_2, \otimes_2, \bar{0}_2, \bar{1}_2)$, die Komposition der Domänen der Bewertungssemiringe ist dann folgendermaßen definiert:

(16) $$S_{\downarrow} = S_1 \times^{\downarrow} S_2 = (A_1 \times A_2, \oplus^{\downarrow}, \otimes', (\bar{0}_1, \bar{0}_2), (\bar{1}_1, \bar{1}_2))$$

Die Operationen des Semirings $S_{\downarrow}$ sind folgendermaßen definiert, wenn $(a_1, a_2) \in A_1 \times A_2$ und $(b_1, b_2) \in A_1 \times A_2$ gegeben sind:[25]

(17) $$(a_1, a_2) \otimes' (b_1, b_2) = (a_1 \otimes_1 b_1, a_2 \otimes_2 b_2)$$

[25] vgl. [8], S.39f.: dort wird die Operation max behandelt.

$$(18) \qquad (a_1, a_2) \oplus^{\downarrow} (b_1, b_2) = \begin{cases} (a_1, a_2 \oplus_2 b_2) & \text{wenn } a_1 = b_1 \\ (a_1, a_2) & \text{wenn } a_1 \oplus_1 b_1 = a_1 \\ (b_1, b_2) & \text{wenn } a_1 \oplus_1 b_1 = b_1 \end{cases}$$

Die Instanz S_1 ist hierbei über der Instanz S_2 angeordnet. Das Kriterium, das durch die Instanz S_1 ausgedrückt wird hat demzufolge eine höhere Relevanz. Der Semiring $S_{\downarrow}$ ist kommutativ, wenn seine Instanzen S_1 und S_2 kommutativ sind und er ist additiv bzw. multiplikativ idempotent, wenn seine Instanzen additiv bzw. multiplikativ idempotent sind. Die Instanz S_2 bezieht sich nämlich gerade auf eine Teilmenge $U \subseteq A_1 \times A_2$, die nicht durch die Instanz S_1 geordnet werden kann. Durch den Semiring $S_{\downarrow}$ können nun die gewichteten Wörter einer gewichteten regulären Sprache linear geordnet werden. Der Semiring $S_{\downarrow}$ ist daher ein Bewertungssemiring, wenn seine Instanzen Bewertungssemiringe sind.

Durch die Operation $\times^{\downarrow}$ sollen auch mehrere Bewertungssemiringe auf die gleiche Weise geordnet werden können:

$$(19) \qquad S_{\downarrow} = (A_1, \oplus_1, \otimes_1, \bar{0}_1, \bar{1}_1) \times^{\downarrow} (A_2, \oplus_2, \otimes_2, \bar{0}_2, \bar{1}_2) \times^{\downarrow} \ldots \times^{\downarrow} (A_n, \oplus_n, \otimes_n, \bar{0}_n, \bar{1}_n)$$

Auf die einzelnen Gewichte der Instanzen des Bewertungssemirings $S_{\downarrow}$ kann dann in einem regulären Ausdruck Bezug genommen werden:

(20) $<a_1, a_2, \ldots, a_n>$ denotiert die rational Power Series $(a_1, a_2, \ldots, a_n)\epsilon$ mit $a_1 \in A_1, a_2 \in A_2 \ldots a_n \in A_n$

Für einen Ausdruck $<a_1, a_2 \ldots a_n, \bar{1}_{n+1}, \bar{1}_{n+2}, \ldots, \bar{1}_{n+i}>$ soll hierbei auch verkürzt $<a_1, a_2 \ldots a_n>$ geschrieben werden. Mit diesem Verfahren können also beliebig

viele Kriterien, die durch Bewertungssemiringe beschrieben werden, unter einer bestimmten Ordnung betrachtet werden:

(21) $$\mathcal{K}_{\downarrow} = \left| \begin{array}{l} \mathcal{K}_1 \\ \mathcal{K}_2 \\ . \\ . \\ . \\ \mathcal{K}_n \end{array} \right.$$

Hierbei besteht das Kriterium $\mathcal{K}_{\downarrow}$ aus mehreren Kriterien, die nach ihrer Relevanz geordnet sind. Um die Instanzen eines Bewertungssemirings, der aus der Komposition gebildet wurde, in einem regulären Ausdruck anzusprechen, ist es notwendig, die Ordnung der Instanzen zu kennen. Daher soll es auch möglich sein, Gewichte mit Angabe des Namens eines Kriteriums einer Instanz zuzuweisen. Hierbei Soll gerade der Kriterienname auf eine Instanz verweisen. Dieses wird durch eine Indexfunktion I realisiert, d. h. $I : \mathcal{K}_n \rightarrow \mathbf{N}$:

(22) $$<\mathcal{K}_n, \mathrm{a}_t> = <\mathrm{a}_1, \mathrm{a}_2, \ldots, \mathrm{a}_{I(\mathcal{K}_n)-1}, \mathrm{a}_{I(\mathcal{K}_n)}> \text{ mit } \mathrm{a}_{I(\mathcal{K}_n)} = \mathrm{a}_t$$

Falls ein Bewertungssemiring nicht aus anderen Bewertungssemiringen zusammengesetzt ist, ist er hierbei seine eigene Instanz.

Es soll ein Beispiel gegeben werden. Es wird das Kriterium $\mathcal{K}_{a\_Pref}$ aus (8) und das Kriterium $\mathcal{K}_{b\_Pref}$ angenommen:

(23) $\mathcal{K}_{b\_Pref}$:
Das Wort wird präferiert, dass die meisten Zeichen *b* enthält.

Das Kriterium $\mathcal{K}_{b\_Pref}$ wird durch den Bewertungssemiring $(\mathbf{N} \cup \{\infty\}, min, +, \infty, 0)$ und durch den regulären Ausdruck b_Pref ausgedrückt:

(24) b_Pref $=_{def}$ [$\sim$ \$[b]b <$\mathcal{K}_{b\_Pref}$,1>]* $\sim$ \$[b]

Da hier die Zeichen b präferiert werden sollen, wird jedem Zeichen b das Gewicht 1 zugeordnet. Die Kriterien sind folgend geordnet:

(25) $$\mathcal{K}_{\downarrow a\_b\_Pref} = \left| \begin{array}{l} \mathcal{K}_{a\_Pref} \\ \mathcal{K}_{b\_Pref} \end{array} \right.$$

Der reguläre Ausdruck, der das Kriterium $\mathcal{K}_{\downarrow a\_b\_Pref}$ ausdrückt ist nun folgend definiert:

(26) a_b_Pref $=_{def}$ a_Pref&b_Pref

Nun soll die reguläre Sprache L(A) aus (10) angenommen werden, die durch a_b_Pref gewichtet wird:

(27) L(A&a_b_Pref) = {(2,0)aca,(0,3)bbb,(2,1)aab,(1,2)bab}

In (27) erfüllt das gewichtete Wort (2,1)aab das Kriterium $\mathcal{K}_{\downarrow a\_b\_Pref}$ am besten. Durch das zusätzliche Kriterium $\mathcal{K}_{b\_Pref}$ ist hier eine weitere Disambiguierung möglich, da das gewichtete Wort (2,0)aca ausgeschlossen werden konnte.

7.1.6 Bestimmung des besten Pfades

Um das gewichtete Wort mit dem kleinsten Gewicht in einem gewichteten endlichen Automaten mittels eines Bewertungssemirings zu finden, können *single-source shortest-path Algorithmen* verwendet werden. Bei diesen Algorithmen

wird aber nur genau ein gewichtetes Wort ermittelt. Falls mehrere gewichtete Wörter gleichsam das kleinste Gewicht Teilen, wird nur eines davon zufällig berechnet. Diese Algorithmen verwenden in der Regel Tropische Semiringe. Die Algorithmen beziehen sich hierbei nur auf die Gewichte eines gewichteten endlichen Automaten. Das bedeutet, dass ein gewichteter endlicher Automat auf einen bewerteten Graphen abgebildet wird. Ein bewerteter Graph ist wie folgt definiert:

(28) G = (E, K, v) ist ein endlicher gerichteter Graph (über S), wobei S = $(S, \oplus, \otimes, \bar{0}, \bar{1})$ ein Semiring ist, falls (i)-(iii) erfüllt sind

i. E ist eine endliche Menge, die Menge der Ecken

ii. $K \subseteq E \times E$ ist eine endliche Menge, die Menge der Kanten

iii. $v : K \rightarrow S$ ist die Bewertungsfunktion

Wenn ein gewichteter Graph G durch einen wie hier definierten Bewertungssemiring bewertet wird, wird er auch als absorptiv bewertet bezeichnet, wenn für das Gewicht jedes Pfades p in G gilt, dass $v(p) \leq \bar{1}$. Genau dieser Fall ist hier interessant.

Wenn ein gewichteter Graph azyklisch ist, kann ein *Generic-Topological-Order-Single-Source-Shortest-Distance Algorithmus* verwendet werden, um den *kürzesten Pfad* zu ermitteln. Ein gewichteter Graph ist azyklisch, wenn der korrespondierende gewichtete endliche Automat azyklisch ist. Der gewichteter endlicher Automat ist dann azyklisch, wenn die korrespondierende gewichtete reguläre Sprache endlich ist.

Die Zeitkomplexität ist bei diesem Algorithmus linear. Hierbei wird jedoch angenommen, dass die Zeit für die Berechnung der Operationen $\oplus$ und $\otimes$ konstant ist. Es wird sich zeigen, dass dies nicht unbedingt der Fall sein muss.

Daher soll hier die Zeitkomplexität genauer angegeben werden. Die Zeitkomplexität des Generic-Topological-Order-Single-Source-Shortest-Distance Algorithmus ist $O(|K| + (T_{\oplus} + T_{\otimes})|E|)$, wobei $T_{\oplus}$ und $T_{\otimes}$ die schlechtesten Zeitkosten von $\oplus$ und $\otimes$ denotieren.

Mit diesem Algorithmus können auch längste Pfade berechnet werden. Es soll daher von der Bestimmung des *besten Pfades* gesprochen werden, da die hier verwendeten Bewertungssemiringe nicht unbedingt Tropische Semiringe sind. Im Folgenden soll der beste Pfad nur auf solchen azyklischen endlichen gerichteten Graphen berechnet werden ([42], S.336ff.).

7.2 Syntaktisches Tagging mit Gewichten

Im Abschnitt (6.1) werden Constraints durch reguläre Sprachen dargestellt, die wohlgeformte Abfolgen von Tags beschreiben. Das Verfahren ist nicht robust und das Zusammenfassen von Constraints zu einer Constraint-Grammatik kann auf Seiten der endlichen Automaten zu Platzproblemen führen. Mit gewichteten Sprachen ist es nun möglich, die Constraints auf eine andere Weise zu realisieren. Anstatt ungültige Abfolgen herauszufiltern, werden gültige Abfolgen durch ein Gewicht präferiert.

7.2.1 Constraints

Constraints beinhalten hier zu präferierende Abfolgen. Diese Präferierungen sollen durch das Kriterium $\mathcal{K}_{Tag\_Pref}$ realisiert werden:

(29) $\mathcal{K}_{Tag\_Pref}$:
Wörter in einer regulären Sprache A werden durch (i) geordnet.

i. Vordefinierte Abfolgen von Zeichen werden präferiert.

Dieses Kriterium kann durch den Bewertungsemiring $(\mathbf{N} \cup \{-\infty\}, max, +, -\infty, 0)$ beschrieben werden. Hierbei werden den zu präferierenden Tagabfolgen positive Gewichte zugewiesen. Die Gewichtszuweisungen müssen hierbei nicht obligatorisch sein.

Um bestimmte Abfolgen zu gewichten, kann der *Optional-Restriction-Operator* verwendet werden:

(30) *Optional-Restriction-Operator:*

- $A(\Rightarrow)B\_C =_{def}$ [?* B A C]* ?*
- $A(\Rightarrow)B\_ =_{def}$ [?* B A]* ?*
- $A(\Rightarrow)\_C =_{def}$ [?* A C]* ?*

Der Optional Restriction Operator in (30) denotiert eine reguläre Sprache, in der ein Auftreten eines Teilwortes aus L(A) von den Wörtern in L(B) geführt und den Wörtern in L(C) gefolgt werden kann, je nachdem ob B oder C oder beide regulären Ausdrücke angegeben sind.

Einem Tag, das in einem bestimmten Kontext auftritt, kann nun optional ein positives Gewicht zugewiesen werden:

(31) [A $<\mathcal{K}_{Tag\_Pref}, 1>$](⇒)B_C

Aus (31) resultiert eine gewichtete reguläre Sprache, die Tagabfolgen (Wörter) in L(A) präferiert. Es kann hierbei auch angegeben werden, wie stark die Tagabfolgen in L(A) in einem bestimmten Kontext präferiert werden sollen.

(32) $[\text{A } {<}\mathcal{K}_{Tag\_Pref}, 999{>}](\Rightarrow)\text{B\_C}$

Die Präferierung in (32) kann im Gegensatz zu der Präferierung in (31) als eine starke Präferierung angesehen werden.

Es sollen Constraints erstellt werden, die die Beispiel-Constraints (8-a)-(8-c) aus Abschnitt (2.1.5) realisieren. Die Beschreibung in (8-a) wird folgendermaßen dargestellt:

(33) $[\text{N } {<}\mathcal{K}_{Tag\_Pref}, 50{>}](\Rightarrow)\text{DET\_}$

und die in (8-b) folgendermaßen:

(34) $[[@/|@{<} \ |@{>}]{<}\mathcal{K}_{Tag\_Pref}, 50{>}](\Rightarrow)$
$[\text{V}[\text{PRES}|\text{PAST}].]\_[.\ \text{V}[\text{PRES}|\text{PAST}]]$

und die in (8-c) folgendermaßen:

(35) $[\text{N . @SUBJ } {<}\mathcal{K}_{Tag\_Pref}, 5{>}](\Rightarrow)\_[\sim [..\ \text{N}\ ..]\text{V}[\text{PRES}|\text{PAST}]]$

Das Uniqueness Principle für die syntaktische Kopf-Funktion @+FMAINV könnte folgendermaßen ausgedrückt werden:

(36) $[@+\text{FMAINV } {<}\mathcal{K}_{Tag\_Pref}, 50{>}](\Rightarrow)[@@\sim [..\ @+\text{FMAINV}\ ..]]\_$
$[\sim [..\ @+\text{FMAINV}\ ..]@@]$

Die Präferierung in den Beispielen (33), (34) und (36) ist stark, da das Uniqueness Principle als sehr wichtig zu bewerten ist, nach einem Artikel stets ein Nomen folgen kann und ein Satz nicht zwei nebeneinander liegende finite Verben beinhalten kann. In (35) dagegen ist das Vergeben einer Subjektfunktion in

diesem Kontext nicht sicher.

Um die Vergabe von syntaktischen Funktionen unterdrücken zu können, wird das syntaktische Tag @DUMMY eingeführt, das in Optionalität zu jedem syntaktischen Tag steht. Durch dieses Tag wird ausgedrückt, dass keine syntaktische Funktion vergeben ist. Durch die Präferierung von @DUMMY in den Constraints ist es nun möglich, Zuweisungen von syntaktischen Funktionen zu unterbinden. Es gäbe hierbei auch die Möglichkeit, die syntaktischen Tags optional anzugeben und durch negative Gewichtungen zu unterdrücken. Jedoch würde dies das Konstruieren der Constraints stark verkomplizieren.

7.2.2 Constraint-Grammatik

Die einzelnen Constraints können nun durch die Schnittoperation (Hadamard-Produkt) nach Definition (9) in Abschnitt (6.1.3) zusammengefasst werden. Die Anwendung der Grammatik G besteht nun aus dem Gewichten der Lesarten einer Eingabe S und dem Präferieren der Lesarten mit dem besten Gewicht:

(37) $S' =_{def} \Phi[S \& G]$

Die Eingabe aus Beispiel (1) in Abschnit (6.1.1) soll nun durch eine Constraint-Grammatik gewichtet werden, die aus den Constraints in (33), (34), (35) und (36) besteht, wobei hier zur Vereinfachung potentielle Nebensatzgrenzen weggelassen sind:

(38) @@
the DET
@
[[program N NOM SG <50>[@DUMMY|@SUBJ <5> |@OBJ]] |
[program V PRES NON-SG3 [@DUMMY|@+MAINV]]]
@
[[run V PRES SG3 [@DUMMY|@+MAINV] <50>] |
[run N NOM PL [@DUMMY|@SUBJ|@OBJ]]]
@@

Das Kriterium $\mathcal{K}_{Tag\_Pref}$ erfüllt die folgende Lesart am besten:

(39) @@
the DET
@
program N NOM SG @SUBJ
@
run V PRES SG3 @+MAINV
@@
<105>

7.2.3 Berechnung und Robustheit

Die gewichteten endlichen Automaten, die aus dem Optional-Restriction-Operator entstehen, werden nicht so groß wie die endlichen Automaten, die aus dem Restriction-Operator resultieren, da keine Komplementierung vorgenommen wird. Aus diesem Grund ist der Schnitt (Hadamard-Produkt) aller Constraints zu einer Constraint-Grammatik nicht so verheerend wie bei dem Ver-

fahren zum syntaktischen Tagging in Abschnitt (6.1). Jedoch lässt sich eine Constraint-Grammatik, die durch einen gewichteten endlichen Automaten beschrieben wird, nicht in jedem Fall determinisieren.

Nach der Anwendung einer Constraint-Grammatik auf eine Eingabe muss der beste Pfad berechnet werden. Diese Berechnung ist hierbei linear und ändert nicht die ursprünglichen Komplexität der Anwendung einer Constraint-Grammatik, die durch die Komposition quadratisch ist.

Bei dem Verfahren zum syntaktischen Tagging in Abschnitt (6.1) ist eine robuste Verarbeitung nicht gewährleistet. Die Constraint-Grammatik, die durch eine reguläre Sprache dargestellt wird und gültige Abfolgen beschreibt, akzeptiert nicht in jedem Fall eine Eingabe. Bei dem syntaktischen Tagging mit gewichteten regulären Sprachen ist das Ergebnis der Anwendung einer Constraint-Grammatik auf eine nicht leere Eingabe nie leer. Die Eingabe muss nicht unbedingt ein wohlgeformtes Wort enthalten, um akzeptiert zu werden. Wenn ein Teil eines längeren Satzes, der mehrere Nebensätze enthält, nicht wohlgeformt ist, kann der Satz trotzdem teilweise disambiguiert werden. Bis zur Bestimmung des Wortes mit dem besten Gewicht werden hierbei keine Lesarten entfernt. Das Verfahren ist bis dahin konstruktiv. Die Eingabe wird lediglich mit Gewichtungen angereichert. Durch die Bestimmung des Wortes mit dem besten Gewicht ist letztlich gewährleistet, dass die Analyse nicht leer ist. An diesem Punkt ist das Verfahren reduktionistisch.

7.3 Chunking mit Gewichten

Im Abschnitt (6.2) werden reguläre Relationen verwendet, um Chunks und deren Struktur durch Klammerungen zu bilden. Hierbei kann es beim Chunking zu Attachment-Ambiguitäten kommen. Um einzelne Chunk-Typen eindeutig zu

klammern, werden Constraints in reguläre Relationen kompiliert, die dann nach einem bestimmten Prinzip eine eindeutige Klammerung gewährleisten. Diese Constraints können zu Garden-Path-Effekten führen. Hierbei wird jedoch davon ausgegangen, dass die Eingabe nicht ambig ist, da eine ambige Eingabe nicht disambiguiert werden kann, weil sich die Constraints nur lokal auf die einzelnen Wörter einer Eingabe beziehen.

Mit gewichteten regulären Sprachen ist es nun möglich, die Constraints auf eine andere Weise zu definieren als in Abschnitt (6.2). Constraints werden hier nicht mit Hilfe von Suffix- und Präfixeigenschaften definiert, sondern anhand von Kriterien, die durch Semiringe ausgedrückt werden können. Durch diese Kriterien ist es dann auch möglich, eine ambige Eingabe zu disambiguieren, da sich die Kriterien global auf alle Wörter der Eingabe beziehen können.

Chunk-Typen sollen hier durch gewichtete Relationen geklammert und gewichtet werden. Die gewichteten Relationen, die die einzelnen Chunk-Typen beschreiben, können dann wie in Abschnitt (6.2.2) im Sinne einer Kaskade komponiert werden. Eine gewichtete reguläre Relation, die Chunks und deren Struktur klammert, soll dann auf eine Eingabe angewendet werden. Nach der Anwendung kann dann das Wort ermittelt werden, das die aufgestellten Kriterien am besten erfüllt. Im Folgenden sollen Kriterien aufgestellt werden, die Attachment-Ambiguitäten innerhalb des Chunkings mit regulären Relationen auflösen und so eindeutige Klammerungen liefern.

Im ersten Schritt soll eine reguläre Sprache, die einen Chunk-Typ beschreibt, ohne Einschränkungen in der Universalsprache geklammert werden. Es ist hierbei nicht notwendig, die reguläre Sprache obligatorisch zu klammern. Chunk-Typen können also mit dem Optional-Insertion-Operator geklammert werden. Jedoch müssen Gewichte eingeführt werden, um bestimmte Kriterien auszu-

drücken. Um die Gewichtungen durch reguläre Ausdrücke K_1, K_2, ..., K_n zu realisieren, soll der Optional Insertion Operator als *Optional-Criterion-Insertion-Operator* (OCI-Operator) definiert werden:

(40) *OCI-Operator*:
$A(\rightarrow)P...S//K_1,K_2, ... ,K_n =_{def}$
$[[\sim \$[<|>]][[0.x.<]A[0.x.>]]]^* \sim \$[<|>]].o.$
$Id(K_1).o.Id(K_2).o. ... .o.Id(K_n).o.[< \rightarrow P].o.[> \rightarrow S]$

In (40) kann eine reguläre Relation, die Chunk-Typen klammert, durch mehrere gewichtete reguläre Sprachen gewichtet werden, die einzelne Kriterien realisieren. Die Kriterien können sich hierbei auf die eingefügten spitzen Klammern beziehen. Die spitzen Klammern werden dann durch gewünschte reguläre Sprachen ersetzt, die den Chunk-Typ angeben. Das Gewichten soll hierbei in den Operator integriert werden, damit sich die gewichteten regulären Sprachen nur auf die Klammerungen beziehen, die einen Chunk markieren. Nach dem Ersetzen der spitzen Klammern ist dies ohne Angabe der regulären Sprache L(A) nicht mehr möglich. Die gewichteten Sprachen, die Kriterien ausdrücken, sollen aber unabhängig von der zu klammernden Sprache modelliert werden.

7.3.1 Kriterien

Im Folgenden sollen Kriterien aufgestellt und durch gewichtete endliche Automaten realisiert werden. Hierbei wird angenommen, dass alle Wörter der Eingabe die gleiche Länge haben.

Exhaustive-Match-Kriterium

Bei dem Gruppieren von Chunks gilt das Prinzip Chunk Inclusiveness. Chunks, die durch eine reguläre Sprache beschrieben werden, sollen demnach auch geklammert werden. Daher ist das Kriterium $\mathcal{K}_{Ex\_Match}$ folgend definiert:

(41) $\mathcal{K}_{Ex\_Match}$:

Wörter in einer regulären Sprache A werden durch (i) geordnet.

i. Umso mehr Zeichen ein Wort in A innerhalb von Klammerungen enthält, desto stärker wird es präferiert.

Mit Hilfe von gewichteten Relationen ist es nun möglich, mit einem geeigneten Bewertungssemiring Zeichen innerhalb von Klammerungen zu zählen. Hierfür wird der Bewertungssemiring $(\mathbf{N} \cup \{-\infty\}, max, +, -\infty, 0)$ verwendet. Jedes Zeichen innerhalb einer Klammerung erhält das Gewicht 1. Somit stellt das Wortpaar einer gewichteten regulären Relation mit dem größten Gewicht das Wortpaar, das die meisten umschlossenen Zeichen im Wertebereich beinhaltet, dar (vgl. [22], Abschnitt „Das neue Verfahren“).

Eine reguläre Relation kann gewichtet werden, wenn sie mit der Identitätsrelation einer gewichteten regulären Sprache komponiert wird. Um das Kriterium $\mathcal{K}_{Ex\_Match}$ auszudrücken, ist ein gewichteter endlicher Automat durch *Exhaustive_Match* definiert, der Klammerinhalte gewichtet:

(42) Exhaustive_Match $=_{def}$

$[\sim \$[<|>]<[[?-[<|>]]<\mathcal{K}_{Ex\_Match}, 1>]^{*}>]^{*} \sim \$[<|>]$

Bei diesem Verfahren werden erst alle Klammerungsvarianten berechnet und gewichtet. Dann wird aus allen Klammerungsabfolgen die Klammerungsabfol-

ge mit dem besten Gewicht anhand des Kriteriums $\mathcal{K}_{Ex\_Match}$ herausgefiltert. Wenn die gewichtete reguläre Relation R([ab|b|ba|aba](→){...}// Exhaustive_Match) auf die Eingabe L(aba) angewendet wird, werden folgende Klammerungen und Gewichtungen vorgenommen:

(43)
$$\begin{array}{ccccc} 0aba & 0aba & 0aba & 0aba & 0aba \\ 1a\{b\}a & 2a\{ba\} & 2\{ab\}a & 3\{aba\} & 0aba \end{array}$$

In (43) erfüllt das gewichtete Wort $3\{aba\}$ das Kriterium $\mathcal{K}_{Ex\_Match}$ am besten.

Das Kriterium $\mathcal{K}_{Ex\_Match}$ reicht allerdings nicht für eine eindeutige Klammerung aus. Wenn die gewichtete reguläre Relation R([a+](→){...}// Exhaustive_Match) auf die Eingabe L(aa) angewendet wird, werden folgende Klammerungen und Gewichtungen vorgenommen:

(44)
$$\begin{array}{ccccc} 0aa & 0aa & 0aa & 0aa & 0aa \\ 0aa & 1a\{a\} & 1\{a\}a & 2\{a\}\{a\} & 2\{aa\} \end{array}$$

Hierbei erfüllen die gewichteten Wörter $2\{aa\}$ und $2\{a\}\{a\}$ das Kriterium $\mathcal{K}_{Ex\_Match}$ gleichermaßen.

Economic-Match-Kriterium

Bei dem Gruppieren von Chunks gilt das Prinzip Chunk Connectedness. Demnach muss ein funktionales Element zu dem Chunk gruppiert werden, den es definiert. Zudem soll hier die Selektion des thematischen Elements präferiert werden, das zu der größten Gruppierung führt. Daher sollen so viele Zeichen wie möglich in einer Klammerung untergebracht werden. Es soll das Kriterium $\mathcal{K}_{Ec\_Match}$ eingeführt werden:

(45) $\mathcal{K}_{Ec\_Match}$:

Wörter in einer regulären Sprache A werden durch (i) geordnet.

i. Umso weniger Klammerungen ein Wort in A enthält, desto stärker wird es präferiert.

Das Kriterium $\mathcal{K}_{Ec\_Match}$ soll mit Hilfe des Bewertungssemirings $(\mathbf{N} \cup \{\infty\}, min, +, \infty, 0)$ ausgedrückt werden. Jede Klammer erhält das Gewicht 1. Somit stellt das Wort mit dem besten Gewicht das Wort, dass die wenigsten Klammern beinhaltet, dar.

Die gewichtete Sprache, die Klammern gewichtet, und das Kriterium $\mathcal{K}_{Ec\_Match}$ realisiert, wird durch *Economic_Match* erzeugt:

(46) Economic_Match $=_{def}$
$[\sim \$[<|>][<]<\mathcal{K}_{Ec\_Match}, 1> \sim \$[<|>][>]<\mathcal{K}_{Ec\_Match}, 1>]^*$
$\sim \$[<|>]$

Nun sollen die Kriterien $\mathcal{K}_{Ex\_Match}$ und $\mathcal{K}_{Ec\_Match}$ zusammengeführt werden. Das Kriterium $\mathcal{K}_{Ec\_Match}$ darf nur angewendet werden, wenn durch das Kriterium $\mathcal{K}_{Ex\_Match}$ nicht disambiguiert werden konnte. Es wäre möglich, beide Kriterien in den Bewertungssemiring $(\mathbf{N} \cup \{\infty\}, min, +, \infty, 0)$ zu kodieren. Hierbei müsste das Kriterium $\mathcal{K}_{Ex\_Match}$ in diesem Tropischen Semiring durch negative Gewichte dargestellt werden (vgl. [22], Abschnitt „Das neue Verfahren“). Jedoch würde hierbei eine klare Trennung der Kriterien verlorengehen. Die Ordnung, die zwischen den beiden Kriterien besteht, müsste in diesem Fall in Zahlen kodiert sein. Die Kriterien $\mathcal{K}_{Ex\_Match}$ und $\mathcal{K}_{Ec\_Match}$ können auch durch die Operation $\times^{\downarrow}$ auf die entspechenden Bewertungssemiringe geordnet werden. Durch den resultierenden Bewertungssemiring können dann beide Kri-

terien unter einer bestimmten Ordnung betrachtet werden:

(47) $\mathcal{K}_{\downarrow Ex\_Ec\_Match} \left| \begin{array}{l} \mathcal{K}_{Ex\_Match} \\ \mathcal{K}_{Ec\_Match} \end{array} \right.$

Das Kriterium $\mathcal{K}_{\downarrow Ex\_Ec\_Match}$ wird durch den regulären Ausdruck Ex_Ec_Match ausgedrückt:

(48) Ex_Ec_Match $=_{def}$ Exhaustive_Match&Economic_Match

Wenn die gewichtete reguläre Relation R(a^{+}(→){...}//Ex_Ec_Match) auf die Eingabe L(aa) angewendet wird, werden folgende Klammerungen und Gewichtungen vorgenommen.

(49)
$$\begin{array}{lllll} (0,0)aa & (0,0)aa & (0,0)aa & (0,0)aa & (0,0)aa \\ (0,0)aa & (1,2)a\{a\} & (1,2)\{a\}a & (2,4)\{a\}\{a\} & (2,2)\{aa\} \end{array}$$

In (49) erfüllt das gewichtete Wort $(2,2)\{aa\}$ das Kriterium $\mathcal{K}_{\downarrow Ex\_Ec\_Match}$ am besten.

Es soll eine ambige Eingabe betrachtet werden. Wenn die gewichtete reguläre Relation R([[ab]|[b]|[ba]|[aba]](→){...}//Ex_Ec_Match) auf die Eingabe L([a|b]b[a|b]) angewendet wird, werden folgende Klammerungen und Gewichtungen vorgenommen:

(50)
$$\begin{array}{llll} (0,0)aba & (0,0)aba & (0,0)aba & (0,0)aba \\ (3,2)\{aba\} & (2,2)b\{ba\} & (3,6)\{b\}\{b\}\{b\} & (2,2)\{ab\}b \end{array}$$

In (50) erfüllt das gewichtete Wort $(3,2)\{aba\}$ das Kriterium $\mathcal{K}_{\downarrow Ex\_Ec\_Match}$ am besten. Die Kriterien beziehen sich also global auf alle Wörter der Eingabe. So

wird eine Disambiguierung der Eingabe möglich.

Wenn die gewichtete reguläre Relation R([[aba]|[aaa]](→){...}// Ex_Ec_Match) auf die Eingabe L(a[b|a]a) angewendet wird, werden folgende Klammerungen und Gewichtungen vorgenommen:

(51)
$$\begin{array}{cc} (0,0)aba & (0,0)aaa \\ (3,2)\{aba\} & (3,2)\{aaa\} \end{array}$$

Hierbei erfüllen die gewichteten Wörter $(3,2)\{aba\}$ und $(3,2)\{aaa\}$ gleichermaßen das Kriterium $\mathcal{K}_{\downarrow Ex\_Ec\_Match}$. Die Klammerung ist zwar eindeutig, jedoch ist die Eingabe nicht disambiguiert.

Selection-Match-Kriterium

Es soll möglich sein, bestimmte Abfolgen innerhalb von Chunk-Typen zu disambiguieren. Bestimmte Wörter einer regulären Sprache, die einen Chunk-Typ beschreibt, sollen präferiert werden. Die Präferierung wird durch das Kriterium $\mathcal{K}_{Se\_Match}$ ausgedrückt.

(52) $\mathcal{K}_{Se\_Match}$:
Wörter in einer regulären Sprache A werden durch (i) geordnet.

i. Vordefinierte Abfolgen von Zeichen werden präferiert.

Die Präferierung innerhalb der Sprachen der Chunk-Typen soll durch den Bewertungssemiring $(\mathbf{R} \cup \{-\infty\}, max, +, -\infty, 0)$ beschrieben werden. Die bis hier aufgeführten Kriterien werden nun folgendermaßen geordnet:

(53) $$\mathcal{K}_{\downarrow Ex\_Ec\_Se\_Match} = \Big\downarrow \begin{array}{l} \mathcal{K}_{Ex\_Match} \\ \mathcal{K}_{Ec\_Match} \\ \mathcal{K}_{Se\_Match} \end{array}$$

Das Kriterium $_{Se\_Match}$ wird hierbei durch Gewichtungen in der zu klammernden gewichteten regulären Sprache ausgedrückt.

Wenn die gewichtete reguläre Relation R([[aba<$\mathcal{K}_{Se\_Match}$, 1>]|[aaa]](→){...}//Ex_Ec_Match) auf die Eingabe L(a[b|a]a) angewendet wird, werden folgende Klammerungen und Gewichtungen vorgenommen:

(54) $$\begin{array}{ll} (0,0,1)aba & (0,0,0)aaa \\ (3,2,1)\{aba\} & (3,2,0)\{aaa\} \end{array}$$

In (54) erfüllt das gewichtete Wort $(3,2,1)\{aba\}$ das Kriterium $\mathcal{K}_{Se\_Match}$ am besten. Die Klammerung ist eindeutig und die Eingabe ist disambiguiert.

Global-Left-to-Right-Kriterium

Mit den bisher aufgeführten Kriterien ist immer noch keine eindeutige Klammerung gewährleistet.

Wenn die gewichtete reguläre Relation R([aba](→){...}//Ex_Ec_Match) auf die Eingabe L(ababa) angewendet wird, werden folgende Klammerungen und Gewichtungen vorgenommen:

(55) $$\begin{array}{ll} (0,0,0)ababa & (0,0,0)ababa \\ (3,2,0)\{aba\}ba & (3,2,0)ab\{aba\} \end{array}$$

In (55) erfüllen beide gewichtete Wörter das Kriterium $\mathcal{K}_{\downarrow Ex\_Ec\_Se\_Match}$ gleichermaßen.

Es wäre eine Präferierung wie bei der Anwendung des LRLMI-Operators wünschenswert. Hanneforth verwendet in [22] gerade für diese Disambiguierung den Left-to-Right-Constraint. Klammerungen sollen daher mit dem LRI-Operator vorgenommen werden. Dazu soll der LRI-Operator als *Left-to-Right-Criteria-Operator* (LRCI-Operator) entsprechend dem OCI-Operator auf die Verwendung von Kriterien angepasst werden:

(56) *LRCI-Operator*:
A@(→)P...S//K_1,K_2, ... ,K_n $=_{def}$
Initial-Match(A).o.Left-to-Right(A).o.Insertion(P,S).o.
Id(K_1).o.Id(K_2).o. ... .o.Id(K_n).o.$[_< \rightarrow$ P$]$.o.$[_> \rightarrow$S$]$

Wenn die reguläre Relation R($[$aba$]$@(→){...}//Ex_Ec_Match) auf die Eingabe L(ababa) angewendet wird, wird die folgende Klammerung und Gewichtung vorgenommen:

(57) $(0,0,0)ababa$
$(3,2,0)\{aba\}ba$

Die Klammerung ist in diesem Fall eindeutig.

Wenn aber die gewichtete reguläre Relation R($[$aba$]$@(→){...}// Ex_Ec_Match) auf die Eingabe L($[$a|b$]$baba) angewendet wird, werden folgende Klammerungen und Gewichtungen vorgenommen:

(58) $(0,0,0)ababa$ $(0,0,0)bbaba$
$(3,2,0)\{aba\}ba$ $(3,2,0)bb\{aba\}$

Durch den LRCI-Operator können die ambigen Klammerungen in (58) nicht aufgelöst werden. Es soll daher das Kriterium $\mathcal{K}_{LR\_Match}$ definiert werden, das

sich global auf alle Wörter der Eingabe bezieht:

(59) $\mathcal{K}_{LR\_Match}$:
Wörter in einer regulären Sprache A werden durch (i) geordnet.

i. Umso weiter links eine Klammerung in einem Wort in A steht, desto stärker wird das Wort präferiert.

Das Kriterium $\mathcal{K}_{LR\_Match}$ soll durch den Bewertungssemiring $(\{0,1\}^* \cup \infty, \wedge^{\rightarrow}, \cdot, \infty, \epsilon)$ ausgedrückt werden. Die Operation $\cdot$ ist die klassische Konkatenation von Wörtern mit ϵ als neutrales Element. In diesem Fall bestehen Wörter aus den Zeichen 0 und 1. Die 1 steht für eine öffnende Klammer in einem Wort, die 0 steht für jedes Zeichen in einem Wort der Eingabe ausgenommen der öffnenden und schließenden Klammer. Das Element ∞ repräsentiert das unendlich lange Wort aus den Zeichen 0. Die Operation $\wedge^{\rightarrow}$ ist folgendermaßen definiert, wenn $A \in \{0,1\}^* \cup \infty$ und $B \in \{0,1\}^* \cup \infty$ zwei Wörter sind:

(60)
$$A \wedge^{\rightarrow} B = \begin{cases} A & \text{wenn} \quad |A| = |B|, A = w1v \text{ und } B = w0g \\ & \quad \text{mit } w,v,g \in \{0,1\}^* \\ B & \text{wenn} \quad |A| = |B|, A = w0v \text{ und } B = w1g \\ & \quad \text{mit } w,v,g \in \{0,1\}^* \\ A & \text{wenn} \quad |A| < |B| \\ B & \text{wenn} \quad [B| < |A| \\ A & \text{wenn} \quad A = B \end{cases}$$

Aus (60) resultiert, dass das unendlich lange Wort ∞ das neutrale Element der Operation $\wedge^{\rightarrow}$ ist

(61) $\infty \wedge^{\rightarrow} a = a \wedge^{\rightarrow} \infty = a$ für alle $a, b \in \{0,1\}^* \cup \infty$

und dass der Bewertungssemiring idempotent ist, da

(62) $a \wedge^{\rightarrow} a = a$ für alle $a \in \{0,1\}^* \cup \infty$.

Es gelten die Distributivgesetze, da sich die Operation $\wedge^{\rightarrow}$ auf Präfixeigenschaften bezieht und diese durch die Konkatenation erhalten bleiben:

(63) $a \cdot (b \wedge^{\rightarrow} a) = a \cdot b \wedge^{\rightarrow} a \cdot c$ und $(b \wedge^{\rightarrow} c) \cdot a = b \cdot a \wedge^{\rightarrow} c \cdot a$ für alle $a, b, c \in \{0,1\}^* \cup \infty$

Über der Menge $\{0,1\}^* \cup \infty$ ist über $\wedge^{\rightarrow}$ eine lineare Ordnung definiert:

(64) $(a \wedge^{\rightarrow} b = a) \Leftrightarrow (a \leq b)$ für alle $a, b \in \{0,1\}^* \cup \infty$

Der Semiring ist jedoch nicht kommutativ, da die Konkatenation nicht kommutativ ist. Die Komposition von gewichteten Transduktoren fordert jedoch Kommutativität.

Die Konkatenation ist kommutativ, wenn einer der Operanden ϵ ist. Hierbei gilt $\epsilon \cdot a = a \cdot \epsilon$ mit $a \in \{0,1\}^* \cup \infty$. Genau dieser Spezialfall soll hier betrachtet werden. Ein Operand der Komposition muss also das Gewicht ϵ haben. Dies ist auch ein Grund dafür, dass die Gewichtszuweisungen nicht für einzelne Chunk-Typen gemacht werden können, wenn sie zu einer Kaskade komponiert werden sollen. Die Gewichtszuweisungen sollen also vollzogen werden, nachdem die Chunk-Typen zu einer Kaskade komponiert wurden. Hierbei soll sich die gewichtete reguläre Sprache, die das Kriterium $\mathcal{K}_{LR\_Match}$ beschreibt, auf die Klammertypen (Left_Brackets, Right_Brackets) der Chunks beziehen.

Die gewichtete Sprache, die linke Klammerungen präferiert, wird durch *Global_Left_to_Right* erzeugt:

(65) Global_Left_to_Right(Left_Brackets,Right_Brackets) $=_{def}$
[[$\sim$ \$[Left_Brackets|Right_Brackets].o.[? <$\mathcal{K}_{LR\_Match}$, 0>]*]
[[Left_Brackets <$\mathcal{K}_{LR\_Match}$, 1>]|
[Right_Brackets <$\mathcal{K}_{LR\_Match}$, ϵ>]]]*
[$\sim$ \$[Left_Brackets|Right_Brackets].o.[? <$\mathcal{K}_{LR\_Match}$, 0>]*]

Die Kriterien $\mathcal{K}_{Ex\_Match}$, $\mathcal{K}_{Ec\_Match}$, $\mathcal{K}_{Se\_Match}$ und $\mathcal{K}_{LR\_Match}$ können nun geordnet werden:

(66) $$\mathcal{K}_{\downarrow Ex\_Ec\_Se\_GLtR\_Match} = \Big\downarrow \begin{array}{l} \mathcal{K}_{Ex\_Match} \\ \mathcal{K}_{Ec\_Match} \\ \mathcal{K}_{Se\_Match} \\ \mathcal{K}_{LR\_Match} \end{array}$$

Wenn nun die gewichtete reguläre Relation R([aba](→){...}//Ex_Ec_Match) durch L(Global_Left_to_Right([{],[}])) gewichtet und auf die Eingabe L([a|b]baba) angewendet wird, werden folgende Klammerungen und Gewichtungen vorgenommen:

(67) $(0,0,0,\epsilon)ababa$ $(0,0,0,\epsilon)bbaba$
$(3,2,0,100000)\{aba\}ba$ $(3,2,0,001000)bb\{aba\}$

$(0,0,0,\epsilon)ababa$
$(3,2,0,001000)ab\{aba\}$

Das gewichtete Wort $(3,2,0,100000)\{aba\}ba$ in (67) erfüllt letztlich das Kriterium $\mathcal{K}_{\downarrow Ex\_Ec\_Se\_GLtR\_Match}$ am besten. Durch das Global-Left-to-Right-Kriterium konnten die Klammerungen disambiguiert werden.

Es soll darauf hingewiesen werden, dass die Berechnung der Operation $\wedge^{\rightarrow}$ nicht konstant, sondern linear ist. Hierbei müssen im schlechtesten Fall die Zeichenketten beider Operanden vom Anfang bis zum Ende der kürzeren Zeichenkette durchgegangen werden. Dabei ändert sich natürlich die Komplexität der Berechnung des besten Pfades.

Global-Right-to-Left-Kriterium

Korrespondierend zum Kriterium $\mathcal{K}_{LR\_Match}$ lässt sich auch das Kriterium $\mathcal{K}_{RL\_Match}$ definieren:

(68) $\mathcal{K}_{RL\_Match}$:
Wörter in einer regulären Sprache A werden durch (i) geordnet.

i. Umso weiter rechts eine Klammerung in einem Wort in A steht, desto stärker wird das Wort präferiert.

Das Kriterium $\mathcal{K}_{RL\_Match}$ soll durch den Semiring $(\{0,1\}^* \cup \infty, \wedge^{\leftarrow}, \cdot, \infty, \epsilon)$ ausgedrückt werden. Die Operation $\cdot$ ist die Konkatenation von Wörtern mit ϵ als Einselement. Eine 1 steht hier für eine schließende Klammer in einem Wort und die 0 steht für jedes Zeichen in einem Wort der Eingabe ausgenommen der öffnenden und schließenden Klammer. Das Element ∞ repräsentiert auch hier das unendlich lange Wort aus den Zeichen 0. Die Operation $\wedge^{\leftarrow}$ ist folgendermaßen definiert, wenn $A \in \{0,1\}^* \cup \infty$ und $B \in \{0,1\}^* \cup \infty$ zwei Wörter sind:

(69) $A \wedge^{\leftarrow} B = rev(\ rev(A) \wedge^{\rightarrow} rev(B)\ )$

Aus der Operation $\wedge^{\rightarrow}$ resultiert, dass ∞ auch neutrales Element der Operation $\wedge^{\leftarrow}$ ist

(70) $\infty \wedge^{\leftarrow} a = a \wedge^{\leftarrow} \infty = a$ für alle $a, b \in \{0,1\}^* \cup \infty$

und dass der Bewertungssemiring idempotent ist, da

(71) $a \wedge^{\leftarrow} a = a$ für alle $a \in \{0,1\}^* \cup \infty$.

Es gelten die Distributivgesetze, da sich die Operation $\wedge^{\leftarrow}$ auf Suffixeigenschaften bezieht und diese wie die Präfixeigenschaften durch die Konkatenation erhalten bleiben:

(72) $a \cdot (b \wedge^{\leftarrow} a) = a \cdot b \wedge^{\leftarrow} a \cdot c$ und $(b \wedge^{\leftarrow} c) \cdot a = b \cdot a \wedge^{\leftarrow} c \cdot a$ für alle $a, b, c \in \{0,1\}^* \cup \infty$

Über der Menge $\{0,1\}^* \cup \infty$ ist über $\wedge^{\leftarrow}$ eine lineare Ordnung definiert:

(73) $(a \wedge^{\leftarrow} b = a) \Leftrightarrow (a \leq b)$ für alle $a, b \in \{0,1\}^* \cup \infty$

Der Semiring ist nicht kommutativ, da die Konkatenation nicht kommutativ ist, daher müssen Annahmen wie bei dem Global-Left-to-Right-Kriterium gemacht werden. Zudem ist die Berechnung der Operation $\wedge^{\leftarrow}$ nicht konstant, da sie durch die Operation $\wedge^{\rightarrow}$ definiert ist (die Komplexität der Umkehrung soll hierbei vernachlässigt werden).

Die Kriterien $\mathcal{K}_{Ex\_Match}$, $\mathcal{K}_{Ec\_Match}$, $\mathcal{K}_{Se\_Match}$ und $\mathcal{K}_{RL\_Match}$ können nun geordnet werden:

(74) $$\mathcal{K}_{\downarrow Ex\_Ec\_Se\_GRtL\_Match} = \Big\downarrow \begin{array}{l} \mathcal{K}_{Ex\_Match} \\ \mathcal{K}_{Ec\_Match} \\ \mathcal{K}_{Se\_Match} \\ \mathcal{K}_{RL\_Match} \end{array}$$

Die gewichtete Sprache, die rechte Klammerungen präferiert, wird durch *Global_Right_to_Left* erzeugt:

(75) Global_Right_to_Left(Left_Brackets,Right_Brackets) $=_{def}$
[[$\sim$ \$[Left_Brackets|Right_Brackets].o.[? $<\mathcal{K}_{RL\_Match}, 0>$]*]
[[Left_Brackets $<\mathcal{K}_{RL\_Match}, \epsilon>$]|
[Right_Brackets $<\mathcal{K}_{RL\_Match}, 1>$]]]*
[$\sim$ \$[Left_Brackets|Right_Brackets].o.[? $<\mathcal{K}_{RL\_Match}, 0>$]*]

Wenn die gewichtete reguläre Relation R([aba](→){...}//Ex_Ec_Match) durch L(Global_Right_to_Left([{],[}])) gewichtet und auf die Eingabe L([a|b]baba) angewendet wird, werden nun folgende Klammerungen und Gewichtungen vorgenommen:

(76) $(0,0,0,\epsilon)ababa$ $(0,0,0,\epsilon)bbaba$
$(3,2,0,000100)\{aba\}ba$ $(3,2,0,000001)bb\{aba\}$

$(0,0,0,\epsilon)bbaba$
$(3,2,0,000001)ab\{aba\}$

In (76) erfüllen die gewichteten Wörter $(3,2,0,000001)bb\{aba\}$ und $(3,2,0,000001)ab\{aba\}$ das Kriterium $\mathcal{K}_{\downarrow Ex\_Ec\_Se\_GRtL\_Match}$ am besten. Durch das Global-Right-to-Left-Kriterium konnten die Klammerungen disambiguiert wer-

den.

7.3.2 Berechnung und Robustheit

Die Berechnung der gewichteten Transduktoren, die eine Ebene in einer Kaskade darstellen, ist nicht sehr aufwendig. Der OCI-Operator verlangt keine Komplementierung der zu klammernden regulären Sprache, sondern lediglich eine Komposition mit der Identitätsrelation der gewichteten regulären Sprachen, die die Kriterien realisieren. Ein Transduktor, der eine Kaskade realisiert, kann daher relativ klein gehalten werden. Die aufwendigeren Operationen sind hierbei die Optimierung und Komposition der Ebenen. Hierbei wird nicht in jedem Fall ein sequenzieller gewichteter Transduktor erstellt.

Nach der Anwendung einer Kaskade auf eine Eingabe muss der beste Pfad berechnet werden. Diese Berechnung ist hierbei jedoch nicht linear. Die Komplexität des Generic-Topological-Order-Single-Source-Shortest-Distance Algorithmus ist $O(|Q| + (T_{\oplus} + T_{\otimes})|E|)$, wobei E die Menge der Ecken und $K \subseteq E \times E$ die Menge der Kanten darstellt (siehe Abschnitt 7.1.6). Die Zeitkomplexität der Operation $\otimes$, denotiert durch $T_{\otimes}$, ist bei diesem Verfahren konstant. Die Zeitkomplexität der Operation $\oplus$, denotiert durch $T_{\oplus}$, ist jedoch linear. Die Operation $\oplus$ wird durch die Zeitkomplexität der Operation $\wedge^{\leftarrow}$ bzw. $\wedge^{\rightarrow}$ des Bewertungssemirings $(\{0,1\}^* \cup \infty, \wedge^{\leftarrow}, \cdot, \infty, \epsilon)$ bzw. $(\{0,1\}^* \cup \infty, \wedge^{\rightarrow}, \cdot, \infty, \epsilon)$ dominiert. Die Zeitkomplexität der Operation $\wedge^{\leftarrow}$ bzw. $\wedge^{\rightarrow}$ ist linear und verhält sich proportional zur Tiefe des bewerteten azyklischen Graphen, da die Zeichenketten entlang eines Pfades wachsen. Die Maximaltiefe ist dabei gerade durch die Anzahl der Kanten gesetzt. Die Zeitkomplexität der Operation $\oplus$ ist daher proportional zu $|E|$. Aus diesem Grund ist hierbei die Komplexität des Generic-Topological-Order-Single-Source-Shortest-Distance Algorithmus qua-

dratisch. Aber das ändert nichts an der ursprünglichen Komplexität der Anwendung einer Kaskade, die durch die Komposition quadratisch ist.

Mit dem OCI-Operator ist es möglich, eine reguläre Relation zu erstellen, die Chunk-Typen in allen Varianten klammert. Die Eingabe wird sukzessive durch Informationen angereichert, die Analyse ist hierbei konstruktiv. Es gilt dann, die richtige Information zu extrahieren, nämlich wohlgeformte Chunks. Durch die Bestimmung des Wortes mit dem besten Gewicht ist auch hier gewährleistet, dass zu einer nicht leeren Eingabe eine Analyse nie leer ist. Das Verfahren ist wegen der Bestimmung des besten Wortes nicht mehr konstruktiv. Auf Grund der Eigenschaft des Chunking ist die Analyse letztlich robust.

7.3.3 Morphologische Disambiguierung

Mit dem Chunking mit gewichteten regulären Relationen ist es nun auch möglich, eine Eingabe zu disambiguieren. Die Disambiguierung geschieht hier implizit in den Constraints. Wohlgeformte Abfolgen müssen nicht mehr explizit beschrieben werden.

Die Klammerung von Chunks ist eindeutig, da die erschöpfendste Analyse angenommen wird, die gleichzeitig mit den wenigsten Klammern auskommt. Bei verbleibenden ambigen Klammerungen werden dann die weiter rechts bzw. weiter links stehenden Klammerungen bevorzugt. Es werden hierbei die morphologischen Lesarten herangezogen, die zu einer größtmöglichen Abdeckung der Eingabe durch Chunks führt. Wenn morphologische Varianten zu gleichen Chunks führen, können Präferenzen angegeben werden, um auch diese Ambiguitäten in der Eingabe auflösen zu können. Im Gegensatz zum Chunking mit regulären Relationen ist also eine eindeutige Eingabe nicht zwingend notwendig.

Bei diesem Verfahren wird vorausgesetzt, dass alle Wörter der Eingabe die gleiche Länge haben. Morphologische Eigenschaften zu einem Wort können jedoch ganz unterschiedliche morphologische Merkmale enthalten, was sich dann auch in der variablen Länge der Wörter in einer Eingabe widerspiegelt. Es soll aber auch hier möglich sein, dieses Verfahren anzuwenden. Daher sollen sich die Kriterien $\mathcal{K}_{Ex\_Match}$, $\mathcal{K}_{LR\_Match}$ und $\mathcal{K}_{RL\_Match}$ nicht auf alle Zeichen der Eingabe beziehen, sondern auf Wortgrenzen-Tags (@). Hierbei muss gewährleistet sein, dass vor jedem natürlichsprachlichen Wort das Tag @ steht und somit alle Wörter der Eingabe die gleiche Anzahl an Wortgrenzen-Tags enthalten.

7.3.4 Garden-Path-Effekt

Auch durch das Chunking mit gewichteten regulären Relationen kann es zu Garden-Path-Effekten kommen, da es sich hier, wie beim Chunking mit regulären Relationen, um eine gierige Disambiguierungsstrategie handelt. Jedoch werden hier erst alle Analysen berechnet. Die richtige Analyse, die zu keinem Garden-Path-Effekt führt, ist hierbei enthalten. Nur durch die Ermittlung des besten Gewichts kommt es zu einem Garden-Path-Effekt, da eine Strategie verfolgt wird, die dem Prinzp *Late Closure* ähnlich ist. Es sollten daher Kriterien herangezogen werden, die den Gardenpath-Effekt unterbinden.

7.4 Verbindung von Chunking und syntaktischem Tagging mit Gewichten

Es sollen die Vorteile des Chunking mit gewichteten regulären Relationen und des syntaktischen Taggings mit gewichteten regulären Sprachen zusammenge-

bracht werden.

Bei dem syntaktischen Tagging sind viele Constraints notwendig, um eine Eingabe zu disambiguieren. Hierbei müssen wohlgeformte Lesarten explizit aufgeführt werden. Das Chunking mit gewichteten regulären Relationen dagegen macht es auf einfachste Weise möglich, eine Eingabe zu disambiguieren. Die Disambiguierung ist nicht wie beim syntaktischen Tagging durch die Angabe von wohlgeformten Lesarten bestimmt. Eine Analyse durch syntaktisches Tagging ergibt jedoch reichhaltigere Informationen als eine Analyse durch das Chunking, da das Chunking nur lokale Dependenzbäume behandelt.

Eine Eingabe soll hier durch das Chunking mit gewichteten regulären Relationen teilweise disambiguiert werden. Danach sollen die Chunks in eine Dependenzstruktur im Sinne eines Attacher eingegliedert werden und es sollen zusätzlich syntaktische Funktionen getaggt werden, die keine Chunks betreffen. Morphologische Lesarten, die durch das Chunking mit gewichteten regulären Relationen nicht disambiguiert werden, sollen durch Constraints des syntaktischen Taggings aufgelöst werden. Hierfür soll das syntaktische Tagging mit gewichteten regulären Sprachen verwendet werden. Durch dieses Verfahren wird also eine durch Chunks gebildete Struktur mit zusätzlicher syntaktischer Information angereichert.

7.4.1 Die Eingabe

Die Eingabe wird wie beim syntaktischen Tagging mit regulären Sprachen durch eine Abfolge von natürlichsprachlichen Wörtern mit ihren morphologischen Analysen und potentiellen syntaktischen Funktionen dargestellt. Jedoch befindet sich hier vor jedem Wort das Wortgrenzen-Tag @. Nebensatzgrenzen sollen hier nicht behandelt werden; das bedeutet, dass das hier beschriebene Verfah-

ren nur auf einfache Sätze anzuwenden ist. Dieses Verfahren ist sozusagen der erste wichtige Schritt in die Richtung der Analyse komplexerer Satzstrukturen. Es sollen hier einzelne Wörter nur dann mit syntaktischen Funktionen getaggt werden, wenn sie nicht in die Analyse durch Chunks involviert sind. Wenn zum Beispiel Verben nicht durch einen Chunk-Typ behandelt werden, muss ihnen eine potentielle syntaktische Funktion zugewiesen werden. Auch funktionale Elemente, das heißt potentiell verwaiste Wörter, sollen mit potentiellen syntaktischen Funktionen versehen werden. So kann später der Bezug zu einem Chunk hergestellt werden, den ein verwaiste funktionale Element definiert.

7.4.2 Ordnung der Kriterien

Mit Hilfe des syntaktischen Taggings ist es nun möglich, Garden-Path-Effekte zu vermeiden. Hierfür soll das Kriterium $\mathcal{K}_{Tag\_Pref}$ als $\mathcal{K}_{St\_Pref}$ definiert werden. Das bedeutet, dass dieses Kriterium die Klammerungen von Chunks mitbestimmen soll. Durch das Kriterium $\mathcal{K}_{St\_Pref}$ soll vermieden werden, dass durch die gierige Strategie des Chunkings mit gewichteten regulären Relationen ein Objekt oder ähnliches im Satz fehlt.

Jedoch sollen die syntaktischen Funktionen nicht generell die Klammerungen von Chunk-Typen mitbestimmen. Daher soll das Kriterium $\mathcal{K}_{Tag\_Pref}$ als $\mathcal{K}_{We\_Pref}$ definiert werden. Dieses Kriterium soll niedriger eingeordnet sein als die Kriterien $\mathcal{K}_{Ex\_Match}$ und $\mathcal{K}_{Ec\_Match}$ und soll das Kriterium $\mathcal{K}_{Se\_Match}$ verdrängen.

Zur endgültigen Disambiguierung von Klammerungen soll hier das Kriterium $_{LR\_Match}$ verwendet werden. Die Kriterien sind nun folgendermaßen geordnet:

$$(77) \qquad \mathcal{K}_{\downarrow St\_Ex\_Ec\_We\_GLtR} = \left| \begin{array}{l} \mathcal{K}_{St\_Pref} \\ \mathcal{K}_{Ex\_Match} \\ \mathcal{K}_{Ec\_Match} \\ \mathcal{K}_{We\_Pref} \\ \mathcal{K}_{LR\_Match} \end{array} \right.$$

7.4.3 Potentielle syntaktische Funktionen von Chunks

Die Kopfmarkierungen in den Chunks können gleich bei der Erstellung der Klammerungen eingefügt werden, so muss später nicht mehr auf die innere Struktur der Chunk-Typen Bezug genommen werden. In dem folgenden Beispiel soll innerhalb eines NP-Chunks im Genitiv das letzte Nomen als Kopf getaggt werden:

(78) $[$@ . DET @DUMMY @ . N GEN SG 0:@HEAD$](\rightarrow)\{_{np}...\}_{np}$

Den Chunk-Typen sollen potentielle syntaktische Funktionen zugewiesen werden. Dies kann wie in Abschnitt (6.3.2) mit dem Conditional-Replacement-Operator realisiert werden. Es ist jedoch auch möglich, Chunk-Typen von vornherein syntaktische Funktionen zuzuweisen. Wenn eine potentielle syntaktische Funktionen von chunkinternen Merkmalen abhängt, muss so später nicht mehr auf die interne Struktur der Chunk-Typen Bezug genommen werden. Ein NP-Chunk, dessen Kopf im Genitiv steht, hat die potentielle syntaktische Funktion @GN> nicht aber die potentielle syntaktische Funktion @SUBJ. Auch hier bedeutet das Tag @DUMMY, dass keine syntaktische Funktion vergeben ist. Die Funktionen der Chunk-Typen werden hierbei bei der rechten Typ-Klammerung des OCI-Operator mit angegeben:

(79) [@ . DET @DUMMY @ . N GEN SG 0:@HEAD](→)
{$_{np}$...[}$_{np}$[@GN>|@DUMMY]]

Hierbei wird den funktionalen Elementen innerhalb der Chunk-Typen das syntaktische Tag @DUMMY zugeteilt, indem alle anderen potentiellen syntaktischen Funktionen blockiert werden. Auch den eingebetteten Chunk-Typen wird auf die gleiche Weise das syntaktische Tag @DUMMY zugeteilt.

7.4.4 Constraints

Es soll hier zwischen Initialisierung- und Kontext-Constraints unterschieden werden.

Initialisierung-Constraints geben syntaktischen Tags ein initiales Gewicht. Syntaktische Tags können so ein bestimmtes Potential in die Analyse mitbringen. Die Initialisierung-Constraints sollen mit dem Optional-Restriction-Operator realisiert werden.

Kontext-Constraints behandeln kontextabhängige Gewichtszuweisungen, wie zum Beispiel das Uniqueness Principle, PP-Attachment, Abhängigkeiten zwischen verwaisten Wörtern und den Chunks, die sie definieren, usw. Auch die Kontext-Constraints sollen mit dem Optional-Restriction-Operator realisiert werden.

Zwischen den beiden Arten von Constraints gibt es jeweils die Unterscheidung zwischen *strong* und *weak*. Das bedeutet, es wird entweder das Kriterium $\mathcal{K}_{St\_Pref}$ oder $\mathcal{K}_{We\_Pref}$ benutzt. Alle syntaktischen Tags sollen zusätzlich eine *Status-Information* beinhalten. Diese Information soll darüber Auskunft geben, ob ein syntaktisches Tag aktiv ist oder nicht. Tags, die nicht aktiv sind, sollen für Constraints transparent sein. Ob ein syntaktisches Tag aktiv, ist soll durch

die Tags *yes* und *no* dargestellt werden. *yes* bedeutet aktiv, *no* bedeutet inaktiv. Diese Information wird den syntaktischen Tags nachgestellt. Es wird auch notwendig sein syntaktische Tags zu markieren, um Kontext-Constraints zu synchronisieren. Daher soll auch das Tag *sync* als Status-Information möglich sein.

Die Status-Information kann innerhalb von Kontext-Constraints durch eine Transduktion verändert werden. Das bedeutet, dass die Kontext-Constraints nicht mehr durch gewichtete reguläre Sprachen, sondern durch gewichtetete reguläre Relationen dargestellt werden.

7.4.5 Attacher

Es soll ein Attacher aus einzelnen Constraints erstellt werden. Die Kontext-Constraints müssen miteinander komponiert werden. Hierbei ist im Gegensatz zum Schnitt die Reihenfolge der Anwendung relevant, da die Komposition nicht kommutativ ist. Die Initialisierung-Constraints werden dahingegen miteinander geschnitten. Ein Attacher *G*, der aus den Initialisierung-Constraints I_1, I_2, ...,I_n und den Kontext-Constraints C_1, C_2, ...,C_n besteht, ist folgend definiert:

(80) $G =_{def} Id(I_1 \& I_2 \& \ldots \& I_n).o.C_1.o.C_2.o. \ldots .o.C_n$

Nun soll der Attacher mit der Kaskade des Chunking mit gewichteten regulären Relationen komponiert werden:

(81) $Parser =_{def} Kaskade.o.Attacher$

Die gewichtete reguläre Relation R(Parser) kann nun auf eine Eingabe angewendet werden. Danach wird das Wort mit dem besten Gewicht ermittelt.

7.4.6 Berechnung und Robustheit

Da keine Komplementbildungen verlangt werden, werden die gewichteten Transduktoren, die aus dem Chunking mit gewichteten regulären Relationen und dem syntaktischen Tagging mit gewichteten regulären Relationen resultieren nicht sehr groß. Die Anwendung einer Kaskade mit der Constraint-Grammatik auf eine Eingabe ist hierbei wegen der Komposition und dem ermitteln des besten Pfades quadratisch.

Ein Transduktor kann sich bei einer Analyse vor der Bestimmung des besten Gewichts stark aufblähen. Es werden keine Lesarten entfernt und sukzessive Analysemöglichkeiten eingefügt. Die Analyse ist in diesem Fall konstruktiv, aber nicht monoton, da Statusinformationen verändert werden können. Um die Größe der Eingabe wärend einer Analyse kleiner zu halten, können einige Constraints wie in Abschnitt (6.1.2) realisiert werden. Hierbei müsste dann der Restriction-Operator verwendet werden. Das syntaktische Tagging mit gewichteten Transduktoren vor der Bestimmung des besten Wortes wäre dann aber nicht mehr konstruktiv.

Bei diesem Verfahren wird die Eingabe zu erst sukzessiv durch Informationen angereichert. Einerseits durch das Chunking mit gewichteten regulären Relationen und anderseits durch das syntaktische Tagging mit gewichteten regulären Relationen. Durch die Ermittlung des Wortes mit dem besten Gewicht ist dann gewährleistet, das die Analyse nicht leer ist. Gerade durch die Grundlegende Definition des Chunkings ist die Analyse robust. Es werden lokale Dependenzbäume erstellt. Diese Bäume haben potentielle syntaktische Funktionen im Satz. Lediglich durch das Gewichten dieser Funktionen wird ein lokaler Dependenzbaum in einen globalen Dependenzbaum eingebunden. Ein Scheitern einzelner Eingliederungen führt hierbei nicht zum völligen Zusammenbruch ei-

ner Analyse.

7.4.7 Garden-Path-Effekt

Bei diesem Verfahren ist es möglich, dass das syntaktische Tagging die Klammerung von Chunks beinflussen kann. Es werden durch das Chunking mit gewichteten regulären Relationen alle Klammerungsmöglichkeiten berechnet. Die Analysemöglichkeiten enthalten dann auch die Analyse, die zu keinem Garden-Path-Effekt führt, d.h. es findet eine parallele Verarbeitung aller möglichen Strukturen statt. Durch das syntaktische Tagging mit gewichteten regulären Relationen ist es nun möglich, diese korrekten Analysen zu extrahieren. Hierbei ist ein Garden-Path-Effekt nicht mehr spührbar, da keine Reanalyse notwendig ist. Garden-Path-Effekte, die aus lokalen Ambiguitäten resultieren, können hier also durch weiteren Kontext aufgelöst werden.

Eine ähnliche Technik wendet Brants in [10] an. In seinem Ansatz werden Markov-Modelle verwendet, um phrasale Kategorien und syntaktische Funktionen zuzuweisen. Hierzu werden Markov-Modelle kaskadiert, ähnlich wie kaskadierte gewichteten Transduktoren. Es ist möglich, innerhalb einer Ebene eine ambige Ausgabe zusammen mit einer Wahrscheinlichkeitsverteilung als Eingabe einer höheren Ebene zu generieren. So wird das Ergebnis in einer Ebene nicht zwingend nur von einer Ebene selbst bestimmt.

7.5 Chunk-Typen für das Deutsche

Es sollen hier NP-Chunks, PP-Chunks und AP-Chunks für das Deutsche mit gewichteten regulären Relationen entwickelt werden. Im Folgenden soll das STTS

(Stuttgart-Tübinger Tagset) verwendet werden.[26] Dieses ist in dieser Arbeit jedoch durch morphologische Eigenschaften ergänzt, die den Kasus von Nomen und Eigennamen behandeln: nom (Nominativ), akk (Akkusativ), dat (Dativ), gen (Genitiv). Diese Tags sind ausschließlich den Tags NN, NE, ART, PDAT, PIAT, PPOSAT, PWAT, PRELAT und ADJA nachgestellt. Das bedeutet, dass hier nur die Kasuskongruenz, aber nicht Geschlecht und Numerus berücksichtigt werden sollen. Alle anderen Kategorien besitzen keine weiteren morphologischen Spezifikationen. Konjunktionen haben das potentielle syntaktische Tag @CC, dem das Tag @DUMMY in Optionalität steht. Das hier verwendete Makro „.“ ist ähnlich wie in Abschnitt (6.1.2) definiert:

(82) $. =_{def} \sim \$[@]$

7.5.1 NP-Chunks

Die folgenden Definitionen beziehen sich auf den Kasus Akkusativ, sie sind aber auf die anderen Kasus übertragbar.

Funktionale Elemente in NP-Chunks sind ART, PDAT, PIAT, PPOSAT, PWAT und PRELAT:

(83) NP_Akk_Funktionales_Element $=_{def}$
@ . [[ART akk]|[PDAT akk]|[PIAT akk]|[PPOSAT akk]|[PWAT akk]|
[PRELAT akk]]@DUMMY yes

Diese funktionalen Elemente selektieren die thematischen Elemente NN und NE:

[26] www.sfs.nphil.uni-tuebingen.de/Elwis/stts/stts-guide.ps.gz

(84) NP_Akk_Thematisches_Element $=_{def}$
@ . [[NN akk]|[NE akk]]0:@HEAD 0:yes

Hierbei werden die funktionalen Elemente nach ihren thematischen Elementen flektiert. Es können hierbei auch leere funktionale Elemente angenommen werden (vgl. [16], S.109f.):

(85) dem Menschen, des Politikers, ein Mitbewerber, $Ø_{det}$ Hans

Mehrwortausdrücke wie *New York* oder *Werder Havel* werden als ein thematisches Element behandelt:

(86) NP_Akk_Thematisches_Element_Mehrwortausdruck $=_{def}$
NP_Akk_Thematisches_Element^{+}

Pronomen, PDS, PIS, PPER, PPOSS, PRELS, und PWS sind in NP-Chunks ausschließlich Komplemente von leeren funktionalen Elementen:

(87) NP_Akk_Thematisches_Element_Pronomen $=_{def}$
@ . [[PDS akk]|[PIS akk]|[PPER akk]|[PPOSS akk]|[PRELS akk]|
[PWS akk]]0:@HEAD 0:yes

Zwischen den funktionalen Elementen und den Komplementen können Adjektive stehen:

(88) Akk_Adjektive $=_{def}$
[@ . ADJA akk]$^{+}$

Diese sind durch kein funktionales Element selektiert und flektieren nach dem

thematischen Element des NP-Chunks ([16], S.19):

(89) mit grüner Tinte, die heiligen Hallen, manche vergessenen Briefe

Ein Nomen kann durch ein *Nomen invarians* spezifiziert werden. Dieses trägt den Kasus Nominativ.

(90) Nomen_Invarians $=_{def}$
@ . [[NN nom]|[NE nom]]

Das Nomen invarians stellt Vornamen, Titel und Berufsnamen dar, die einem thematischen Element vorangestellt sind, das eine Person bezeichnet und durch ein leeres funktionales Element selektiert ist ([16], S.111):

(91) Daniela Obermayer, Lehrer Lämpel, Onkel Christian

Das Nomen invarians kann aber auch Personennamen und geographische Namen sowie gewisse Fach- und Sachbezeichnungen darstellen, die einem thematischen Element nachgestellt sind ([16], S.111):

(92) der Betriebsrat Müller, des Schulfachs Mathematik, der Monat September

Im Gegensatz zum vorangestellten Nomen invarians steht das nachgestellte Nomen invarians nicht zwischen einem funktionalen Element und seinem Komplement und wird so nicht in einen Chunk integriert. Jedoch herrscht das Prinzip Chunk Inclusiveness, d. h. das nachgestellte Nomen invarians muss zu einem Chunk gruppiert werden. Für das Nomen invarians soll aber kein leeres funktionales Element angenommen werden, daher soll es zu dem vorangehenden

Chunk gruppiert werden.

Adjektive und Nomen invarians können durch Konjunktionen koordiniert werden:

(93) der weite *und* unentdeckte Weltraum, die Wüste Namib *oder* Kalahari, Vater *und* Freund Karl

Die Konjunktion stellt hierbei weder ein thematisches noch ein funktionales Element dar und erhält das syntaktische Tag @DUMMY:

(94) Akk_Adjektiv_Konjunktion $=_{def}$
$[$Akk_Adjektive @ . KON @DUMMY yes$]^*$Akk_Adjektive

(95) Nomen_Invarians_Konjunktion $=_{def}$
$[$Nomen_Invarians @ . KON @DUMMY yes$]^*$Nomen_Invarians

Es können aber auch thematische Elemente koordiniert werden:

(96) das Auto *und* Sammlerstück, die schöne Eva *und* erste Liebe, er *und* sie

Jedoch erlauben funktionale Elemente nur ein Komplement. Thematische Elemente, die durch eine koordinierende Konjunktion verbunden sind, sollen also als ein thematisches Element angesehen werden. Hierbei müssen aber die durch eine Konjunktion verbundenen thematischen Elemente als Kopf markiert sein, um einen Chunk in einen lokalen Dependenzbaum überführen zu können. Die Definition von koordinierten thematischen Elementen mit vorangestelltem Nomen invarians ist folgend definiert (hierbei wird vernachlässigt, dass der Kopf eine Person bezeichnen muss):

(97) NP_Akk_Thematisches_Element_Konjunktion_vorangestellt $=_{def}$
[[Nomen_Invarians_Konjunktion*
NP_Akk_Thematisches_Element_Mehrwortausdruck
@ . KON @DUMMY yes]*
Nomen_Invarians_Konjunktion*
NP_Akk_Thematisches_Element_Mehrwortausdruck]

und mit nachgestelltem Nomen invarians folgendermaßen:

(98) NP_Akk_Thematisches_Element_Konjunktion_Nachgestellt $=_{def}$
[[Akk_Adjektiv_Konjunktion*
NP_Akk_Thematisches_Element_Mehrwortausdruck
NP_Nomen_Invarians_Konjunktion*
@ . KON @DUMMY yes]*
Akk_Adjektiv_Konjunktion*
NP_Akk_Thematisches_Element_Mehrwortausdruck
NP_Nomen_Invarians_Konjunktion*]

und die Definition von koordinierten Pronomen folgendermaßen:

(99) NP_Akk_Thematisches_Element_Pronomen_Konjunktion $=_{def}$
[[NP_Akk_Thematisches_Element_Pronomen
@ . KON @DUMMY yes]*
NP_Akk_Thematisches_Element_Pronomen]

Bei NP-Chunks können verwaiste funktionale Elemente angenommen werden (siehe Abschnitt 2.2.5). Also hat ein NP-Chunk im Akkusativ folgende Form:

(100) NP_Akk $=_{def}$
[[(NP_Akk_Funktionales_Element)
NP_Akk_Thematisches_Element_Konjunktion_nachgestellt]|
NP_Akk_Thematisches_Element_Konjunktion_vorangestellt|
NP_Akk_Thematisches_Element_Pronomen_Konjunktion]

Die gleichen Definitionen gelten für NP-Chunks im Nominativ, Dativ und Genitiv. Lediglich der Kasus in den regulären Ausdrücken ist dem entsprechenden Kasus des Chunk-Typs angepasst.

Nun sollen die Chunk-Typen in der Universalsprache geklammert werden. Hierfür soll der OCI-Operator verwendet werden. Den NP-Chunks sollen hierbei potentielle syntaktische Funktionen zugewiesen werden. NP-Chunks können in einem Satz die Kopf-Funktionen @SUBJ, @OBJ und @I-OBJ (vgl. [15], S.58-70) und die Modifizierer-Funktionen @<GN (Genitivisches Attribut), @NOM (nominales Attribut) und @<P annehmen. Bei dem syntaktischen Tag @NOM soll unterspezifiziert bleiben, ob es sich um einen Predependenten oder Postdependenten handelt. Die Funktionen @NOM und @<P werden hier in allen Kasus angenommen. Die Vergabe der anderen syntaktischen Funktionen hängt vom Kasus eines NP-Chunks ab:

NP-Chunks im Nominativ können Subjekt des Hauptverbs sein (*Die Frau* ermordet den Ehemann.):

(101) NP_Chunk_Nom $=_{def}$
NP_Nom ($\rightarrow$)
$\{_{np}$...$[\}_{np}$[@DUMMY|@SUBJ|@NOM|@<P] yes]//Ex_Ec_Match

NP-Chunks im Akkusativ können potentielles direktes Objekt des Hauptverbs

sein (Die Frau ermordet *den Ehemann*.):

(102) NP_Chunk_Akk $=_{def}$
NP_Akk ($\rightarrow$)
$\{_{np}$...$[\}_{np}$[@DUMMY|@OBJ|@NOM|@<P] yes]//Ex_Ec_Match

NP-Chunks im Dativ können potentielles direktes oder indirektes Objekt des Hauptverbs sein (Ich leihe *dir* mein Fahrrad.; Ich vertraue *dir*.):

(103) NP_Chunk_Dat $=_{def}$
NP_Dat ($\rightarrow$)$\{_{np}$...
$[\}_{np}$[@DUMMY|@OBJ|@I-OBJ|@NOM|@<P] yes]//Ex_Ec_Match

NP-Chunks im Genitiv können potentielles direktes Objekt des Hauptverbs sein (Er beschuldigt ihn *des Verbrechens*.). Zudem können sie die Funktion eines nachgestellten genitivischen Attributs annehmen (das Haus *des Mannes*):

(104) NP_Chunk_Gen $=_{def}$
NP_Gen ($\rightarrow$)$\{_{np}$...
$[\}_{np}$[@DUMMY|@OBJ|@NOM|@<P|@<GN] yes]//Ex_Ec_Match

Vorangestellte genitivische Attribute sollen hier nicht behandelt werden. Nun können die NP-Chunks mit den verschiedenen Kasus zusammengefasst werden.

7.5.2 PP-Chunks

Es sollen PP-Chunks definiert werden. Funktionale Elemente innerhalb von PP-Chunks sind APPR und APPRART. Diese funktionalen Elemente nehmen als

Komplement eine NP, also einen NP-Chunk (in *der Stadt*). Ein leeres funktionales Element kann hier nicht angenommen werden:

(105) PP $=_{def}$
@ .[APPR|APPRART]0:@HEAD$\{_{np}\sim$ \$[$\{_{np}$|$\}_{np}$]$\}_{np}$

Es kann aber zu einem verwaisten funktionalen Element kommen (in der auf der Rückseite beschriebenen Aufgabe). PP-Chunks mit verwaistem funktionalen Element werden nicht als PP-Chunk geklammert, da sonst Gruppierungen doppelt geklammert wären (über den ... $\{_{pp}\{_{np}$Wolken$\}_{np}\}_{pp}$).

Ein PP-Chunk kann in einem Satz ein präpositionales Adverbial (Helga trinkt die Milch *in der Küche*.) (vgl. [15], S.293-299) oder ein Präpositionalattribut (das Bild *über dem Kamin*) (vgl. [15], S.256-263) sein. Demnach erhält ein PP-Chunk das potentielle Tag @<NOM-ADVL. Die Unterscheidung zwischen den syntaktischen Funktionen @<NOM und @ADVL soll hier nicht gemacht werden:

(106) PP_Chunk $=_{def}$
PP($\rightarrow$)
$\{_{PP}$...[$\}_{PP}$[@DUMMY|@<NOM-ADVL] yes]//Ex_Ec_Match

7.5.3 AP-Chunks

Es sollen AP-Chunks definiert werden. Zu AP-Chunks kann es nur kommen, wenn es im Zusammenhang mit NP-Chunks zu verwaisten funktionalen Elementen kommt (der $\{_{ap}$ große $\}_{ap}$ .... $\{_{np}$ Fisch $\}_{np}$). Das bedeutet, dass AP- und NP-Chunks in einer Ebene behandelt werden müssen.

Innerhalb von AP-Chunks werden grundsätzlich leere funktionale Elemente angenommen:

(107) AP $=_{def}$
Akk_Adjektiv_Konjunktion^{+}

Ein AP-Chunk hat die potentielle Modifizierer-Funktion @AN>:

(108) AP_CHUNK $=_{def}$
AP($\rightarrow$)$\{_{ap}$...$[\}_{ap}$[@DUMMY|@AN>] yes]//Ex_Ec_Match

7.6 Attacher für das Deutsche

Aufbauend auf den in Abschnitt (7.5) entwickelten Chunk-Typen sollen Constraints entwickelt werden, die diese Chunk-Typen in eine globale Dependenzstruktur eingliedern und Verben mit syntaktischen Funktionen versehen. Daher wird hier das STTS verwendet. Es werden hierbei grundlegende Probleme, wie verwaiste funktionale Elemente, das Uniqueness Principle und die Valenz von Verben behandelt. Es soll hier keine erschöpfende Grammatik entwickelt, jedoch grundlegende Lösungen aufgezeigt werden.

7.6.1 Initialisierung-Constraints

Initialisierung-Constraints sollen ein Potential von funktionalen Tags festsetzen. Anhand dieses Potentials kann später zwischen mehreren potentiellen syntaktischen Funktionen zu einem Wort disambiguiert werden.

Wenn ein Verb die Kopf-Funktionen @-FMAINV und @+FAUXV hat, sollte @+FAUXV mehr Potential haben, da das Uniqueness Principle herrscht und

somit ein anderes Verb die Kopf-Funktion @+FMAINV haben könnte, aber nicht die zweite Lesart mit der Kopf-Funktion @+FAUXV (der Mann darf$_{[@+FAUXV|@+FMAINV]}$ das Kleid kaufen$_{@-FMAINV}$, um die Frau küssen$_{@-FMAINV}$ zu können$_{[@-FAUXV|@-FMAINV]}$.):

(109) Initialisierung_Constraint_@+FAUXV $=_{def}$
$<1,\mathcal{K}_{We\_Pref}> \Rightarrow$ [@+FAUXV yes]_

(110) Initialisierung_Constraint_@-FAUXV $=_{def}$
$<1,\mathcal{K}_{We\_Pref}> \Rightarrow$ [@-FAUXV yes]_

Die initialen Gewichtungen sollen hier vorerst auf die Verben beschränkt bleiben.

7.6.2 Kontext-Constraints

Es sollen grundlegende Kontext-Constraints entwickelt werden. Diese beziehen sich einerseits auf Chunk-Typen und anderseits auf Verben und deren potentielle syntaktische Funktionen.

Koordination von Chunk-Typen

NP- und PP-Chunks können durch eine koordinierende Konjunktion verbunden werden. Hierbei soll nur die Koordination von genau zwei Konjunkten betrachtet werden. Zwei Chunks teilen sich in diesem Fall eine syntaktische Funktion. Hierbei muss die Status-Information einer potentiellen syntaktischen Funktion auf no gesetzt werden, da es sonst bei der Realisierung des Uniqueness Principle zu Problemen kommen würde. Das bedeutet auch, dass diese Kontext-Constraints vor denjenigen Kontext-Constraints angewendet werden müssen,

die das Uniqueness Principle realisieren. Zudem muss die Status-Information der koordinierenden Konjunktion ebenfalls auf no geändert werden, da diese von anderen Kontext-Constraints, die zum Beispiel die Koordination von Verbalphrasen behandeln, nicht mehr verwendet werden darf. Es wird hier nach dem Kriterium $\mathcal{K}_{St\_Pref}$ gewichtet, um eine gemeinsame syntaktische Funktion zu erzwingen und später mit anderen Constraints, die das Kriterium $\mathcal{K}_{St\_Pref}$ verwenden, konkurrieren zu können.

Es soll die Koordination von NP-Chunks definiert werden. Es wird hier nur die Behandlung eines NP-Chunks mit der Kopf-Funktion @SUBJ angegeben. Die NP-Chunks mit anderen syntaktischen Funktionen werden aber gleichermaßen behandelt (der Mann$_{@SUBJyes:no}$ und$_{@CCyes:no}$ der Hund$_{@SUBJyes}$):

(111) Kontext_Constraint_Koordination_NP_@SUBJ $=_{def}$
[@ und KON @CC yes:no <5,$\mathcal{K}_{St\_Pref}$>](⇒)
[}$_{np}$ @SUBJ yes:no]_[{$_{np}$∼ $[{$_{np}$|}$_{np}$]}$_{np}$@SUBJ yes]

Es soll auch die Koordination von PP-Chunks behandelt werden:

(112) Kontext_Constraint_Koordination_PP_@ADVL $=_{def}$
[@ und KON @CC yes:no <5,$\mathcal{K}_{St\_Pref}$>](⇒)
[}$_{pp}$ @<NOM-ADVL yes:no]_
[{$_{pp}$∼ $[{$_{pp}$|}$_{pp}$]}$_{pp}$@<NOM-ADVL yes]

Nominale Attribute

Nominale Attribute können ein Nomen präzisieren ([16], S.114):

(113) Bismark gilt *als großer Staatsmann*.

Als Soldat zieht der Mann in den Krieg.
Wir lieben ihn *als einen Politiker*.

Die nominalen Attribute werden mit der syntaktischen Funktion @NOM ausgezeichnet. Hierbei bleibt die Richtung des Kopfes unspezifiziert. Dies könnte mit Hinzunahme weiteren Kontextes aufgelöst werden. Hier wird lediglich geprüft, ob einem NP-Chunk die Vergleichskonjunktion *als* vorausgeht. Wenn dieser Fall auftritt, wird die Präferierung der Lesart als nominales Attribut durch das Kriterium $\mathcal{K}_{St\_Pref}$ realisiert, da dieser Fall relativ eindeutig ist:

(114) Context_Constraint_Nominales_Attribut $=_{def}$
[@ als KOKOM @CC yes:no <10,$\mathcal{K}_{St\_Pref}$>] ($\Rightarrow$)
-[{$_{np}\sim$ \$[{$_{np}$|}$_{np}$]}$_{np}$@NOM yes]

Nachgestellte genitivische Attribute

Es sollen hier nachgestellte genitivische Attribute behandelt werden:

(115) die Mündung *der Mosel*, die Gesundheit *deiner jüngsten Kinder*, der Untergang *der Welt*

Hierbei wird geprüft, ob einem NP-Chunk im Genitiv ein NP-Chunk vorangeht, auf den sich die syntaktische Funktion @<GN bezieht. Ist dies der Fall, wird die Präferierung der Lesart als genitivisches Attribut durch das Kriterium $\mathcal{K}_{We\_Pref}$ realisiert:

(116) Context_Constraint_Genitivisches_Attribut $=_{def}$
[{$_{np}\sim$ \$[{$_{np}$|}$_{np}$]}$_{np}$@ <GN yes <3,$\mathcal{K}_{We\_Pref}$>] ($\Rightarrow$)

$[\}_{np}$[@DUMMY|@SUBJ|@OBJ|@I-OBJ|@NOM|@<P|@<GN]
yes] _

Behandlung von verwaisten funktionalen Elementen

Verwaiste funktionale Elemente können bei einem NP-Chunk auftreten:

(117) die große $\{_s$auf der Rückseite beschriebene$\}_s$ Aufgabe

In (117) soll angenommen werden, dass das Verb *beschriebene* ein Partizip II ist und daher ein Nebensatz gebildet wird. Unter dieser Betrachtungsweise ist die Behandlung von verwaisten funktionalen Elemente einfacher. Es soll hier angenommen werden, dass diese Nebensätze bereits durch $\{_s$ und $\}_s$ markiert sind. Bei verwaisten funktionalen Elementen eines NP-Chunks wird hier zur Vereinfachung nicht die Kasuskongruenz geprüft. Zudem wird nicht überprüft, ob dem NP-Chunk überhaupt ein funktionales Element fehlt. Hierfür müsste ein NP-Chunk genauere Auskunft über den Kasus und seinen Status bezüglich des funktionalen Elementes geben. Es wäre in diesem Zusammenhang praktisch, den Typ-Klammerungen der Chunks morphologische Eigenschaften zuzuweisen. Hier soll aber die Behandlung von verwaisten funktionalen Elementen der NP-Chunks nur skizziert werden. Verwaiste funktionale Elemente werden mit dem Kriterium $\mathcal{K}_{St\_Pref}$ gewichtet. So wird verhindert, dass verwaiste funktionale Elemente unter einer anderen morphologischen Lesart als NP-Chunk geklammert werden:

(118) Context_Constraints_Verwaistes_Funkt_Element_NP $=_{def}$
[@ .[ARTDEF|ARTINDEF]. @DN> yes <7,$\mathcal{K}_{St\_Pref}$>]
(⇒)_[($\{_{ap} \sim$ \$[$\{_{ap}$|$\}_{ap}$]$\}_{ap}$)$\{_s \sim$ \$[$\{_s$|$\}_s$]$\}_s\{_{np}$]

Es sollen auch die verwaisten funktionalen Elemente eines PP-Chunk, der nicht geklammert wurde, behandelt werden:

(119) in der großen $\{_s$auf der Rückseite beschriebenen$\}_s$ Aufgabe

Wenn ein verwaistes funktionales Element existiert, dann soll das Komplement, ein NP-Chunk, durch das Tag @<P an das verwaiste funktionale Element gebunden werden:

(120) Context_Constraints_Verwaistes_Funkt_Element_PP $=_{def}$
[@ . [APPR|APPRART]@NOM-ADVL yes
<7,$\mathcal{K}_{St\_Pref}$>](⇒)
_[@ .[ARTDEF|ARTINDEF]. @DN> yes ($\{_{ap}\sim$ \$[$\{_{ap}$|$\}_{ap}$]$\}_{ap}$)
$\{_s\sim$ \$[$\{_s$|$\}_s$]$\}_s\{_{np}\sim$ \$[$\{_{np}$|$\}_{np}$]$\}_{np}$@<P yes]

Uniqueness Principle

Es soll die Behandlung des Uniqueness Principle anhand der Kopf-Funktion @+FMAIN aufgezeigt werden. Hierbei ist die Koordination von Verbalphrasen berücksichtigt. Der Status der Konjunktion wird hierbei auf sync geändert, damit sich die Constraints für das Uniqueness Principle für andere Kopf-Funktionen auf die gleiche Konjunktion beziehen können:

(121) Context_Constraints_Uniqueness_Principle_Sync $=_{def}$
[?* @ [[und]|[oder]]KON @CC yes:sync ?*]
(⇒)@@_@@

Nun kann das Uniqueness Principle für finite Hauptverben entwickelt werden. Dieses Prinzip wird hierbei durch den Restiction-Operator definiert. Gewichte

werden hierbei nicht zugewiesen:

(122) Context_Constraints_Uniqueness_Principle_@+FMAINV $=_{def}$
[(∼ $[@+FMAINV yes] (@+FMAINV yes) ∼ $[@+FMAINV yes]
@[[und]|[oder]]KON @CC sync)
∼ $[@+FMAINV yes] (@+FMAINV yes) ∼ $[@+FMAINV yes]]
⇒@@_@@

Das Uniqueness Principle für die anderen Kopf-Funktionen @SUBJ, @OBJ und @I-OBJ wird auf die gleiche Weise realisiert.

Reihenfolgenpräferenz

Im Deutschen gibt es im Hauptsatz eine bestimmte Reihenfolgenpräferenz der Verbargumente, @SUBJ>@I-OBJ>@OBJ:

(123) Eva$_{@SUBJ}$ gibt Hans$_{@I-OBJ}$ das Buch$_{@OBJ}$.

Diese Präferenz wird durch das Kriterium $\mathcal{K}_{We\_Pref}$ realisiert.

(124) Context_Constraints_Reihenfolgenpräferenz $=_{def}$
[?* (@SUBJ yes <1,$\mathcal{K}_{We\_Pref}$>)
?* (@iOBJ yes <1,$\mathcal{K}_{We\_Pref}$>)
?* (@OBJ yes<1,$\mathcal{K}_{We\_Pref}$>) ?*]
(⇒)@@_@@

Obligatorisches Hauptverb

Es wird angenommen, dass ein Satz ein Hauptverb enthält. Dies wird durch das Kriterium $\mathcal{K}_{St\_Pref}$ realisiert. Hierbei muss die Koordination von Verbalphrasen berücksichtigt werden, wobei sich der Constraint auf die Konjunktion mit dem Status sync bezieht:

(125) Context_Constraints_Obligatorisches_Hauptverb $=_{def}$
[($[@+FMAINV yes] <50,$\mathcal{K}_{St_Pref}$>
@[[und]|[oder]]KON @CC sync)
$[@+FMAINV yes]<50,$\mathcal{K}_{St_Pref}$>]
(⇒)@@_@@

Gleiches gilt für die Behandlung der Kopf-Funktion @-FMAIN.

Valenz von Verben

In einer Dependenzrelation eröffnet der Kopf eine Leerstelle, die vom Dependenten gefüllt wird. Es gibt im Allgemeinen zwei Arten von Leerstellen: wortartspezifische Leerstellen und wortspezifische Leerstellen. Wortartspezifische Leerstellen werden von einem Wort einer bestimmten Wortart eröffnet. Die wortartspezifischen Leerstellen betreffen zum Beispiel Chunks. Das funktionale Element eröffnet eine Leerstelle und diese wird durch das thematische Element gefüllt.

Hier interessieren die wortspezifischen Leerstellen. Diese Leerstellen sind auf einen Teil der Wörter einer Wortart beschränkt. Diese wortspezifischen Leerstellen werden auch als Valenz bezeichnet. Alle Hauptverben verlangen ein Subjekt, sie unterscheiden sich aber in ihren anderen Leerstellen; sie haben ver-

schiedene Valenzen (vgl. [13], S.8). Die wortartspezifische Subjekt-Leerstelle muss hierbei nicht behandelt werden, dahingegen sollen die wortspezifischen Leerstellen der Verben dargestellt werden. Hierfür sollen die morphologischen Merkmale *obj* und *iobj* dienen. Diese sind jedem Verb entsprechend hinzugefügt, je nachdem, ob ein Verb einwertig (keine Vergabe von *obj* oder *iobj*), zweiwertig (Vergabe von *obj*) oder dreiwertig (Vergabe von *obj* und *iobj*) ist. Eine Unterscheidung zwischen obligatorisch oder optional besetzbaren Leerstellen muss hier nicht gemacht werden. Es wird hier versucht, so weit wie möglich alle Leerstellen zu füllen. Als erstes soll die wortartspezifische Leerstelle @SUBJ von Verben behandelt werden. Hier ist zu beachten, dass nach der Konjunktion kein Subjekt mehr stehen darf:

(126) Context_Constraints_Verb_Valenz_@SUBJ $=_{def}$
[($[@SUBJ yes]<50,$\mathcal{K}_{St_Pref}$>
@[[und]|[oder]]KON @CC sync)
$\sim$ $[@SUBJ yes]]
(⇒)@@_@@

Nun sollen die wortspezifischen Leerstellen von Verben behandelt werden. Die Notwendigkeit, die Leerstellen zu füllen, wird durch das Kriterium $\mathcal{K}_{St\_Pref}$ ausgedrückt:

(127) Context_Constraints_Verb_Valenz_@OBJ $=_{def}$
[(?* (@OBJ yes <50,$\mathcal{K}_{St\_Pref}$>)
?* @ . obj [@+FMAIN|@-FMAIN] yes
?* (@OBJ yes <50,$\mathcal{K}_{St\_Pref}$>)
?* @ [[und]|[oder]] KON @CC sync)

?* (@OBJ yes <50,$\mathcal{K}_{St\_Pref}$>)
?* @ . obj [@+FMAIN|@-FMAIN] yes
?* (@OBJ yes <50,$\mathcal{K}_{St\_Pref}$>) ?*]
(⇒)@@_@@

Gleiches gilt für die Behandlung von indirekten Objekten.

7.7 Praktische Anwendung

Die Implementierung ist mit der Potsdam FSM Bibliothek *FSMlib* [23] realisiert. Diese Bibliothek ist in C++ geschrieben und macht ein komfortables Arbeiten mit Finite-State Maschinen möglich. Alle notwendigen Operationen wie Äquivalenztransformationen, rationale und kombinatorische Operationen werden unterstützt.

Der Bewertungssemiring aus Abschnitt (7.4.2) wurde in C++ implementiert und in die FSMlib integriert. Innerhalb der FSMlib ist eine Speicherung eines FSM im AT&T Format möglich. Zur Speicherung des Bewertungssemiringes ist dieses Format so abgeändert, dass die notwendigen Informationen ausgedrückt werden können (vgl. [43], S.8):

(128)

Src	Dest	In	Out	Cost
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
0	0	y	y	
0	0	z	z	
0	1	<epsilon>	{np	(0,0,1,0,1)
1	2	#	#	(0,1,0,0,0)
2	2	0	0	
2	2	1	1	
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.

98	(0,0,1,0,<epsilon>)
Final	Cost

In (128) sind die Gewichte als Quintupel dargestellt. Die Abfolge der Elemente des Quintupels entspricht der Ordnung des Bewertungssemirings.

7.7.1 Morphologie und Eingabe

Für die morphologische Analyse wird die TAGH Morphologie [19] verwendet. Hierbei handelt es sich um eine deutsche Morphologie, die auf gewichteten Transduktoren basiert.[27] Mit Hilfe dieser Morphologie wird die ambige Eingabe gebildet. Die potentiellen syntaktischen Funktionen werden dann durch einen

[27] Die Morphologie basiert auf einem Stammlexikon mit Allomorphen und einem konkatenativen Mechanismus für Flexion und Wortbildungen.

einfachen Transduktor eingefügt. Der Aufbau der Eingabe gleicht letztlich der Beschreibung in Abschnitt (7.4.1).

Um einen Bezug zu Wörtern in einem Text herzustellen, sind die Wortgrenzen in der Eingabe als Offset-Informationen realisiert. Diese Offset-Informationen beschreiben die Position eines Wortes im Text: die Zeile, den Anfangspunkt eines Wortes in einer Zeile und die Länge eines Wortes.

7.7.2 Kaskade und Attacher

Die Definitionen der Chunk-Typen und Constraints sind hauptsächlich in C++ kodiert. Ein geeigneter Compiler für die hier verwendeten komplexen Operatoren ist noch nicht implementiert. Das Alphabet ($|\Sigma|$) hat hierbei die Größe 365.

Bei der praktischen Anwendung werden nur NP-Chunks und PP-Chunks geklammert. Die Definition dieser Chunk-Typen entspricht größtenteils der Definition in Abschnitt (7.5). Köpfe werden jedoch innerhalb der Chunk-Typen nicht markiert und unbekannte Wörter werden zu einem NP-Chunk gruppiert. Der Transduktor, der die Kaskade realisiert, hat folgende Eigenschaften:

(129)

Kaskade

Zustände	Übergänge	Speicherplatz
937	24896	307 KB

Nicht alle in Abschnitt (7.6) definierten Constraints werden hier verwendet. Verwaiste funktionale Elemente werden nicht behandelt, hierzu wäre die Klammerung von Nebensätzen nötig. Auch wortspezifische Leerstellen von Verben sind nicht integriert, da zum Zeitpunkt dieser Arbeit die entsprechenden Daten fehlten. Der Transduktor, der den Attacher realisiert, hat folgende Eigenschaften:

(130)

Attacher		
Zustände	Übergänge	Speicherplatz
6175	1270647	29 MB

Die Verlagerung der Constraints für die Koordination von NP- und PP-Chunks zusammen mit den Constraints für nominale und genitivische Attribute auf den gewichteten Transduktor der Kaskade ergibt hierbei eine verbesserte Größenverteilung:

(131)

Kaskade'		
Zustände	Übergänge	Speicherplatz
3207	104960	1,7 MB

(132)

Attacher'		
Zustände	Übergänge	Speicherplatz
484	136698	2,5 MB

7.7.3 Analyse eines einfachen Satzes

Alle Wörter eines einfachen Satzes werden durch die TAGH-Morphologie analysiert. Hierbei werden nicht erkannte Wörter mit *unknown* klassifiziert. Dann wird anhand dieser analysierten Wörter die Eingabe erstellt, indem die einzelnen Analysen konkateniert werden. Auf den Ergebnisautomat, der die Eingabe der Analyse darstellt, wird erst der gewichtete Transduktor *Kaskade'* angewendet, dann der gewichtete Transduktor *Attacher'*. Daraufhin wird auf den Wertebereich des gewichteten Ergebnistransduktors der beste Pfad berechnet. Die Analyse des Satzes *Dabei könne die vorgeschlagene internationale Konferenz gute Dienste leisten*[28] ergibt folgendes Wort (Gewichte sind nicht mehr rele-

[28] Dieser Satz entstammt dem Testkorpus, der in Abschnitt 7.7.4 näher beschrieben werden wird.

vant):

```
[@@]@0/0/5@dabei[ADV]@0/6/5@könn˜en[VMFIN Person=first Nu
mber=sg Tense=pres Mood=ind][@+FAUXV][yes][{np]@0/12/3@di
e[ARTDEF Number=sg Case=nom Gender=fem][@NONE][yes]@0/16/
14@vor|ge|schlag˜n[ADJA Degree=pos Number=sg Case=nom Gen
der=* ADecl=weak]@0/31/14@inter|national[ADJA Degree=pos
Number=sg Case=nom Gender=* ADecl=weak]@0/46/9@Konferenz[
NN Gender=fem Number=sg Case=*][}np][@SUBJ][yes][{np]@0/5
6/4@gut[ADJA Degree=pos Number=sg Case=acc Gender=fem ADe
cl=strong]@0/61/7@Dienst[NN Gender=masc Number=pl Case=ac
c][}np][@OBJ][yes]@0/69/7@leist˜en[VVINF][@-FMAINV][yes]@
0/76/1@[<$.>][@@]
```

Mit Hilfe der im Wort enthaltenen Informationen (Offset, Typ-Klammern, syntaktische Tags) kann der Eingabetext formatiert werden:

(133) Dabei könne @+FAUXV {np die vorgeschlagene internationale Konferenz }np @SUBJ {np gute Dienste }np @OBJ leisten@-FMAINV .

In diesem Beispiel hat die Eingabe vor der Analyse folgende Eigenschaften:

(134)

Eingabe	
Zustände	Übergänge
357	437

Durch die Anwendung der Kaskade und des Attachers bläht sich die Eingabe auf:

(135)

Eingabe’	
Zustände	Übergänge
39443	49556

In (135) sind nun alle möglichen Analysen enthalten. In diesem gewichteten endlichen Automaten gibt es viele Zustände, von denen kein Endzustand erreicht werden kann. Diese Zustände machen hierbei sogar den größten Teil aus. Das zeigt sich, wenn diese Zustände entfernt werden:

(136)

Eingabe”	
Zustände	Übergänge
19885	26209

Bei jedem Zeichen der Eingabe können öffnende Typ-Klammern angenommen werden. Die meisten von diesen Hypothesen werden jedoch abgebrochen. Auch bei den Verfahren, die den LRLMI- oder RLLMI-Operator verwenden, tritt dieser Effekt auf. Da die aus diesen Operatoren resultierenden Transduktoren nicht unbedingt sequenziell sind, werden intern viele Hypothesen gemacht, die später abgebrochen werden. Eine Beschränkung der Annahme von Typ-Klammern auf den Beginn des Offset wäre hier durchaus denkbar. Hierdurch ließe sich die Anzahl von toten Zuständen reduzieren; das wurde hier jedoch nicht getestet.

Trotz alledem ist die hier erreichte Zustandsanzahl kein wirkliches Problem, da die Verarbeitung innerhalb der Automaten sehr effizient abläuft.

7.7.4 Ein kleines Testkorpus

Im Rahmen dieser Arbeit war eine ausführlichere Evaluation nicht möglich. Hier sollen anhand eines kleinen Korpus Problemstellungen und Lösungen des

hier entwickelten Parsingverfahrens diskutiert werden.

Für das verwendete Testkorpus wurden 50 einfache Sätze aus dem erweiterten DWDS-Korpus (1 Mrd. Tokens) mit insgesamt 697 Wörtern extrahiert. Von diesen Sätzen enthalten 46 Sätze die Lemmata „Dienst“ und „leisten“; die Lemmata kommen in 37 Sätzen als Funktionsverbgefüge (FVG) vor (vgl. [15], S.299-307).[29] Es ist zu berücksichtigen, dass das Testkorpus sehr klein und dadurch nicht unbedingt immer repräsentativ ist. Die einzelnen einfachen Sätze wurden von Hand annotiert (Gold-Standard). Der Parser wurde einzeln auf die einfachen Sätze des Testkorpus angewendet. Daraufhin wurden die Ergebnisse ausgewertet. Folgende Resultate ergaben sich für NP- und PP-Chunks:

(137)

Chunk-Typen	Precision(%)	Recall(%)
NP-Chunk	96,85	95,56
PP-Chunk	95,31	94,20

Trotz der relativ kleinen Regelbasis sind die Recall- und Precision-Werte relativ hoch. Fehlanalysen waren das Resultat von Garden-Path-Effekten, von ungenügender Behandlung von funktionalen Elementen und von fehlender Information aus der Morphologie. Hierbei waren falsche Klammerungen von NP-Chunks der hauptsächliche Grund für falsche Klammerungen von PP-Chunks:

- Ein Auflösen eines Garden-Path-Effektes war nicht möglich, da die Valenzinformation von Verben nicht berücksichtigt wurde.

- Eigennamen wurden durch die Morphologie nicht im Kasus unterschieden. Die Eigennamen, die ein genitivische Attribute darstellten, wurden

[29] Vorarbeit hierzu wurde von Sokirko geleistet, der 500 Belege mit den Lemmata „Dienst“ und „leisten“ zusammengestellt und nach FVG und Nicht-FVG klassifiziert hat (siehe [47]).

daher fälschlicherweise zu dem führenden Chunk gruppiert.

- Verwaiste funktionale Elemente wurden unter der Lesart eines Pronomens fälschlicherweise als Chunk geklammert.

Dahingegen schnitten die Resultate für die Bestimmung von syntaktischen Funktionen nicht so gut ab. Bei den Verben wurde nicht zwischen finit und infinit unterschieden, da die Finitheit in den Constraints nicht behandelt wurde und so die Ergebnisse nicht aussagekräftig gewesen wären. Die syntaktische Funktion @<NOM-ADVL wurde nicht aufgeführt, da sie bei PP-Chunks immer angenommen wurde:

(138)

Funktionen	Precision(%)	Recall(%)
@+FMAINV @-FMAINV	98,18	98,18
@+FAUXV @-FAUXV	84,61	100,00
@SUBJ	92,00	92,00
@OBJ	84,31	89,58
@I-OBJ	85,71	75,00
@<GN	92,59	75,76
@NOM	100,00	83,33

Es sollen Gründe für die Resultate zusammengetragen werden:

- @+FMAINV, @-FMAINV:
  Hauptverben wurden relativ sicher und ausgiebig erkannt. Hier kam es zu falschen Analysen, wenn zwei Verben im gleichen Satz auftraten, die

potentiell ein Hilfsverb oder das Hauptverb sein konnten, oder wenn eine Konjunktion, die zwei NP-Chunks koordiniert, fälschlicherweise als Konjunktion, die eine Verbalphrase koordiniert, interpretiert wurde.

- @+FAUXV, @-FAUXV:
  Die Recall-Werte bei der Klassifizierung sind relativ schlecht, da die Vergabe der syntaktischen Funktion nicht weiter eingeschränkt wurde. Aus diesem Grund wurden zu viele Hilfsverben vorhergesagt.

- @SUBJ:
  Die schwachen Werte für die Klassifikation resultieren einerseits daraus, dass bei den NP-Chunks die Genus- und Numerus-Kongruenz nicht überprüft wurde, und anderseits daraus, dass koordinierende Konjunktionen falsch interpretiert wurden.

- @OBJ, @-IOBJ:
  Die mangelnde Präzision und Trefferquote resultierten hauptsächlich aus der fehlenden Valenzinformation und aus der mangelnden Kongruenzüberprüfung in den NP-Chunks. So wurden die syntaktischen Funktionen @SUBJ, @OBJ und I-OBJ manchmal vermischt. Zudem kam es bei adverbialen Ausdrücken zu Problemen (zwei Jahre lang). Hier kam es zu Komplikationen mit dem Uniqueness Principle, da diese Fälle nicht durch die Constraints abgedeckt wurden.

- @<GN:
  Auf die Klassifizierung wirkten sich die mangelnde Kongruenzüberprüfung in den NP-Chunks und Eigennamen, die durch die morphologie keine Genitivinformationen enthielten, negativ aus.

- @NOM:
  Der Precision-Wert ist hier maximal, da Fälle, in denen *als* die Funktion einer Subjunktion hat, nicht vorkamen. Kopulaverben wurden in den Constraints nicht behandelt, daher ist der Recall-Wert etwas schlechter (zudem kamen nominale Attribute nicht sehr oft vor).

Aus den analysierten Sätzen konnte auf Funktionsverbgefüge geschlossen werden, wenn das Lemma „Dienst“ als direktes Objekt auftrat und das Lemma „leisten“ Hauptverb war. Wenn das Lemma „leisten“ mit dem Lemma „Dienst“ in einem NP-Chunk auftrat, wurde dies auch als FVG interpretiert. Passivkonstruktionen kamen im Zusammenhang mit den Funktionsverbgefügen nicht vor, daher musste dieser Fall nicht behandelt werden. Eine Auswertung wurde entsprechend für die 46 Sätze, die die Lemmata „Dienst“ und „leisten“ enthielten, durchgeführt:

(139)

FVG	Precision(%)	Recall(%)
Dienst-leisten	94,43	89,20

Die Werte sind besser als die Werte für die Klassifikation von direkten Objekten. Das liegt daran, dass FVG sowohl in NP-Chunks klassifiziert als auch innerhalb von PP-Chunks eindeutig ausgeschlossen werden können.

Die Fehler in den Analysen lassen sich beheben und sind nicht durch das Parsingverfahren bedingt, sondern entspringen einer rudimentären Grammatik. Es liegt jetzt daran, eine adäquate Grammatik zu entwickeln, die aus einer Kaskade und einem Attacher besteht.

8 Schlussbemerkungen

8.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde ein Verfahren entwickelt, um innerhalb von Sätzen Gruppierungen zu bilden und syntaktische Funktionen zu vergeben. Dieses Verfahren stützt sich auf die linguistischen Theorien des Chunking und des syntaktischen Tagging. Beide linguistischen Ansätze wurden hier eingeführt und unter der Betrachtung von lokalen und globalen Dependenzbäumen unter bestimmten Einschränkungen zusammengebracht. Darauf aufbauend wurden Ansätze diskutiert, die das Chunking, das syntaktische Tagging und deren Symbiose mit endlichen Automaten und Transduktoren realisieren. Ausgehend von den dort auftretenden Problematiken wurde eine Realisierung mit gewichteten endlichen Automaten und Transduktoren entwickelt. Es wurde gezeigt, dass sich mit diesen Maschinen die Probleme der Berechnung, der Robustheit, der ambigen Eingabe und des Garden-Path-Effekts lösen lassen. Eine Analyse wird hierbei nicht nur durch lineare Abfolgen bestimmt, sondern auch durch die Bewertung von Alternativen. Um diese Bewertung zu realisieren, wurde ein Bewertungssemiring entwickelt, der mehrere Kriterien nach ihrer Relevanz geordnet betrachten kann. Auf diese Weise konnten das Chunking und das syntaktische Tagging endgültig zusammengeführt werden. Die Erprobung des Verfahrens an einem kleinen Testkorpus ergab trotz einer Grammatik mit kleiner Regelbasis relativ gute Ergebnisse. Entstandene Analysefehler ließen sich hierbei auf eine unvollständige Regelbasis zurückführen.

8.2 Ausblick

Das hier entwickelte Verfahren lässt viel Freiraum für Ergänzungen:

- Die in dieser Arbeit verwendeten regulären Ausdrücke wurden in C++ kodiert. Die Entwicklung eines Compilers für die Operatoren sollte das Entwerfen einer Grammatik auch für Linguisten möglich machen.

- Die Typ-Klammern von Chunks könnten um Merkmale erweitert werden. So könnten sich die Kontext-Constraints einfacher auf chunk-interne Informationen beziehen. So wäre das Überprüfen der Kongruenz zwischen einem verwaisten funktionalen Element und seinem Komplement bequem zu realisieren. Zudem könnte leicht festgestellt werden, ob einem Chunk überhaupt das funktionale Element fehlt.

- Nebensätze könnten einerseits durch Kontext-Constraints oder andererseits durch den OCI-Operator realisiert werden. Hier wäre auch ein Mittelweg denkbar; der OCI-Operator sollte hierbei um Kontextangaben erweitert werden. So würden die Sätze bottom-up aufgebaut werden. Es wäre hierbei möglich, in Zwischenschritten das beste Wort zu ermitteln, um die Effizienz zu steigern. Jedoch könnte es dadurch zu falschen Analysen kommen. Es sollte besser ein Bewertungssemiring entwickelt werden, der Ambiguitäten innerhalb von Nebensätzen auflösen kann und Ambiguitäten außerhalb von Satzklammerungen belässt. Hierfür müsste jedoch ein *k-shortest-distance Algorithmus* verwendet werden. Der Begriff des Bewertungssemirings müsste dann um die k-Abgeschlossenheit erweitert werden.

- In Zusammenhang mit komplexeren Sätzen wäre das Taggen von rhetorischen Relationen denkbar. Über rhetorische Tags könnte eine Diskursstruktur über mehrere Sätze erstellt werden. Auch hierfür müsste ein neuer Semiring entwickelt werden.

- Es wäre möglich, diesen regelbasierten Ansatz durch statistische Informationen zu ergänzen. So könnten Bigramminformationen durch einen Bewertungssemiring behandelt und zur weiteren Disambiguierung verwendet werden.

Eine Verfeinerung der Bewertungssemiringe und der Regelbasis als auch Tests an größeren Korpora mit einer erschöpfenderen Auswertung stellen letztlich ein interessantes Projekt für die Zukunft dar.

A Syntaktische Funktionen

@+FAUXV	finites Hilfsverb
@-FAUXV	infinites Hilfsverb
@+FMAINV	finites Hauptverb
@-FMAINV	infinites Hauptverb
@SUBJ	Subjekt
@OBJ	direktes Objekt
@I-OBJ	indirektes Objekt
@ADVL	adverbialer Ausdruck
@<NOM	postnominales Attribut
@GN>	vorangestelltes genitivisches Attribut
@<GN	nachgestelltes genitivisches Attribut
@AN>	pränominales Adjektiv
@DN>	Determinator
@<P	Komplement einer Präposition
@CC	Konjunktion
@<NOM-ADVL	postnominales Attribut oder Adverbial
@NOM	nominales Attribut
@HEAD	syntaktischer Kopf eines Chunks
@DUMMY	ohne syntaktische Funktion

Literaturverzeichnis

[1] Steven Abney, *The English Noun Phrase in Its Sentential Aspect*, unveröffentlichte Doktorarbeit, MIT, Cambridge, Massachusetts, 1987. <www.ai.uga.edu/ftplib/ai-reports/ai199401.pdf>

[2] Steven Abney, „Rapid Incremental Parsing with Repair“, in: *Proceedings of the 6th New OED Conference*, University of Waterloo, Ontario, 1990.

[3] Steven Abney, „Syntactic Affixation and Performance Structures“, in: Denis Bouchard und Katherine Leffel (Hrsg.), *Views on Phrase Structure*, Kluwer Academic Publishers: Dordrecht, 1990.

[4] Steven Abney, „Parsing by Chunks“, in: Robert C. Berwick, Steven P. Abney, und Carol Tenny (Hrsg.), *Principle-Based Parsing: Computation and Psycholinguistics*, S.257-278, Kluwer Academic Publishers: Dordrecht, 1991.

[5] Steven Abney, „Chunks and dependencies: Bringing processing evidence to bear on syntax“, in: Jennifer Cole und Georgia Green und Jerry Morgan (Hrsg.), *Computational Linguistics and the Foundations of Linguistic Theory*, S.145-164, CSLI, 1995.

[6] Steven Abney, „Partial Parsing via finite-state cascades“, in: *Proceedings of the ESSLI workshop on robust parsing*, Prag, 1996.

[7] Salah Aït-Mokhtar und Jean-Pierre Chanod, „Incremental finite state parsing“, in: *ANLP'97*, 1997.

[8] Stefano Bistarelli, *Semirings for Soft Constraint Solving and Programming*, Lecture Notes in Computer Science Bd.2962, Springer: Berlin, 2004.

[9] Robert D. Borsley, *Syntax-Theorie. Ein zusammengefaßter Zugang*, deutsche Bearbeitung von Peter Suchsland, Niemeyer: Tübingen, 1997.

[10] Thorsten Brants, *Tagging and Parsing with Cascaded Markov Models – Automation of Corpus Annotation*, Saarbrücken Dissertations In Computational Linguistics and Language Technology, Bd.6, 1999.

[11] Michael A. Covington, „An Empirically Motivated Reinterpretation of Dependency Grammar“, AI-1994-01, 1994. <www.ai.uga.edu/ftplib/ai-reports/ai199401.pdf>

[12] Michael A. Covington, „A Fundamental Algorithm for Dependency Parsing“, 2000. <www.ai.uga.edu/ftplib/ai-reports/ai199401.pdf>

[13] Ricarda Dormeyer, *Syntaxanalyse auf der Basis der Dependenzgrammatik*, Logos Verlag: Berlin, 2004.

[14] David Elworthy, „A Finite-State Parser with Dependency Structure Output“, in: *Proceedings of International Workshop on Parsing Technologies*, 2000.

[15] Peter Eisenberg, *Grundriss der deutschen Grammatik Band 2: Der Satz*, Metzler: Stuttgart, 1999.

[16] Ulrich Engel, *Kurze Grammatik der deutschen Sprache*, ludicium: München, 2002.

[17] Gisbert Fanselow und Sascha W. Felix, *Sprachtheorie II: Die Rektions- und Bindungstheorie*, UTB Francke: Tübingen, 1993.

[18] Lyn Frazier und Charles Clifton, Jr., *Construal*, MIT Press: Cambridge, Massachusetts, 1996.

[19] Alexander Geyken und Thomas Hanneforth, *TAGH: a complete Morphology for German based on weighted Finite State Automaton*, eingereicht bei FSMNLP 2005, 2005.

[20] Gregory Grefenstette, „Light parsing as finite state filtering“, in: *Workshop on Extended finite state models of language*, ECAI'96, Budapest, Ungarn, 1996.

[21] Hans van Halteren (Hrsg.), *Syntactic Wordclass Tagging*, Text, speech and language technology series, Bd. 9, Kluwer Academic Publishers: Dordrecht, 1999.

[22] Thomas Hanneforth, *Longest-match recognition with weighted automata*, eingereicht bei FSMNLP 2005, 2005.

[23] Thomas Hanneforth, *FSMlib: C++ library for finite state device operations*, unveröffentlichte Entwicklungsversion, 2005.

[24] Udo Hebisch und Hanns Joachim Weinert, *Halbringe*, Teubner: Stuttgart, 1993.

[25] John E. Hopcroft und Jeffrey D. Ullman, *Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie*, Addison-Wesley (Deutschland): Bonn/ München, 1990.

[26] Aravind K. Joshi und Philip D. Hopely, „A parser from antiquity“, in: *Extended Finite State Models of Language*, Andras Kornai (Hrsg.), Cambridge University Press: Cambridge, S.6-15, 1999.

[27] Ronald M. Kaplan und Martin Kay, „Regular models of phonological rule systems“, in: *Computational Linguistics*, 20(3), S.331-378, 1994.

[28] Fred Karlsson, “Constraint grammar as a framework for parsing running text“, in: *Proceedings of the 13th International Conference on Computational Linguistics (COLING-90)*, Bd. III, S.168-173, Helsinki, 1990.

[29] Fred Karlsson, Atro Voutilainen, Juha Heikkilä und Arto Anttila (Hrsg.), *Constraint Grammar — A language-independent system for parsing unrestricted text*, Mouton de Gruyter: Berlin/ New York, 1995.

[30] Lauri Karttunen, „The Replace Operator“, in: *Meeting of the Association for Computational Linguistics*, S.16-23, 1995.

[31] Lauri Karttunen, „Directed Replacement“, in *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Compu-*

tational Linguistics, Aravind Joshi und Martha Palmer (Hrsg.), Morgan Kaufmann Publishers: San Francisco, S.108-115, 1996.

[32] Lauri Karttunen, J.-P. Chanod, Gregory Grefenstette und Anne Schiller, „Regular expressions for language engineering“, in: *Natural Language Engineering*, 2(4), S.305-238, 1996.

[33] A. Kempe und Lauri Karttunen, „Parallel replacement in finite-state calculus“, in: *COLING-96*, Kopenhagen, 1996.

[34] Kimmo Koskenniemi, *Two-level Morphology: A General Computational Model for Word-Form Recognition and Production*, Publication No. 11, Department of General Linguistics, University of Helsinki, 1983.

[35] Kimmo Koskenniemi, „Finite-state parsing and disambiguation“, in: H. Karlgren (Hrsg.), *COLING-90. Papers presented to the 13th International Conference on Computational Linguistics*, Helsinki, Bd.2, S.229-232, 1990.

[36] Kimmo Koskenniemi und Pasi Tapanainen und Atro Voutilainen, “Compiling and using finite-state syntactic rules“, in: *Proceedings of the fifteenth International Conference on Computational Linguistics. COLING-92*, Nantes, France, Bd. I, S.156-162, 1992.

[37] Werner Kuich und Arto Salomaa, *Semirings, automata, languages*, Springer-Verlag: Berlin/ Heidelberg/ New York/ Tokyo, 1985.

[38] Beáta Megyesi und Sara Rydin, „Towards a Finite-State Parser for Swedish“, in: *Proceedings of NoDaLiDa 99*, 1999.

[39] Mehryar Mohri, „Finite-State Transducers in Language and Speech Processing“, in: *Computational Linguistics* 23, S.269-311, 1997.

[40] Mehryar Mohri und Richard Sproat, „An Efficient Compiler for Weighted Rewrite Rules“, in: *Meeting of the Association for Computational Linguistics*, S.231-238, 1996.

[41] Mehryar Mohri, „Edit-Distance of Weighted Automata“, in: Jean-Marc Champarnaud und Denis Maurel (Hrsg.), *Seventh International Conference, CIAA 2002*, 2002.

[42] Mehryar Mohri, „Semiring Frameworks and Algorithms for Shortest-Distance Problems“, in: *Journal of Automata, Languages and Combinatorics*, 7(3), S.321-350, 2002.

[43] Mehryar Mohri und Michael Riley, „Weighted Finite-State Transducer in Speech Recognition Part I. Mathematical Foundation and Algorithms“, in: *International Conference on Spoken Language Processing 2002 (ICSLP ’02)*, Denver, 2002.

[44] Oflazer Kemal, „Dependency Parsing with an Extended Finite State Approach“, in: *Proceedings of the ACL’99*, College Park, Maryland, 1999.

[45] Emmanuel Roche und Yves Schabes, „Introduction to finite-state devices in natural language processing“, Mit-

subishi Electric Research Laboratories, TR-96-13, 1996. <www.merl.com/reports/docs/TR96-13.pdf>

[46] Emmanuel Roche und Yves Schabes (Hrsg.), *Finite-State Language Processing*, MIT Press: Cambridge, Massachusetts, 1997.

[47] Alexey Sokirko, *Ein verbspezifischer Parser zur Unterstützung der lexikographischen Analyse von Nominalisierungsverbgefügen*, unveröffentlichtes Manuskript, 2004.

[48] Atro Voutilainen, „Designing a (Finite-State) Parsing Grammar“, in: Emmanuel Roche und Yves Schabes, *Applying a Finite-State Intersection Grammar*, MIT Press: Cambridge, Massachusetts, Kap. 10, S.311-327, 1997.

[49] Atro Voutilainen und Pasi Tapanainen, „Ambiguity resolution in a reductionistic parser“, in: Proceedings of EACL’93, Utrecht, Holland, 1993.

[50] Atro Voutilainen, „Designing a (Finite-State) Parsing Grammar“, in: Emmanuel Roche und Yves Schabes, *Finite-State Language Processing*, MIT Press: Cambridge, Massachusetts, Kap. 9, S.283-310, 1997.